ROS LAB MANUAL

Dr. Michael George

Dr. Michael George

1 ROS Installation

1.1 Virtual Environment

The virtual environment can be created using VMware Player. We will be using the virtual machine image created by Mathworks which has ROS 2 Humble desktop nstallation, ROS Noetic desktop installation, Gazebo robot simulator 11, and Example Gazebo worlds for a simulated TurtleBot 3. See here about ROS distributions (`http://wiki.ros.org/Distributions`).

The following steps may be used for setup:

1. Install VMware Player downloaded from the internet.
2. Download the archive containing the virtual machine (from `https://ssd.mathworks.com/supportfiles/ros/virtual_machines/v10/ros_noetic_humble_gazebov11_linux_win_v2.zip`)
3. Decompress the archive to a location on your hard drive.
4. Start VMware Player.
5. In VMware Player, press Open a Virtual Machine.
6. Browse to the location of the Ubuntu image, select the vmx file and press OK.
7. The virtual machine is now added to your library.
8. In VMware Player, start the virtual machine.
9. Press I copied it if a window opens that asks if you copied or moved the virtual machine.

1.1.1 Troubleshoot

- To run the virtual machine, your processor's virtualization extensions must be enabled in the BIOS (see this `https://www.howtogeek.com/213795/how-to-enable-intel-vt-x-in-your-computers-bios-or-uefi-firmware/` for more information).
- By default, the virtual machine uses 2 CPU cores and allocates a maximum of 4,096 MB of RAM. If your computer does not support these default settings, you must modify the virtual machine settings before starting it.

1.2 Visual Studio (VS) Code Installation

The following steps may be used to install and setup VS Code for ROS based development:

1. The VS code debian package (.deb file) may be downloaded from `https://code.visualstudio.com/Download`.
2. Navigate to the location of the above .deb file and open terminal at this location and type the following:

   ```
   $ sudo dpkg -i 'packagename'
   ```

   The 'packagename' should be the full file name of the above .deb file along with the .deb file extension.
3. Enter your password(actually you can't see your password onscreen but it's there just type it and press Enter).
4. Now the package is being installed.
5. Open VS Code and from the extensions tab on the left, search for ROS extension published by Microsoft.
6. Install this extension and new files can be created from File option.

2 Environment Setup

The follwing steps needs to be undertaken to setup the environment:

1. To use the ROS commands (like roscore, rosrun e.t.c.) we need to type the following command in the terminal that we need to run ROS commands:

Dr Michael George

```
$ source /opt/ros/noetic/setup.bash
```

Alternatively we can add the above command at the end of the .bashrc file using a text editor like gedit,

```
$ gedit ~/.bashrc
```

Adding the source command in the .bashrc file once only is necessary to use the ROS commands in all the terminal windows opened after this addition.

2. Next, we create a workspace (called experiments_ws) where our experiment packages will reside:

```
$ mkdir -p ~/experiments_ws/src
```

3. We change the directory,

```
$ cd ~/experiments_ws/src
```

4. Initialise the workspace with CMakeLists.txt (which describes the build project structure),

```
$ catkin_init_workspace
```

5. To create the build folder (holds libraries and executables when using C++) and devel folder (contains the setup.bash file that needs to be sourced to use the workspace packages) we need to run catkin_make from the workspace folder,

```
$ cd ~/experiments_ws
$ catkin_make
```

6. Similar to first step we can source the setup.bash file in the devel folder within the .bashrc file, so add

```
$ source ~/experiments_ws/devel/setup.bash
```

to the .bashrc file.

7. Next we need to add packages to the src directory of the workspace,

```
$ cd ~/experiments_ws/src
$ catkin_create_pkg experiment0 rospy
```

Dr. Michael George

Here we are creating a new package called experiment0, which depends on rospy package. This creates a folder named experiment0 under the src directory of the workspace. The experiment0 folder would contain its own CMakeLists.txt, src directory, and a package.xml file (which contains metadata like build dependencies, package name, author name, package description related to the new package).

8. We would place the python file (e.g. exp0.py) with our code within the src directory of the package. To run this code we do,

```
$ rosrun experiment0 exp0.py
```

3 Turtlebot

TurtleBot is a ROS standard platform robot. Turtle is derived from the Turtle robot, which was driven by the educational computer programming language Logo in 1967. In addition, the turtlesim node, which first appears in the basic tutorial of ROS, is a program that mimics the command system of the Logo turtle program. It is also used to create the Turtle icon as a symbol of ROS. The nine dots used in the ROS logo derived from the back shell of the turtle. TurtleBot, which originated from the Turtle of Logo, is designed to easily teach people who are new to ROS through TurtleBot as well as to teach computer programming language using Logo. Since then TurtleBot has become the standard platform of ROS, which is the most popular platform among developers and students.

There are 3 versions of the TurtleBot model. TurtleBot1 was developed by Tully (Platform Manager at Open Robotics) and Melonee (CEO of Fetch Robotics) from Willow Garage on top of the iRobot's Roomba-based research robot, Create, for ROS deployment. It was developed in 2010 and has been on sale since 2011. In 2012, TurtleBot2 was developed by Yujin Robot based on the research robot, iClebo Kobuki. In 2017, TurtleBot3 was developed with features to supplement the lacking functions of its predecessors, and the demands of users. The TurtleBot3 adopts ROBOTIS smart actuator DYNAMIXEL for driving.

TurtleBot3 is a small, affordable, programmable, ROS-based mobile robot for use in education, research, hobby, and product prototyping. The goal of TurtleBot3 is to dramatically reduce the size of the platform and lower the price without having to sacrifice its functionality and quality, while at the same time offering expandability. The TurtleBot3 can be customized into various ways depending on how you reconstruct the mechanical parts and use optional parts such as the computer and sensor. In addition, TurtleBot3

Dr. Michael George

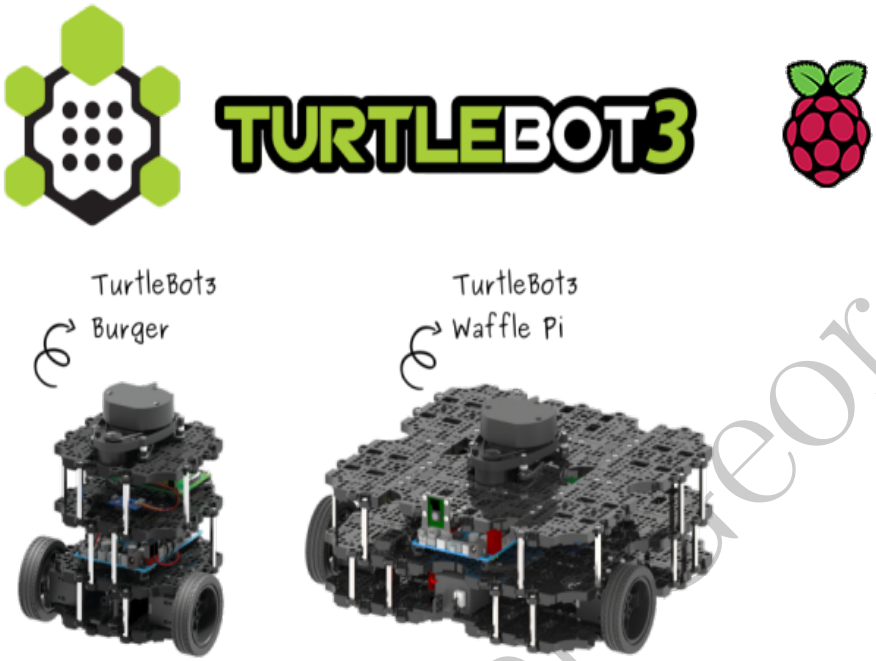

Figure 1: The Turtlebot 3

is evolved with cost-effective and small-sized SBC that is suitable for robust embedded system, 360 degree distance sensor and 3D printing technology.

The TurtleBot3's core technology is SLAM, Navigation and Manipulation, making it suitable for home service robots. The TurtleBot can run SLAM(simultaneous localization and mapping) algorithms to build a map and can drive around your room. Also, it can be controlled remotely from a laptop, joypad or Android-based smart phone. The TurtleBot can also follow a person's legs as they walk in a room. Also the TurtleBot3 can be used as a mobile manipulator capable of manipulating an object by attaching a manipulator like OpenMANIPULATOR.

4 Gazebo

Robot simulation is an essential tool in every roboticist's toolbox. A well-designed simulator makes it possible to rapidly test algorithms, design robots, perform regression testing, and train AI system using realistic scenarios. Gazebo offers the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments. At your fingertips is a

Dr. Michael George

robust physics engine, high-quality graphics, and convenient programmatic and graphical interfaces. Gazebo is free with a vibrant community.

ROS integrates closely with Gazebo through the gazebo_ros package. This package provides a Gazebo plugin module that allows bidirectional communication between Gazebo and ROS. Simulated sensor and physics data can stream from Gazebo to ROS, and actuator commands can stream from ROS back to Gazebo. In fact, by choosing consistent names and data types for these data streams, it is possible for Gazebo to exactly match the ROS API of a robot. When this is achieved, all of the robot software above the device-driver level can be run identically both on the real robot, and (after parameter tuning) in the simulator.

5 rviz

rviz stands for ROS visualization. It is a general-purpose 3D visualization environment for robots, sensors, and algorithms. Like most ROS tools, it can be used for any robot and rapidly configured for a particular application.

6 Experiment 0

We will try to simulate a turtlebot3 (waffle) moving in its environment. The following steps may be undertaken:

1. Open a terminal and set the turtlebot type:

   ```
   $ export TURTLEBOT3_MODEL=waffle
   ```

2. In same terminal launch the gazebo simulation

   ```
   $ roslaunch turtlebot3_gazebo turtlebot3_world.launch
   ```

   This should open up gazebo with turtlebot3 (waffle) in a simulated environment.

3. Look at the topics by opening a new terminal and entering:

   ```
   $ rostopic list
   ```

4. Topics of interest include cmd_vel topic, scan topic, camera topic etc.

5. For now let us look at message type published by cmd_vel topic:

   ```
   $ rostopic info /cmd_vel
   ```

Dr. Michael George

We see that the message is Twist message from geometry_msgs package.

6. Lets look at this message in more detail:

   ```
   $ rosmsg info geometry_msgs/Twist
   ```

   We can see that this message can set linear and angular velocities along x,y, and z axes.

7. Let us try to move the turtlebot, we can publish a Twist message on the cmd_vel topic:

   ```
   $ rostopic pub /cmd_vel geometry_msgs/Twist ''linear: .."
   ```

   We see that the Turtlebot will move in its environment as per the velocities set.

8. The Plotting utility can be used to graph the variations in pose, velocity etc.

9. Open a new terminal and set the turtlebot type:

   ```
   $ export TURTLEBOT3_MODEL=waffle
   ```

10. In same terminal launch the rviz

    ```
    $ roslaunch turtlebot3_gazebo turtlebot3_gazebo_rviz.launch
    ```

    This should open rviz where the laser scan range and camera output can be visualised.

7 Experiments

7.1 Experiment 1

Simulate the turtlebot robot that alternates between driving forward and stopping.

1. Create workspace experiments_ws as per first step of section 2, add experiment1 package with depend packages as rospy and geometry_msgs.

2. Then in the src folder of the experiment1 package, add the python code as exp1.py.

Dr. Michael George

```
#!/usr/bin/env python3
import rospy
from geometry_msgs.msg import Twist

rospy.init_node('forward_stop')
cmd_vel_pub = rospy.Publisher('cmd_vel',Twist,queue_size=1)

forward_twist = Twist()
forward_twist.linear.x = 0.2

stop_twist = Twist()

rate = rospy.Rate(0.2) #in hertz

while not rospy.is_shutdown():
    cmd_vel_pub.publish(forward_twist)
    rate.sleep()
    cmd_vel_pub.publish(stop_twist)
    rate.sleep()
```

3. Make the above code by running within the src directory of experiment1 package,

   ```
   $ chmod +x exp1.py
   ```

4. Open a terminal and set the turtlebot type:

   ```
   $ export TURTLEBOT3_MODEL=waffle
   ```

5. In same terminal launch the gazebo simulation

   ```
   $ roslaunch turtlebot3_gazebo turtlebot3_world.launch
   ```

6. Then execute the python code with,

   ```
   $ rosrun experiment1 exp1.py
   ```

7.2 Experiment 2

Simulate the turtlebot robot that finds the distance to an obstacle in front of the robot.

Dr. Michael George

1. The turtlebot has laser scanner that gives range to the nearest obstacles, this is available under the scan topic. The information on this can be found using,

   ```
   $ rostopic info /scan
   ```

2. The scan topic is published with LaserScan message of sensor_msgs package, more details can be found using,

   ```
   $ rosmsg info sensor_msgs/LaserScan
   ```

3. The LaserScan message has a ranges variable which is a vector of float values representing the distance to the obstacles in all directions. The distance to the obstacle in front of the robot can be found by looking at the middle indexed ranges value.

4. Create workspace experiments_ws as per first step of section 2, add experiment2 package with depend packages as rospy and sensor_msgs.

5. Then in the src folder of the experiment2 package, add the python code as exp2.py.

```
#!/usr/bin/env python3
import rospy
from sensor_msgs.msg import LaserScan

def scan_callback(msg):
    range_ahead = msg.ranges[int(len(msg.ranges)/2)]
    print("range ahead: %0.1f"%range_ahead)

rospy.init_node('range_ahead')
scan_sub = rospy.Subscriber('scan',LaserScan,scan_callback)
rospy.spin()
```

7.3 Experiment 3

Simulate the turtlebot robot that wanders the environment avoiding obstacles.

1. Create workspace experiments_ws as per first step of section 2, add experiment3 package with depend packages as rospy and geometry_msgs.

2. Then in the src folder of the experiment3 package, add the python code as exp3.py.

Dr. Michael George

7.4 Experiment 4

Simulate the turtlebot robot that moves according to keyboard teleoperation commands.

1. Create workspace experiments_ws as per first step of section 2, add experiment4 package with depend packages as rospy, geometry_msgs, std_msgs, sys, select, tty, termios, and math.
2. Then in the src folder of the experiment4 package, add the python code as exp4.py.

7.5 Experiment 5

Simulate the turtlebot robot that follows a line.

1. Create workspace experiments_ws as per first step of section 2, add experiment5 package with depend packages as rospy, geometry_msgs, sensor_msgs, cv2, cv_bridge, numpy.
2. Then in the src folder of the experiment5 package, add the python code.
3. Start the gazebo with the turtlebot3 in the world environment,

   ```
   $ roslaunch turtlebot3_gazebo turtlebot3_world.launch
   ```

4. In another terminal look at the topics including camera related topics,

   ```
   $ rostopic list
   ```

5. Create follower.py in the src folder of the package and make it executable (using chmod). Then run the code,

   ```
   $ rosrun experiment5 follower.py
   ```

   Ensure the node is up using rosnode list command. Also look at the connections to this node using rosnode info. The frame rate of the image stream from turtlebot can be verified using rostipic hz.

6. Create follower_opencv.py in the src folder of the package and make it executable (using chmod). Then run the code,

   ```
   $ rosrun experiment5 follower_opencv.py
   ```

   The cv_bridge package converts between the sensor_msgs/Image messages and the objects used OpenCV.

Dr. Michael George

7. We need to create a 'worlds' folder and 'models' folder within the package directory. In the 'worlds' folder we need to add the 'yellow_line.world' file that sets the scene for the simulation. The scene will include the ground model and the course model. The course model will be defined in the 'models' folder through a 'course.material' and 'course.png' file.

8. Start the gazebo with the turtlebot3 in the yellow line world environment,

```
$ roslaunch experiment5 yellow_line_world.launch
```

9. Create follower_color_filter.py in the src folder of the package and make it executable (using chmod). Then run the code,

```
$ rosrun experiment5 follower_color_filter.py
```

   The HSV values can be calibrated in the lower and upper mask values to filter out the yellow track. The mask output of the track is displayed in the opencv window.

10. Create follower_line_finder.py in the src folder of the package and make it executable (using chmod). Then run the code,

```
$ rosrun experiment5 follower_line_finder.py
```

    We are isolating a thin sliver of the yellow track and finding the centroid of the blob (marked by the red dot). This will be used in the next step to control the robot.

11. Create follower_p.py in the src folder of the package and make it executable (using chmod). Then run the code,

```
$ rosrun experiment5 follower_p.py
```

    The red dot marks the center of the thin sliver of yellow track. The error value is calculated as the difference between this center and the middle point of the thin sliver overall. The angular z velocity is set to be proportional to this error, so that we steer in the direction of the track.

7.6 Experiment 6

Create a new robotic arm.

1. Create workspace experiments_ws as per first step of section 2, add cougarbot package with depend package as rospy.

Dr. Michael George

2. Then in the package folder, add the urdf file.

3. The robot can be viewed in rviz using,

```
$ roslaunch urdf_tutorial display.launch model:=cougarbot.urdf
```

4. The controllers.yaml file can be located under the package folder.

5. The launch file can be located within launch folder under the package folder.

6. The arm can be viewed in Gazebo using,

```
$ roslaunch cougarbot cougarbot.launch
```

7. Move the robotic arm by publishing onto the /arm_controller/command topic and visualise in rviz.

8. The joint state positions can be plotted using rqt_plot.

7.7 Experiment 7

Simulate a new robotic arm to move as per plan.

1. Install the required packages using the following commands:

```
$ sudo apt-get install ros-noetic-moveit-ros-planning
$ sudo apt-get install ros-noetic-moveit-visual-tools
$ sudo apt-get install ros-noetic-moveit
$ sudo apt-get update
$ sudo apt-get upgrade
```

2. Use the moveit setup assistant to plan and execute the arm motion.

Dr. Michael George