MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

DEPARTMENT OF MECHATRONICS ENGINEERING

ROBOTICS LAB -II MANUAL (MTE 3162)

5th Semester B.Tech (Mechatronics Engineering)

Prepared By	Dr. Asha C S Mrs. Pooja Nag Mrs. Vibha Damodara K	Original Manual
Approved By	Dr. Chandrashekhar Bhat. (HoD- Department of Mechatronics Engineering)	

MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

DEPARTMENT OF MECHATRONICS ENGINEERING

ROBOTICS LAB -II MANUAL (MTE 3162)

5th Semester B.Tech (Mechatronics Engineering)

NAME: _____

REG NO: _____

ROLL NO: _____

VISION OF THE MECHATRONICS ENGINERING DEPARTMENT

Excellence in Mechatronics Education through Innovation and Team Work.

MISSION OF THE MECHATRONICS ENGINEERING DEPARTMENT

Educate students professionally to face societal challenges by providing a healthy learning environment grounded well in the principles of Mechatronics engineering, promoting creativity, and nurturing teamwork.

NBA PROGRAM EDUCATIONAL OUTCOMES OF THE MECHATRONICS ENGINEERING DEPARTMENT (PEOs)

The graduates**:**

PEO1: Are expected to apply analytical skills and modelling methodologies to recognize, analyze, synthesize and implement operational solutions to engineering problems, product design and development, and manufacturing.

PEO2: Will be able to work in national and international companies as engineers who can contribute to research and development and solve technical problems by taking an initiative to develop and execute projects and collaborate with others in a team.

PEO3: Shall be capable of pursuing higher education in globally reputed universities by conducting original research in related disciplines or interdisciplinary topics, ultimately contributing to the scientific community with novel research findings.

PEO4: Are envisioned to become technology leaders by starting high – tech companies based on social demands and national needs.

PEO5: Shall develop flexibility to unlearn and relearn by being in pursuit of research and development, evolving technologies and changing societal needs thus keeping themselves professionally relevant.

NBA PROGRAM OUTCOMES (PO):

The POs are exemplars of the attributes expected of a graduate of an accredited programme:

PO1- Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2- Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3- Design solutions for complex engineering problems and design system components or processes that meet t h e specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4- Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5- Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

PO6- Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal, and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7- Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8- Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9- Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10- Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11- Demonstrate knowledge and understanding of t h e engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12- Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

NBA PROGRAM SPECIFIC OUTCOMES OF THE MECHATRONICS ENGINEERING DEPARTMENT (PSO'S)

At the end of the course the student will be able to:

PSO 1 Able to apply the knowledge of sensors, drives, actuators, controls, robotics and modern software tool to integrate a system to perform specified tasks.

PSO 2. Able to design, model, analyze, and testing of intelligent products, systems and controllers using appropriate technology and software tools.

PSO 3. Able to interface devices and elements to a central system having the capability of real time data sharing, storage, retrieval, analysis, decision making with global connectivity features for visibility and intervention.

IET LEARNING OUTCOMES (LO):

C1. Apply knowledge of mathematics, statistics, natural science and engineering principles to the solution of complex problems. Some of the knowledge will be at the forefront of the particular subject of study

C2. Analyse complex problems to reach substantiated conclusions using first principles of mathematics, statistics, natural science and engineering principles

C3. Select and apply appropriate computational and analytical techniques to model complex problems, recognising the limitations of the techniques employed

C4. Select and evaluate technical literature and other sources of information to address complex problems

C5. Design solutions for complex problems that meet a combination of societal, user, business and customer needs as appropriate. This will involve consideration of applicable health & safety, diversity, inclusion, cultural, societal, environmental and commercial matters, codes of practice and industry standards

C6. Apply an integrated or systems approach to the solution of complex problems

C7. Evaluate the environmental and societal impact of solutions to complex problems and minimise adverse impacts

C8. Identify and analyse ethical concerns and make reasoned ethical choices informed by professional codes of conduct

C9. Use a risk management process to identify, evaluate and mitigate risks (the effects of uncertainty) associated with a particular project or activity

C10. Adopt a holistic and proportionate approach to the mitigation of security risks

C11. Adopt an inclusive approach to engineering practice and recognise the responsibilities, benefits and importance of supporting equality, diversity and inclusion

C12. Use practical laboratory and workshop skills to investigate complex problems

C13. Select and apply appropriate materials, equipment, engineering technologies and processes, recognising their limitations

C14. Discuss the role of quality management systems and continuous improvement in the context of complex problems

C15. Apply knowledge of engineering management principles, commercial context, project and change management, and relevant legal matters including intellectual property rights

C16. Function effectively as an individual, and as a member or leader of a team

C17. Communicate effectively on complex engineering matters with technical and non-technical audiences

C18. Plan and record self-learning and development as the foundation for lifelong learning/CPD

COURSE LEARNING OUTCOMES

CO1: Understand the real-world scenarios where robotics and autonomous systems might be applied using ROS package.

CO2: Develop different robotic models using simulation tools and apply kinematic and dynamic models of robots for moving it.

CO3: Use sensing and actuation systems applied to robotic systems, and the importance of using multiple sensors in robotic and autonomous systems in simulation and real mobile robots/industrial robots for automation.

CO4: Develop SLAM approaches for mobile robot navigation, as well as control of multi-axis manipulators.

CO5: Develop an application involving functional safety, health, ethical, management, legal, society and environment as a team member or leader.

CO6: Communicate the results to peers through report writing and oral presentation.

CO7: Understand the use of various industry standards, refer literatures, blogs, books while implementing the mini project.

LIST OF EXPERIMENTS:

Sl. No.	TITLE OF EXPERIMENT	DATE	PAGE NO.	FACULTY SIGN	MARKS
1.	Introduction to ROS				
2.	Turtlesim programming in ROS				
3.	Robot vision- part shape and color detection using Sherlock – image processing software				
4.	Image Processing using OpenCV and ROS				
5.	Depth Camera Image Processing using OpenCV				
6.	Robot Motion in ROS Gazebo/RViz simulation environment.				
7.	Introduction to Mobile Robot simulation using URDF and SLAM				
8.	Introduction to Mobile Robots (Turtlebot)				
9.	Introduction to COBOT, implementation of industrial processes with COBOT				
10.	Mini-Project: a) Model a mobile robot using 3D printer b) Model the Gazebo environment using URDF/XACRO programming c) Develop a solution for robotic vision application using OpenCV				

EXPERIMENTS MAPPING

Sl. No.	TITLE OF EXPERIMENT	CO	PO	LO	BL
1.	Introduction to ROS				
2.	Turtlesim programming in ROS				
3.	Robot vision- part shape and color detection using Sherlock – image processing software				
4.	Image Processing using OpenCV and ROS				
5.	Depth Camera Image Processing using OpenCV				
6.	Robot Motion in ROS Gazebo/RViz simulation environment.				
7.	Introduction to Mobile Robots simulation using URDF and SLAM				
8.	Introduction to Mobile Robots				
9.	Introduction to COBOT, implementation of industrial processes with COBOT				
10.	Mini-Project: a) Model a mobile robot using 3D printer b) Model the Gazebo environment using URDF/XACRO programming c) Develop a solution for robotic vision application using OpenCV				

Introduction to Linux Commands

File Commands

ls – directory listing

ls -al – formatted listing with hidden files

cd dir - change directory to dir

cd – change to home

pwd – show current directory

mkdir dir – create a directory dir

rm file – delete file

rm -r dir – delete directory dir

rm -f file – force remove file

rm -rf dir – force remove directory dir

cp file1 file2 – copy file1 to file2

cp -r dir1 dir2 – copy dir1 to dir2; create dir2 if it doesn't exist

mv file1 file2 – rename or move file1 to file2 if fle2 is an existing directory, moves file1 into directory file2

ln -s file link – create symbolic link link to file

touch file – create or update file

cat file – places standard input into file

more file – output the contents of file

head file – output the first 10 lines of file

tail file – output the last 10 lines of file

tail -f file – output the contents of file as it grows, starting with the last 10 lines

Process Management

ps – display your currently active processes

top – display all running processes

kill pid – kill process id pid

killall proc – kill all processes named proc * bg – lists stopped or background jobs; resume a stopped job in the background

fg – brings the most recent job to foreground

fg n – brings job n to the foreground

File Permissions

chmod octal file – change the permissions of file to octal, which can be found separately for user, group, and world by adding:

4 – read (r)

2 – write (w)

1 – execute (x)

Examples:

chmod 777 – read, write, execute for all

chmod 755 – rwx for owner, rx for group and world

chmod +x read.py give permission to file read.py for execution.

SSH

ssh user@host – connect to host as user

Network

ping host – ping host and output results

whois domain – get whois information for domain dig domain – get DNS information for

domain dig -x host – reverse lookup host

wget file – download file

wget -c file – continue a stopped download

Installation

Ctrl+C – halts the current command

Ctrl+Z – stops the current command, resume with **fg** in the foreground or **bg** in the background

Ctrl+D – log out of current session, similar to exit

Ctrl+W – erases one word in the current line

Ctrl+U – erases the whole line

Ctrl+R – type to bring up a recent command

!! – repeats the last command

exit – log out of current session

w – display who is online

whoami –who you are logged in as

finger user – display information about user

uname -a – show kernel information

cat /proc/cpuinfo – CPU information

cat /proc/meminfo – memory information man command – show the manual for command df – show disk usage

du – show directory space usage

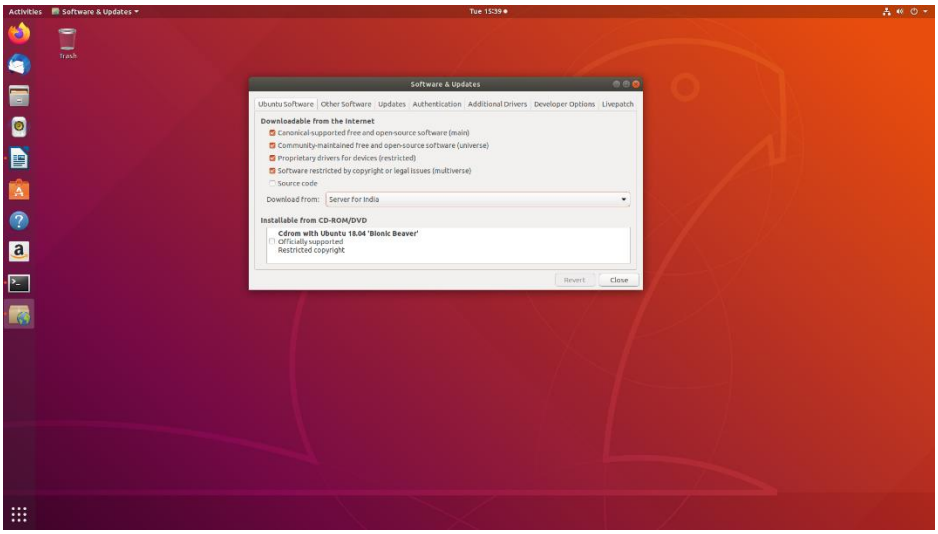
free – show memory and swap usage

whereis app – show possible locations of app

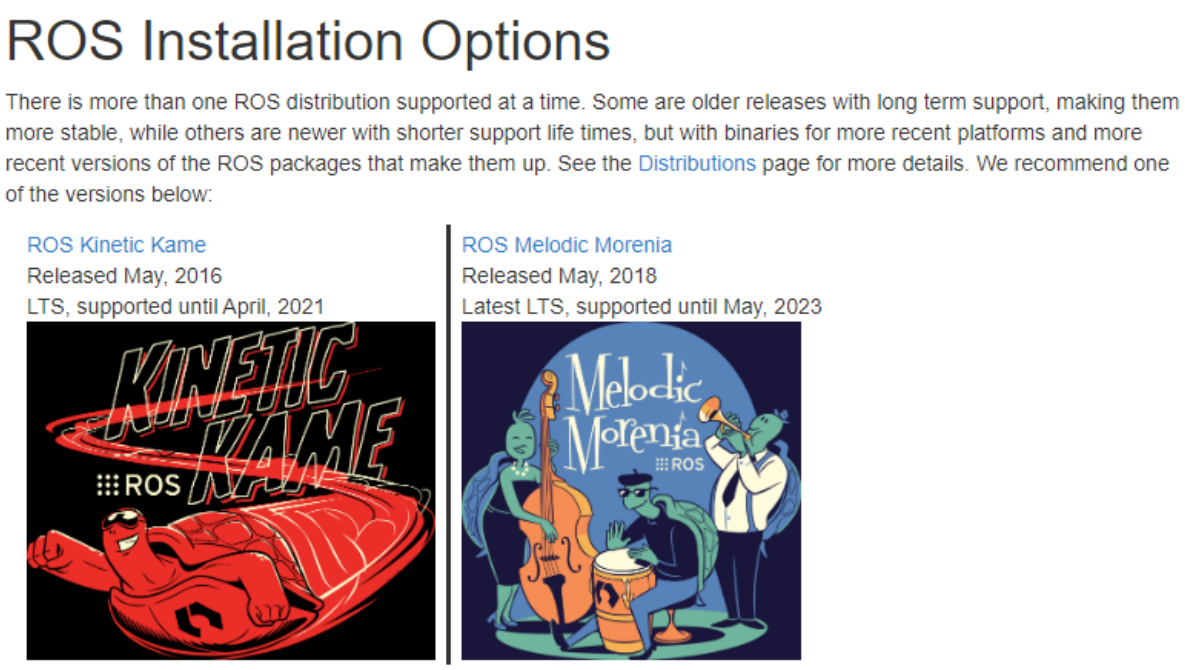
which app – show which app will be run by default

Installing ROS in ubuntu 16.04

✓ Select Softwares and Updates and check as shown in Figure:



1. Choose ROS distribution from http://wiki.ros.org/ROS/Installation
    - We choose ROS Kinetic in the current lab.



2. Select the platform
3. Follow the steps provided in http://wiki.ros.org/kinetic/Installation

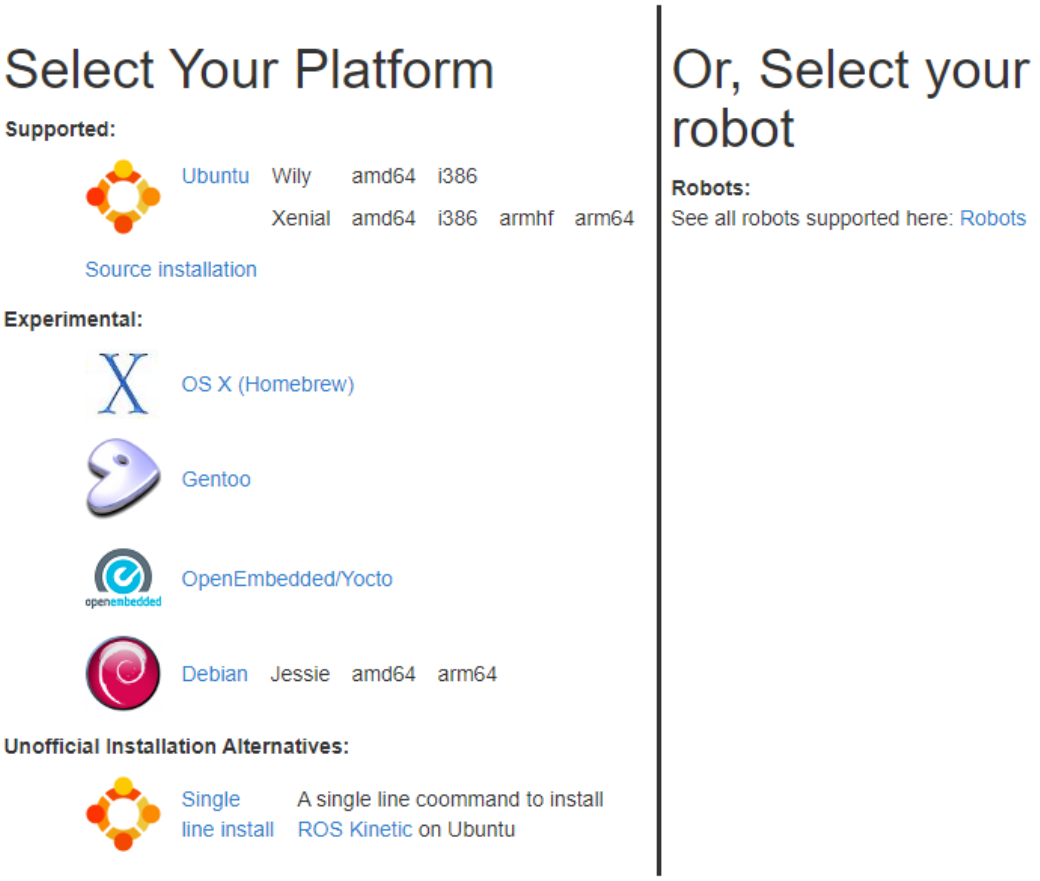


Figure : ROS supporting operating System

4. Setup sources.list

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

5. Set up keys

```
sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
```

6. Update the Debian package

```
sudo apt-get update
```

7. Install Kinetic Desktop Full version

```
sudo apt-get install ros-kinetic-desktop-full
```

8. Environmental setup

```
echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc

source ~/.bashrc
```

9. Install dependencies for building packages

```
sudo apt install python-rosdep python-rosinstall python-rosinstall-generator python-wstool build-essential
```

10. Initialize rosdep

```
sudo apt install python-rosdep

sudo rosdep init

rosdep update
```

11. Create ROS workspace

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

$ cd ~/catkin_ws/

$ catkin_make
```

1 - INTRODUCTION TO ROS

Overview of ROS:

ROS is Robot Operating System. Originally developed in 2007 at the Stanford Artificial Intelligence Laboratory. Today it is used by many robots, universities and companies and has become De facto standard for robot programming.

Some characteristic features of ROS:

- Peer to peer: Individual programs communicate over defined Application Programming Interface (API) such as ROS messages, services, etc.
- Distributed: Programs can be run on multiple computers and communicate over the network.
- Multi-lingual: ROS modules can be written in any language for which a client library exists (C++, Python, MATLAB, Java, etc.).
- Light-weight: Stand-alone libraries are wrapped around with a thin ROS layer.
- Free and open-source: Most ROS software is open-source and free to use.

Terminologies:

ROS Master:

It manages the communication between nodes (processes) by enabling individual ROS nodes to locate one another. Every node in the ROS system needs to register with the ROS Master at startup. The following command has to be used to start a ROS master.

$roscore

ROS Node:

A node is a process that performs computation. It is a single-purpose, executable program that is individually compiled, executed and managed. These nodes are organized in packages. Following are the commonly used commands related to nodes.

i) Run a node:

$rosrun package_name node_name

ii) See active nodes:

$rosnode list

iii) See information about a node:

$rosnode info node_name

ROS Topics:

Topics are named buses over which nodes exchange messages. Nodes can publish or subscribe to a topic. Typically, 1 publisher and n subscribers exist. Some useful commands related to topics are:

i) See active topics:

$rostopic list

ii) Subscribe and print the contents of a topic:

$rostopic echo /topic

iii) See information about a topic:

$rostopic info /topic

ROS Messages:

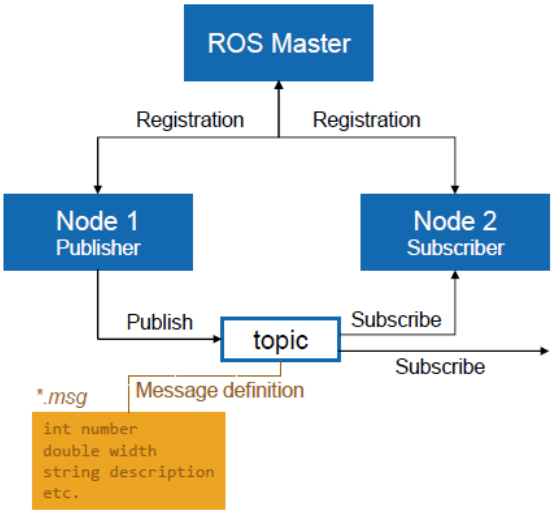
A message is a simple data structure, comprising typed fields. Nodes communicate with each other by publishing/subscribing messages to/from topics. Sending the message is termed Publishing (sender node is called Publisher). Receiving messages is termed subscribing (receiving node is called subscriber). Some useful commands are:

i) See the type of topic:

$rostopic type /topic

ii) Publish a message to topic:

$rostopic pub /topic type data



EXERCISES:

1) Run the Hello World demo program in ROS.

- ✓ Start ROS Master.

  $roscore

- ✓ Open a new terminal and run the talker node. (Note down your observation)

  $rosrun roscpp_tutorials talker.cpp

- ✓ Open another terminal window and start the listener node. (Note down your observation)

  $rosrun roscpp_tutorials listener.cpp

- ✓ Shutdown the above two terminals by pressing CTRL+C through keyboard.
- ✓ Run two nodes together using roslaunch command. (Note down your observation)

  $roslaunch roscpp_tutorials talker_listener.launch

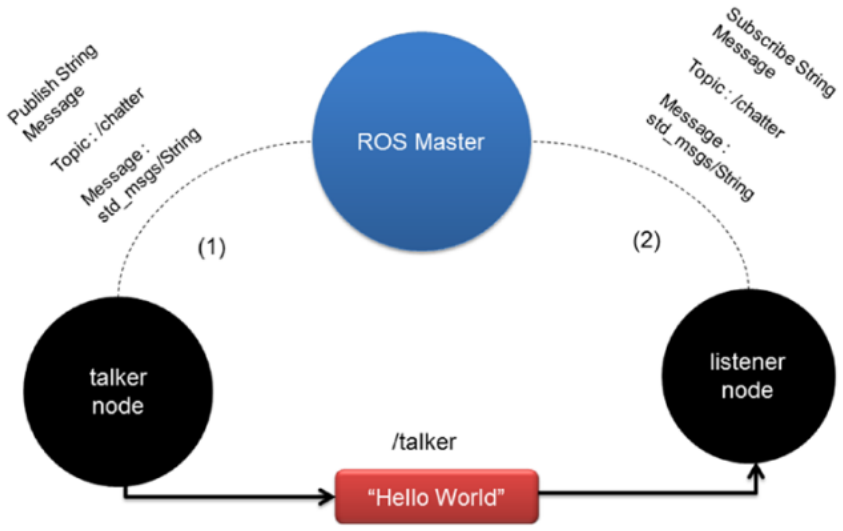


Figure 2: Communication between Talker and Listener Nodes

There are two nodes: talker and listener. The talker nodes publishes the string message and listener node subscribes it. In the Hello world example, talker publishes a Hello World message and the listener subscribes it and prints it in the monitor. Both the nodes communicate with the ROS master.

2) Create and Run a ROS program (python program)

Step 1: Create ROS workspace folder, where ROS packages are kept.

- ✓ Open a new terminal.

  $mkdir -p ~/catkin_ws/src/

This creates the folder called catkin_ws, and src folder in it. ROS workspace is also called catkin workspace.

- ✓ Switch to src folder

  $cd catkin_ws/src

- ✓ Initialize a new ROS workspace.

  $catkin_init_workspace

- ✓ Check the contents in src folder

  $ls

- ✓ Build the workspace; switch to catkin_ws folder.

  $cd catkin_ws

  $catkin_make

  Note: src folder stores our packages. To build a package, it is necessary to copy the packages to the src folder.

- ✓ Add the workspace environment; open .bashrc in home folder

  $cd .. (Remember the space between the cd and ..)

  $gedit .bashrc

- ✓ Add the line at the end

  source ~/catkin_ws/devel/setup.bash

  Now the path is added in the current terminal.

  Note: Use roscd to check the folder

- ✓ Setup catkin workspace in the catkin_ws folder

  $catkin_make install

  It creates the install folder in the workspace.

Step 2: Create the ROS package

- ✓ ROS package is where the ROS nodes are organized such as libraries.

  $ catkin_create_pkg <ros_package_name$ <package_dependencies$

  Example:

  $ **cd catkin_ws/src**

  $ **catkin_create_pkg <Your_Initials$ roscpp rospy std_msgs**

Step 3: Creating python nodes

- ✓ Make a folder scripts inside the package **<Your_Initials$**/src

  $ cd <Your_Initials$/src

  $ mkdir scripts

  $ cd scripts

- ✓ Use any editor to create python file.

  Using terminal window:

  $gedit talker.py

  Using Visual Studio Code:

  Drag and drop the **<Your_Initials$** folder created.

  Right click on the src and create new file.

```
#!/usr/bin/env python
# license removed for brevity
import rospy
from std_msgs.msg import String

def talker():
    pub = rospy.Publisher('chatter', String, queue_size=10)
    rospy.init_node('talker', anonymous=True)
    rate = rospy.Rate(10) # 10hz
    while not rospy.is_shutdown():        hello_str = "hello world %s" % rospy.get_time()
        rospy.loginfo(hello_str)
        pub.publish(hello_str)


        rate.sleep()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        pass
```

Type the following and save

- ✓ In the terminal type

  $ chmod +x talker.py

- ✓ Use any editor to create python file listener.py.

  $ gedit listener.py

Type the following and save

```
 #!/usr/bin/env python
import rospy
from std_msgs.msg import String

def callback(data):
    rospy.loginfo(rospy.get_caller_id() + "I heard %s", data.data)

def listener():
# In ROS, nodes are uniquely named. If two nodes with the same
    # name are launched, the previous one is kicked off. The
    # anonymous=True flag means that rospy will choose a unique
    # name for our 'listener' node so that multiple listeners can
    # run simultaneously.
    rospy.init_node('listener', anonymous=True)

    rospy.Subscriber("chatter", String, callback)

    # spin() simply keeps python from exiting until this node is stopped
    rospy.spin()

if __name__ == '__main__':
    listener()
```

- ✓ In the terminal type

  $chmod +x listener.py
- ✓ **$cd ~/catkin_ws**
- ✓ **$catkin_make**
- ✓ start the roscore in terminal 1

  $ roscore
- ✓ Start the talker.py in the terminal 2

  $ rosrun <Your_Initials$ talker.py
- ✓ Start the listener.py in the terminal 3

  $ rosrun <Your_Initials$ listener.py

Step 4: Create Launch file

- ✓ In <**Your_Initials$**/src folder create launch folder
- ✓ gedit
- ✓ Copy the following

```
<launch$
<node name="listener_node" pkg="<Your_Initials$" type="listener.py"
output="screen"/$
<node name="talker_node" pkg="<Your_Initials$" type="talker.py"
output="screen"/$
</launch$
```

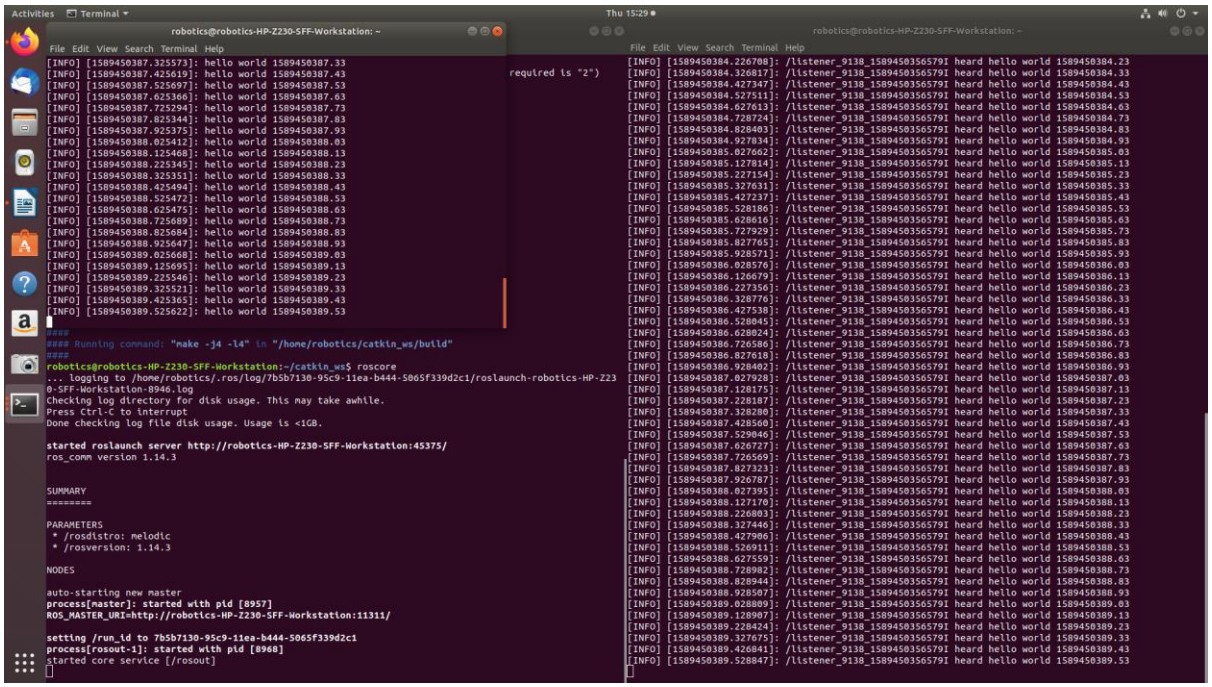
- ✓ Save as **talker_listener.launch**

- ✓ Change the permission of the file as

  $sudo chmod +x talker_listener.launch
- ✓ Execute the launch file

  $roslaunch hello_world talker_listener.launch
- ✓ Visualize computing graph as (Type in new terminal)

  $rqt_graph



3) Publish and subscribe custom data

✓ **Type in terminal window for multi terminal windows**

sudo apt-get install terminator

✓ **Create Package in catkin_ws/src folder**

$cd catkin_ws/src

$catkin_create_pkg test_pub_sub rospy roscpp std_msgs

$cd test_pub_sub/src

$mkdir scripts

$cd scripts

✓ **To view the message**

$roscd std_msgs

$ls

$cd msg

$ls

$ gedit String.msg

✓ **Type the following codes in any editor in the 'scripts' folder that you created**

(Be careful of indentations and quotes)

Type the code for publisher: pub.py

$ gedit pub.py:

```
#!/usr/bin/env python

import rospy
from test_pub_sub.msg import test_custom_msg

def Publisher():

        pub=rospy.Publisher('string_publish', test_custom_msg, queue_size=10)
        rate=rospy.Rate(1)
        msg_to_publish = test_custom_msg()
        counter=0
        while not rospy.is_shutdown():
                string_to_publish="Publishing %d"%counter
                counter+=1
                msg_to_publish.data=string_to_publish
                msg_to_publish.counter=counter
                pub.publish(msg_to_publish)
                rospy.loginfo(msg_to_publish)
                rate.sleep()

if __name__=="__main__":
        rospy.init_node("simple publisher")
        Publisher()
```

✓ **Type the code for subscriber: sub.py**

$gedit sub.py

```
#!/usr/bin/env python

import  rospy
from test_pub_sub.msg import test_custom_msg

def Subscriber():
        sub=rospy.Subscriber('string_publish', test_custom_msg, callback_function)
        rospy.spin()

def callback_function(message):
        string_rec=message.data
        counter_rec=message.counter
        rospy.loginfo("I received: %d"%counter_rec)

if __name__=="__main__":

        rospy.init_node("simple subscriber")
        Subscriber()
```

- ✓ **Give the permission for compilation:**

$chmod +x pub.py

$chmod +x sub.py

- ✓ **Compile with catkin_make**

$roscd

$cd ..

$catkin_make

$source devel/setup.bash

- ✓ **Go to pub.py folder**

$roscd test_pub_sub/src/scripts

$roscore

✓ **In a separate terminal:**

$rosrun test_pub_sub pub.py

✓ **See the list of topics running**

$rostopic list

✓ **See what is being published**

$rostopic echo /string_publish

✓ **Run your node:**

$rosrun test_pub_sub sub.py

$rosrun rqt_graph rqt_graph

4) Modify the code to publish custom data:

✓ **Go to package: cd catkin_ws/src/test_pub_sub**

$mkdir msg

$cd msg

✓ **Create your custom message:**

$gedit test_custom_msg.msg

Type the following and save

string data

int32 counter

✓ **Modify cmakelist**

$ cd ..

$gedit CmakeLists.txt

✓ **Add blue lines at the line at the end:**

find_package (catkin REQUIRED COMPONENTS

roscpp

rospy

std_msgs

message_generation

)

✓ **Uncomment lines in the following:**

add_message_files(

FILES

test_custom_msg.msg

)

✓ **Uncomment generate_messages**

generate_messages(

DEPENDENCIES

std_msgs

Or other packages containing msgs

)

catkin_package(

CATKIN depends roscpp rospy std_msgs message_runtime

)

✓ Add the blue lines in Package.xml

$gedit package.xml

<build_depend$message_generation</build_depend$

<exec_depend$message_runtime</exec_depend$ <buildtool_depend$catkin</buildtool_depend$

<build_depend$roscpp</build_depend$

<build_depend$rospy</build_depend$

<build_depend$std_msgs</build_depend$

<build_depend$message_generation</build_depend$

<build_export_depend$roscpp</build_export_depend$

<build_export_depend$rospy</build_export_depend$

<build_export_depend$std_msgs</build_export_depend$

<exec_depend$roscpp</exec_depend$

<exec_depend$rospy</exec_depend$

<exec_depend$std_msgs</exec_depend$

<exec_depend$message_runtime</exec_depend$

✓ **Invoke catkin_make**

$cd ~/catkin_ws/

$catkin_make

✓ **Run the publisher and subscriber nodes**

$roscore

$rosrun test_pub_sub pub.py

$rostopic list

$rostopic echo /string_publish

$rosrun test_pub_sub sub.py

$rosrun rqt_graph rqt_graph

POST LAB EXERCISES:

1) Modify the python code in Exercise 1 to display your Name.

Steps to follow to Write Publisher and Subscriber for ROS Topics:

Publisher

• **Step 1.** Determine a **name** for the topic to publish

• **Step 2.** Determine the **type** of the messages that the topic will publish

• **Step 3.** Determine the **frequency** of topic publication (how many messages per second)

• **Step 4.** Create a publisher object with parameters chosen

• **Step 5.** Keep publishing the topic message at the selected frequency

Subscriber

• **Step 1.** Identify the **name** for the topic to listen to

• **Step 2.** Identify the **type** of the messages to be received

• **Step 3.** Define a callback function that will be automatically executed when a new message is received on the topic.

• **Step 4.** Start listening for the topic messages

• **Step 5.** Spin to listen for ever (in C++)

2) Write a python code in your 'scripts' folder to draw the letter 'D' using turtlesim.

3) Write a python code in your 'scripts' folder to draw a hexagon using turtlesim.

2- TURTLESIM PROGRAMMING

EXERCISES:

1) Run the turtlesim program and operate using keyboard keys.

- ✓ Open a new terminal window and start roscore and minimize this window.

  $ roscore

  Note: Open another terminal window and type $ rosrun turtlesim and press tab twice to see the list of different nodes present in turtlesim.

- ✓ Open another terminal window (window1), start Turtlesim application

  $ rosrun turtlesim turtlesim_node (You can see a new window with turtle appearing on the screen)

- ✓ Open another terminal window (window 2) and list the topics published by the turtlesim node.

  $ rostopic list

- ✓ List the services created by the turtlesim node.

  $ **rosservice list**

- ✓ Open new terminal (window 3) and type to move the turtle using keyboard

  $ **rosrun turtlesim turtle_teleop_key**

  Note: To end the turtlesim press CTRL+C.

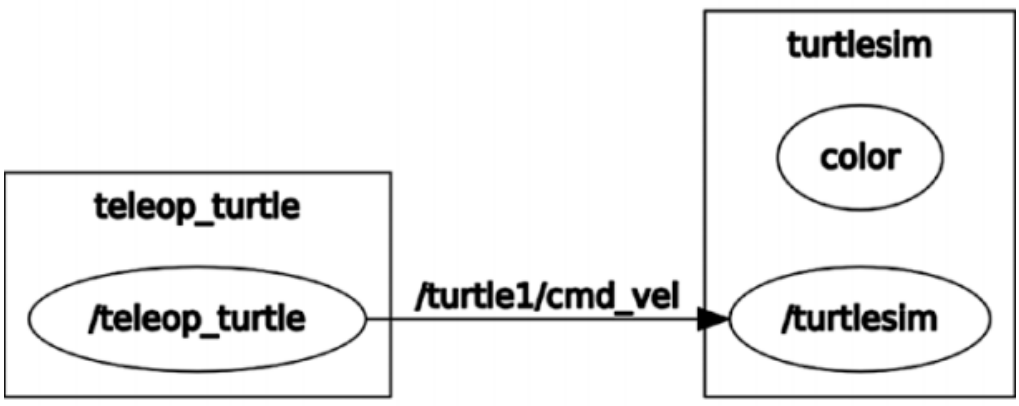
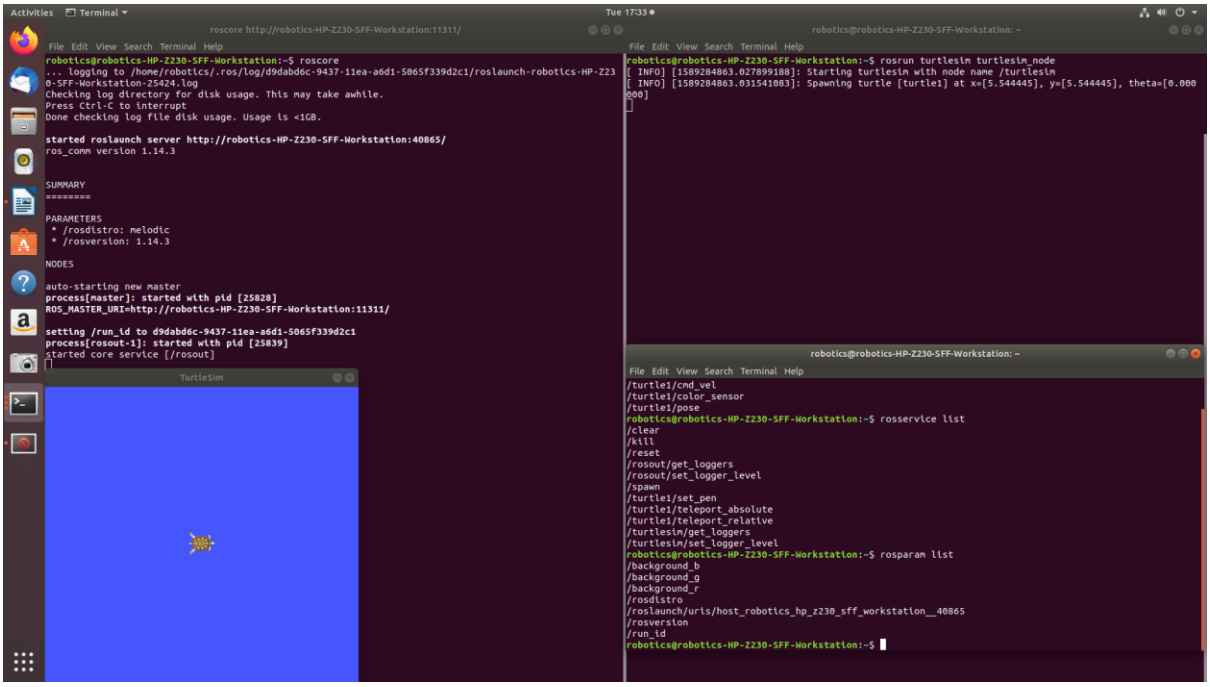
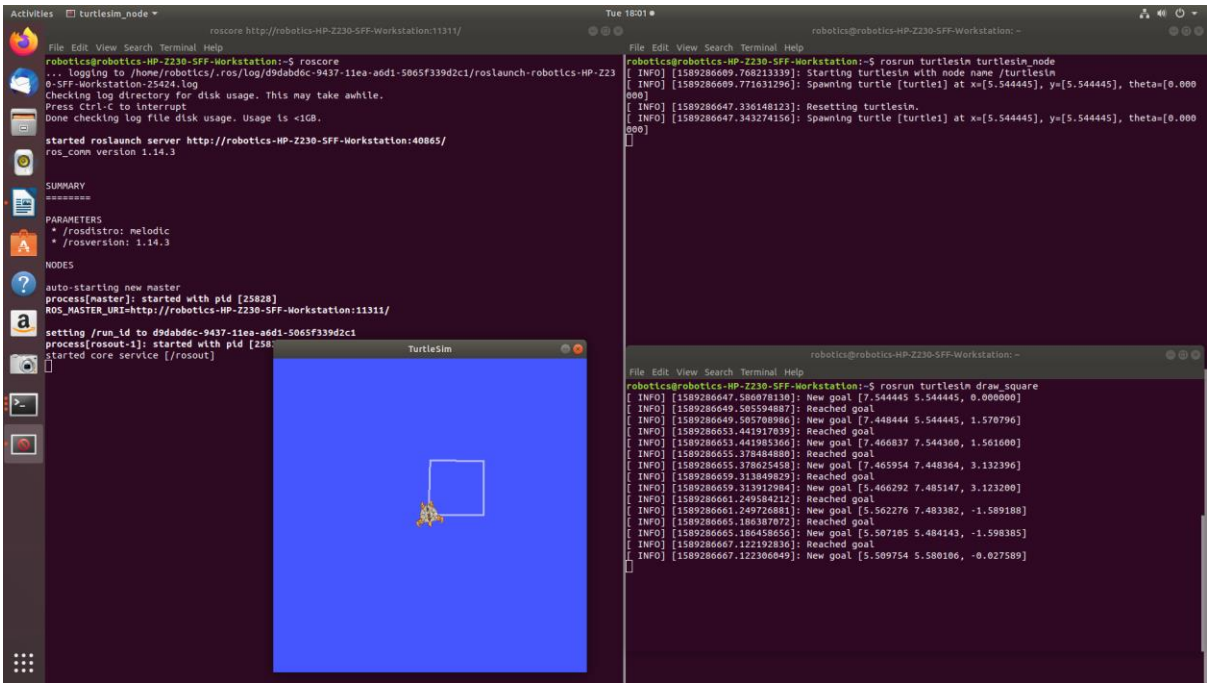


Figure: turtlesim and teleop node backend





2) Find the following:

a) What is the topic used to know the position?

$ rostopic list

b) What is the topic that makes the robot move?

c) What is the type of the topic for position?

$ rostopic info <topic_name$

d) What is the content of the message type found in (c)?

$ rosmsg show <type_of_message$

3) Control the turtle motion by modifying the linear velocity and angular velocity using command line.

- ✓ Start roscore in the terminal 1

  $ roscore

- ✓ Start the turtlesim node in terminal 2

  $ rosrun turtlesim turtlesim_node

- ✓ List of topics published

  $ rostopic list

- ✓ To move the turtle inside the turtlesim application, publish the linear and angular velocity to the /turtle1/cmd_vel topic. Check the type of the /turtle1/cmd_vel topic by using the following command.

  $ rostopic type /turtle1/cmd_vel

  $rosmsg show geometry_msgs/Twist

- ✓ Publish the message to geometry_msgs/Twist to move the robot

  It has 2 components: linear velocity (moves the robot forward and backward) and angular velocity (rotates the robots on its axis).

- ✓ To move the turtle using command line use tab to complete the command

  $ rostopic pub /turtle1/cmd_vel geometry_msgs/Twist “linear:

  x:0.1

  y:0

  z:0

  angular:

  x:0

y:0

z:0”

4) Write a python code to control the turtle motion by modifying the linear velocity and angular velocity (linear velocity and angular velocity given through command line).

- ✓ **Create Package in catkin_ws/src folder**

  $cd catkin_ws/src

  $catkin_create_pkg move_robot rospy roscpp std_msgs

- ✓ **In scripts folder create a file move_turtle.py**

  $cd move_robot

  $cd src

  $mkdir scripts

  $cd scripts

  $gedit move_turtle.py

Write the following code:

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

#/turtle1/Pose topic callback
def pose_callback(pose):
   rospy.loginfo("Robot X = %f : Y=%f :Z=%f\n",pose.x,pose.y,pose.theta)

#Function to move turtle: Linear and angular velocities are arguments
def move_turtle(lin_vel,ang_vel):
   rospy.init_node('move_turtle', anonymous=False)
   #The /turtle1/cmd_vel is the topic in which we have to send Twist messages
   pub = rospy.Publisher('/turtle1/cmd_vel', Twist, queue_size=10)
```

```
    rate = rospy.Rate(10) # 10hz
    #Creating Twist message instance
    vel = Twist()
    while not rospy.is_shutdown():

        #Adding linear and angular velocity to the message
        vel.linear.x = lin_vel
        vel.linear.y = 0
        vel.linear.z = 0
      vel.angular.x = 0
      vel.angular.y = 0
      vel.angular.z = ang_vel
      rospy.loginfo("Linear Vel = %f: Angular Vel =%f",lin_vel,ang_vel)
      #Publishing Twist message
      pub.publish(vel)
      rate.sleep()

if __name__ == "__main__":
   try:
     #Providing linear and angular velocity through command line
      move_turtle(float(sys.argv[1]),float(sys.argv[2]))
   except rospy.ROSInterruptException:
      pass
```

- ✓ **In the terminal type**

  $chmod +x move_turtle.py

  $cd ~catkin_ws/

  $catkin_make

- ✓ Start roscore

  $ roscore

- ✓ Start the turtlesim node

$ rosrun turtlesim turtlesim_node

- ✓ Print the robot position in the terminal 3

  $ rostopic echo /turtle1/pose

- ✓ Move turtle with command line arguments as linear velocity=0.1 m/s and angular velocity as 0.1 rad/s in terminal 4

  $ rosrun hello_world move_turtle.py 0.1 0.1

- ✓ Open terminal 5 to visualize the nodes **$ rqt_graph**

5) Go to Goal: Write a Python code to move the turtle from one position to another

```
#!/usr/bin/env python
import rospy
from geometry_msgs.msg import Twist
from turtlesim.msg import Pose
import math
import time
from std_srvs.srv import Empty

X=0.0
Y=0.0
yaw=0.0
def pose_callback(pose):
        global X, Y, yaw
        rospy.loginfo("X=%f, Y=%f\n", pose.x, pose.y)
        X=pose.x
        Y=pose.y
        yaw=pose.theta

def move(speed, distance, is_forward):
        velocity_message=Twist()
        global X,Y
        X0=X
```

```
        Y0=Y
        if(is_forward):
                velocity_message.linear.x=abs(speed)
        else:
                velocity_message.linear.x=-abs(speed)

        distance_moved=0.0
        loop_rate=rospy.Rate(10)
        cmd_vel_topic='/turtle1/cmd_vel'
        velocity_publisher=rospy.Publisher(cmd_vel_topic, Twist, queue_size=10)
        while(True):
                rospy.loginfo("Turtlesim moves forward")
                velocity_publisher.publish(velocity_message)
                loop_rate.sleep()
                #rospy.loginfo("%f %f %f %f", X,Y,X0,Y0)

                distance_moved =  abs(0.5*math.sqrt(((X-X0)**2)+((X-Y0)**2)))
                print(distance_moved)
                if not(distance_moved<distance):
                        rospy.loginfo("reached")
                        rospy.logwarn("Stopping the Robot")
                        break

        velocity_message.linear.x=0
        velocity_publisher.publish(velocity_message)
def rotate(angular_speed_degree, relative_angle_degree,clockwise):
        global yaw
        velocity_message=Twist()
        velocity_message.linear.x=0
        velocity_message.linear.y=0
        velocity_message.linear.z=0
```

```
        velocity_message.angular.x=0
        velocity_message.angular.y=0
        velocity_message.angular.z=0
        theta0=yaw
        angular_speed=math.radians(abs(angular_speed_degree))
        if(clockwise):
                velocity_message.angular.z=-abs(angular_speed)
        else:
                velocity_message.angular.z=abs(angular_speed)
        angle_moved=0.0
        loop_rate=rospy.Rate(10)
        and_vel_topic='/turtle1/cmd_vel'
        velocity_publisher=rospy.Publisher(cmd_vel_topic,Twist,queue_size=10)
        t0=rospy.Time.now().to_sec()
        while(True):
                rospy.loginfo("Turtlesim rotates")
                velocity_publisher.publish(velocity_message)
                t1=rospy.Time.now().to_sec()
                current_angle_degree=(t1-t0)*angular_speed_degree
                loop_rate.sleep()
                if (current_angle_degree>relative_angle_degree):
                        rospy.loginfo("reached")
                        break
        velocity_message.angular.z=0
        velocity_publisher.publish(velocity_message)
def go_to_goal(x_goal,y_goal):
        global X
        global Y, yaw
        velocity_message = Twist()
        cmd_vel_topic ='/turtle1/cmd_vel'
        while(True):
```

```
                K_linear=0.5
                distance=abs(math.sqrt(((x_goal-X)**2)+((y_goal-Y)**2)))
                linear_speed=distance*K_linear
                K_angular=4.0
                desired_angle_goal=math.atan2(y_goal-Y,x_goal-X)
                angular_speed=(desired_angle_goal-yaw)*K_angular
                velocity_message.linear.x=linear_speed
                velocity_message.angular.z=angular_speed
                velocity_publisher.publish(velocity_message)
                print('x=',X,'y=',Y)
                if (distance<0.01):
                        break

def setDesiredOrientation(desired_angle_radians):
        relative_angle_radians=desired_angle_radians-yaw
        if(relative_angle_radians<0):
                clockwise=1
        else:
                clockwise=0
        print(relative_angle_radians)
        print(desired_angle_radians)
        rotate(30,math.degrees(abs(relative_angle_radians)),clockwise)




if __name__ == '__main__':
        try:
                rospy.init_node('turtlesim_motion_pose',anonymous=True)
                cmd_vel_topic='/turtle1/cmd_vel'
```

```
            #cmd_vel_topic='/cmd_vel_mux/input/teleop'
            velocity_publisher = rospy.Publisher(cmd_vel_topic, Twist, queue_size=10)
            position_topic='/turtle1/pose'
            rospy.Subscriber(position_topic, Pose, pose_callback)
            time.sleep(1)
            move(1.0,3.0,False)
            #rotate(90,90,True)
            #go_to_goal(5.0,9.0)
            #setDesiredOrientation(math.radians(90))

    except rospy.ROSInterruptException:
            rospy.loginfo("node terminated")
```

- ✓ **In the terminal type**

  $chmod +x move_turtle_dir.py
- ✓ **Start roscore**

  $roscore
- ✓ **Start the turtlesim node**

  $rosrun turtlesim turtlesim_node

- ✓ **Print the robot position in the terminal 3**

  $rostopic echo /turtle1/pose
- ✓ Move turtle with command line arguments as linear velocity=0.1 m/s and angular velocity as 0.1 rad/s in terminal 4

  $rosrun move_robot move_turtle_dir.py

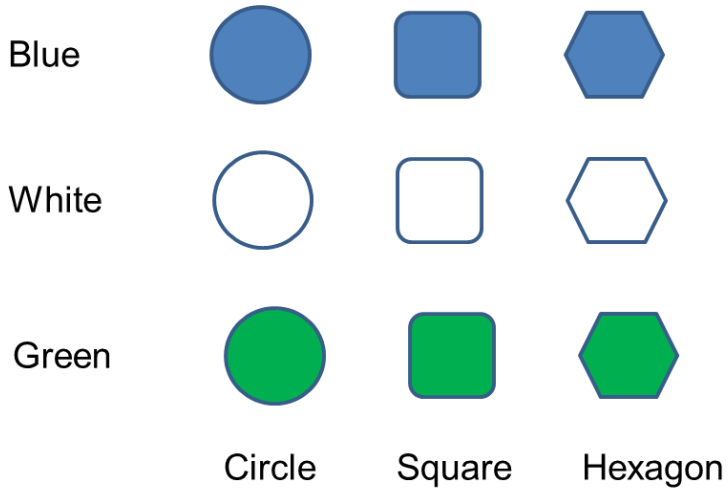
POST LAB EXERCISES:

1. Write a python code in your 'scripts' folder to draw the letter 'D' using turtlesim.
2. Write a python code in your 'scripts' folder to draw a hexagon using turtlesim.

3- ROBOT VISION- PART SHAPE AND COLOR DETECTION USING SHERLOCK –IMAGE PROCESSING SOFTWARE

Aim: Identification and classification of the objects based on shape using suitable algorithms.

Images of objects to be classified based on shape



Procedure

1. Click the Load Image button in the Image window tool bar of Sherlock and load the image file (square).
2. Create one more image window and load monchrome of the image which is loaded before.
3. Create a ROI, click on the rectangle button, in the Image Window toolbar and Draw a box around the object by clicking the left mouse button.
4. To display the ROI options dialog: double-click inside the ROI, or double-click on the ROI name in the Program window, or right-click inside the ROI and select “edit”.
5. Select ‘verify pattern’ operator from the algorithm list.
6. Select ‘parameters’
7. Select “Train’’ from the execution mode and observe the trained pattern. If pattern is not selected properly, change ROI accordingly.

8. Select “Run’’ from the execution mode and observe the trained pattern. If pattern is not selected properly, change ROI accordingly.

9. Click the “Apply” button and Click the “OK” button

10. Click the “Close” button

11. Click on the “Variables” tab at the bottom of the Instructions window.

12. Create three number variables, by clicking the “N” button thrice, in the Variables tool bar.

13. In the Program window, right-click on “score” and select “connect variable,” “varA”.

14. Double click on “varA” and rename it to “square_score”.

15. Run and Save the Basic Investigation. Press F9, or in the Menu bar select Run once. Sherlock calculates the average color inside the rectangle. The values are displayed in the Variables window. You can observe the value of square score will be 100.

16. Load images of hexagonal, circular shapes and repeat the investigation from step1. Observe Hexagon_score and circle_score which are displayed in the variable window.

17. Identify the tolerance value of square, hexagon and circle from the observed values and write a program for classifying the objects based on shape.

18. Create a string variable, by clicking the “S” button, in the Variables tool bar and rename it to “shape”.

19. Click on the “Rect A” in the program window before your script insertion and click on the “Script” button in the Program window toolbar.

20. Double-left-click on the “script A” instruction in the Program window to open the JavaScript editor. There are three panes in the Script editor window: Code, Variables, and Predefined program items. The code window is enabled to Auto- Complete on some of the predefined items or objects.

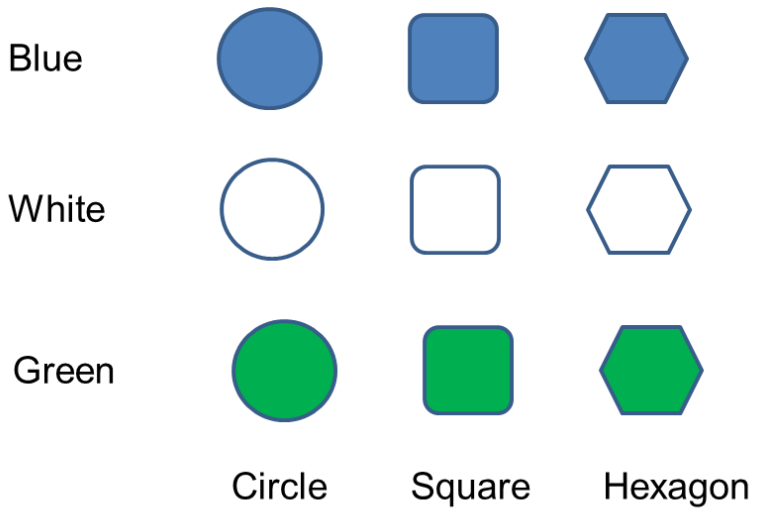
21. Click on Run for executing the script in the editor window.

22. Click OK

23. Load images of different objects and run continuously and thus classify objects based on color.

Aim: Identification and classification of the objects based on color using suitable algorithms.

Images of objects to be classified based on color



Procedure:

1. Start Sherlock by double clicking the Sherlock icon on your desktop, or in the Sherlock program group.
2. Click the Load Image button in the Image window tool bar and load the image file.
3. Create a ROI, click on the rectangle button, in the Image Window toolbar and Draw a box around the object by clicking the left mouse button.
4. Displaying the ROI options dialog: double-click inside the ROI, or double-click on the ROI name in the Program window, or right-click inside the ROI and select “edit”.
5. Choose an operator in the preprocessor list boxes (drop list, list box, or combobox). For this select ‘Normalize by chroma’ for the first list box.
6. Select an ‘Average [channel]’ operator from the algorithm list boxes below the last preprocessor list box.
7. Click the “Close” button.

8. Click on the “Variables” tab at the bottom of the Instructions window.
9. Create three number variables, by clicking the “N” button thrice, in the Variables tool bar.
10. In the Program window, right-click on “red average” and select “connect variable,” “varA”.
11. Double click on “varA” and rename it to “Red_average”.
12. Do step 10 and 11 for “blue_average” and “green_average”.
13. Run and Save the Basic Investigation. Press F9, or in the Menu bar select Run Once. Sherlock calculates the average color inside the rectangle. The values are displayed in the Variables window.
14. Load images of blue, green and white colored objects and run the investigation. Observe RGB values which are displayed in the variable window.
15. Create a string variable, by clicking the “S” button, in the Variables tool bar and rename it to “color”.
16. Identify the tolerance value of green, blue and white components from the observed values and write a program for classifying the objects based on color.
17. Click on the “Rect A” in the program window before your script insertion and click on the “Script” button in the Program window toolbar.
18. Double-left-click on the “scriptA” instruction in the Program window to open the JavaScript editor. There are three panes in the Script editor window: Code, Variables, and Predefined program items. The code window is enabled to auto-complete on some of the predefined items or objects.
19. Click on Run for executing the script in the editor window.
20. Click OK
21. Load images of different objects and run continuously and thus classify objects based on color.

POST LAB EXERCISES:

Identify the shape of the object in the image and communicate with the **Hercules (a dummy client software)** through TCP/IP protocol.

Steps for serial communication

1. Right click on the white space of program window and select “IO: Tcp/Ip”.
2. For sending data, select “Send line A”.
3. For receiving data, select “Receive line A”.
4. Select options□IO□Tcp/Ip

5. Select Server or Client mode, enter an IP Address, and click “Add” to add a connection.
In Server mode, a client will connect to Sherlock using the assigned IP Port number entered here. In Client mode, Sherlock must connect to a Server, using the Server’s IP Address and Port number. You must enter the Server’s correct address and port.
6. Click the “Close” button
7. Connect the variable “shape” to the “text”to be sent under “send line A”.
8. Repeat the same for receiving data through Tcp/Ip connection.
9. Run program continuously

Identify the color of the object in the image and glow corresponding LEDs using Arduino
Steps for serial communication
1. Right click on the white space of program window and select “IO: Serial”.
2. Serial Options allow you to configure the settings of the Serial Ports available on your PC. The Serial port must be configured before it can be used with I/O Control Operations. Select options□IO□serial.
3. Select the serial port and set communication options. Click “Close” to accept your changes.
4. Run program continuously.

4- IMAGE PROCESSING USING OPENCV

Machine vision is used in a variety of industrial processes material inspection, object recognition, pattern recognition, barcode reading and counting, electronic component analysis, medical field food and beverage along with the recognition of signatures, optical characters, and currency. Machine vision is the capability of a computer to perceive the environment. One or more video cameras are used with analog-to-digital conversion and digital signal processing.

Machine vision is usually linked with a computer's ability to see. The term, computer vision, is used to designate the technology in which a computer digitizes an image, processes the data, and takes some type of action.

A machine vision system uses a sensor in the robot for viewing and recognizing an object with the help of a computer. Machine vision is used in a variety of industrial processes, such as material inspection, object recognition, pattern recognition, electronic component analysis, along with the recognition of signatures, optical characters, and currency.

A robot’s vision system consists of several essential components, which include the camera that captures a picture, to the processing mechanism that provides and communicates the result. For any machine vision system to work dependably and produce results that can be repeated, it is important how these essential components interact.

Lighting is very important to machine vision as it illuminates the part to be viewed so that its features stand out allowing the camera to see clearly. The lens captures the image and communicates it to the sensor in the form of light. The sensor in a machine vision camera translates this light into a digital image, which is then relayed to the processor for analysis.

Vision processing employs algorithms that review the image and extricate the required information, run the required inspection, and make a decision. Finally, communication is accomplished by either a discrete I/O signal or data transmitted over a serial connection to a device that is logging or using the information.

A robot's vision system is sorted into three main categories based on the color of the objects. These are (1) binary image, which consists of black and white images; (2) gray images; (3) images based on the colors of red, green or blue. An electronic image is established using pixels classified into these three categories.

Aim: Finding The location of Ball using OpenCV

✓ **Open Terminal Window**

gedit hsv_ball_detect.py

Write the following code:

```
#!/usr/bin/env python

import cv2
import numpy as np

image=cv2.imread('robolab/red1.jpg')
cv2.imshow("image", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
def mouse(event,x,y,flags,param):
        if event==cv2.EVENT_LBUTTONDOWN:
                h=hsv[y,x,0]
                s=hsv[y,x,1]
                v=hsv[y,x,2]
                print("H:",h)
                print("S:",s)
                print("V:",v)

cv2.namedWindow('mouse')
cv2.setMouseCallback('mouse',mouse)
image=cv2.imread('robolab/red1.jpg')

#cap=cv2.VideoCapture(0)
#ret,image=cap.read()
#cap.release()

cv2.imshow("original image", image)

hsv=cv2.cvtColor(image,cv2.COLOR_BGR2HSV)
cv2.imshow("mouse", hsv)



cv2.waitKey(0)
cv2.destroyAllWindows()

redLower=(0,160,80)
#varies based on the color of the ball
redUpper=(30,255,255)
```

```
mask=cv2.inRange(hsv,redLower,redUpper)
cv2.imshow("mask",mask)
cv2.waitKey(0)
cv2.destroyAllWindows()

kernel=np.ones((3,3),np.uint8)
mask=cv2.dilate(mask,kernel,iterations=5)
_, contours, hierarchy=cv2.findContours(mask,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_NONE)
cv2.imshow("dilate",mask)
cv2.waitKey(0)
cv2.destroyAllWindows()
print("Number of contours found="+str(len(contours)))


def get_contour_center(contour):
        M=cv2.moments(contour)
        cx=-1
        cy=-1
        if(M['m00']!=0):
                cx=int(M['m10']/M['m00'])
                cy=int(M['m01']/M['m00'])
        return cx,cy


for c in contours:
        area = cv2.contourArea(c)
        perimeter = cv2.arcLength(c, True)
        ((x,y),radius)= cv2.minEnclosingCircle(c)
        if (area>1000):
```

```
            cv2.drawContours(image,[c],-1,(150,250,150),1)
            cv2.drawContours(mask,[c],-1,(150,250,150),1)
            cx, cy = get_contour_center(c)
            cv2.circle(image, (cx,cy), (int)(radius), (0,0,255),3)
            #cv2.circle(mask, (cx,cy), (int)(radius), (0,0,255),3)
            #cv2.circle(mask, (cx,cy), 5, (0,0,255),-1)
            print("Area.{},Perimeter.{}".format(area,perimeter))
            print("number of contours:{}".format(len(contours)))
            cv2.imshow("RGB", image)
            cv2.imshow("BW", mask)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

✓ python hsv_ball_detect.py

Press enter key

✓ **Using WebCam:**

```
#!/usr/bin/env python

import cv2
import numpy as np

cap=cv2.VideoCapture(0)
```

```
ret,image=cap.read()
cap.release()
cv2.imshow("image", image)
cv2.waitKey(0)
cv2.destroyAllWindows()


def mouse(event,x,y,flags,param):
        if event==cv2.EVENT_LBUTTONDOWN:
                h=hsv[y,x,0]
                s=hsv[y,x,1]
                v=hsv[y,x,2]
                print("H:",h)
                print("S:",s)
                print("V:",v)

cv2.namedWindow('mouse')
cv2.setMouseCallback('mouse',mouse)


cap=cv2.VideoCapture(0)
ret,image=cap.read()
cap.release()

cv2.imshow("original image", image)

hsv=cv2.cvtColor(image,cv2.COLOR_BGR2HSV)
cv2.imshow("mouse", hsv)


cv2.waitKey(0)
```

```
cv2.destroyAllWindows()

redLower=(0,160,120)
#varies based on the color of the ball
redUpper=(30,255,255)


mask=cv2.inRange(hsv,redLower,redUpper)
cv2.imshow("mask",mask)
cv2.waitKey(0)
cv2.destroyAllWindows()

kernel=np.ones((3,3),np.uint8)
mask=cv2.dilate(mask,kernel,iterations=3)
_,                                                                                    contours,
hierarchy=cv2.findContours(mask,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_NONE)
cv2.imshow("dilate",mask)
cv2.waitKey(0)
cv2.destroyAllWindows()
print("Number of contours found="+str(len(contours)))


def get_contour_center(contour):
        M=cv2.moments(contour)
        cx=-1
        cy=-1
        if(M['m00']!=0):
                cx=int(M['m10']/M['m00'])
                cy=int(M['m01']/M['m00'])
        return cx,cy
```

```
for c in contours:
        area = cv2.contourArea(c)
        perimeter = cv2.arcLength(c, True)
        ((x,y),radius)= cv2.minEnclosingCircle(c)
        if (area>1000):
                cv2.drawContours(image,[c],-1,(150,250,150),1)
                cv2.drawContours(mask,[c],-1,(150,250,150),1)
                cx, cy = get_contour_center(c)
                cv2.circle(image, (cx,cy), (int)(radius), (0,0,255),3)
                #cv2.circle(mask, (cx,cy), (int)(radius), (0,0,255),3)
                #cv2.circle(mask, (cx,cy), 5, (0,0,255),-1)
                print("Area.{},Perimeter.{}".format(area,perimeter))
                print("number of contours:{}".format(len(contours)))
                cv2.imshow("RGB", image)
                cv2.imshow("BW", mask)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Aim: Finding Ball Location in OpenCV

```
#!/usr/bin/env python

# Standard imports
import cv2
import numpy as np;
```

```
def blob_detect(image,hsv_min,hsv_max,blur=0,blob_params=None,imshow=False):

    if blur > 0:
        image=cv2.blur(image, (blur, blur))

        if imshow:
            cv2.imshow("Blur", image)
            cv2.waitKey(0)

    # BGR to HSV
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

    #- Apply HSV threshold
    mask = cv2.inRange(hsv,hsv_min, hsv_max)

    #- Show HSV Mask
    if imshow:
        cv2.imshow("HSV Mask", mask)

    #- dilate makes the in range areas larger
    mask = cv2.dilate(mask, None, iterations=2)

    if imshow:
        cv2.imshow("Dilate Mask", mask)
        cv2.waitKey(0)

    mask = cv2.erode(mask, None, iterations=2)

    #- Show dilate/erode mask
    if imshow:
```

```
    cv2.imshow("Erode Mask", mask)
    cv2.waitKey(0)


if blob_params is None:
    # Set up the SimpleBlobdetector with default parameters.
    params = cv2.SimpleBlobDetector_Params()

    # Change thresholds
    params.minThreshold = 0;
    params.maxThreshold = 50;

    # Filter by Area.
    params.filterByArea = True
    params.minArea = 5000
    params.maxArea = 500000

    # Filter by Circularity
    params.filterByCircularity = True
    params.minCircularity = 0.1

    # Filter by Convexity
    params.filterByConvexity = True
    params.minConvexity = 0.5

    # Filter by Inertia
    params.filterByInertia =True
    params.minInertiaRatio = 0.5

else:
    params = blob_params
```

```
    #- Apply blob detection
    detector = cv2.SimpleBlobDetector_create(params)

    # Reverse the mask: blobs are black on white
    reversemask = 255-mask

    if imshow:
        cv2.imshow("Reverse Mask", reversemask)
        cv2.waitKey(0)

    keypoints = detector.detect(reversemask)

    return keypoints, reversemask


def draw_keypoints(image,keypoints,line_color=(255,0,255),imshow=True):

    im_with_keypoints = cv2.drawKeypoints(image, keypoints, np.array([]), line_color,
cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

    if imshow:
        cv2.imshow("Keypoints", im_with_keypoints)

    return(im_with_keypoints)

def draw_window(image,   window_adim,  color=(255,0,0),  line=5,  imshow=False ):

    rows = image.shape[0]
    cols = image.shape[1]
```

```
    x_min_px    = int(cols*window_adim[0])
    y_min_px    = int(rows*window_adim[1])
    x_max_px    = int(cols*window_adim[2])
    y_max_px    = int(rows*window_adim[3])

    #-- Draw a rectangle from top left to bottom right corner
    image = cv2.rectangle(image,(x_min_px,y_min_px),(x_max_px,y_max_px),color,line)

    if imshow:
        cv2.imshow("Keypoints", image)

    return(image)

def apply_search_window(image, window_adim=[0.0, 0.0, 1.0, 1.0]):
    rows = image.shape[0]
    cols = image.shape[1]
    x_min_px    = int(cols*window_adim[0])
    y_min_px    = int(rows*window_adim[1])
    x_max_px    = int(cols*window_adim[2])
    y_max_px    = int(rows*window_adim[3])

    #--- Initialize the mask as a black image
    mask = np.zeros(image.shape,np.uint8)

    #--- Copy the pixels from the original image corresponding to the window
    mask[y_min_px:y_max_px,x_min_px:x_max_px] =
image[y_min_px:y_max_px,x_min_px:x_max_px]

    #--- return the mask
    return(mask)
```

```
def get_blob_relative_position(image, keyPoint):
    rows = float(image.shape[0])
    cols = float(image.shape[1])
    # print(rows, cols)
    center_x    = 0.5*cols
    center_y    = 0.5*rows
    # print(center_x)
    x = (keyPoint.pt[0] - center_x)/(center_x)
    y = (keyPoint.pt[1] - center_y)/(center_y)
    return(x,y)

#Find the HSV value using demo1

red_min = (100,75,130)
red_max = (180,250,255)

string1="first frame"
string2=""
string3=""
window = [0,0,1,1]
cap = cv2.VideoCapture(0)
while(True):

    ret, frame = cap.read()
    keypoints, _ = blob_detect(frame, red_min, red_max, blur=3, blob_params=None,
imshow=False)
    final_image= draw_keypoints(frame, keypoints, imshow=True)
    cv2.putText(final_image,string1, (25,25), cv2.FONT_HERSHEY_COMPLEX,1,(255,0,0),1)
    cv2.putText(final_image,string2, (25,50), cv2.FONT_HERSHEY_COMPLEX,1,(255,0,0),1)
    cv2.putText(final_image,string3, (25,75), cv2.FONT_HERSHEY_COMPLEX,1,(255,0,0),1)
```

```
    cv2.imshow("final image", final_image)
    for i, keyPoint in enumerate(keypoints):
        x = keyPoint.pt[0]
        y = keyPoint.pt[1]
        s = keyPoint.size
        print ("kp %d: s = %3d   x = %3d  y= %3d"%(i, s, x, y))

        x, y = get_blob_relative_position(frame, keyPoint)
        print (" x = %3d  y= %3d"%(x, y))
        string1= 'x='+str(x)
        string2= 'y='+str(y)
            string3= 's='+str(s)

    key=cv2.waitKey(1) & 0xFF
    if key == ord('q'):
            break
cap.release()
cv2.destroyAllWindows()
```

Aim: Ball Detection in OpenCV+ROS

- ✓ **Install the following:**

$sudo apt-get update

$sudo apt-get install ros-kinetic-opencv3

$sudo apt-get install ros-kinetic-usb-cam

$sudo apt-get install ros-kinetic-image-view

$cd catkin_ws/src

$catkin_create_pkg sample_opencv_pkg sensor_msgs cv_bridge rospy std_msgs

$cd sample_opencv_pkg/src

mkdir scripts

$cd scripts

$gedit read_opencv.py

Write the following code:

```
#!/usr/bin/env python

import rospy
import cv2
from std_msgs.msg import String
from sensor_msgs.msg import Image, CameraInfo
from cv_bridge import CvBridge,CvBridgeError
import sys

bridge=CvBridge()


def get_contour_center(contour):
        M=cv2.moments(contour)
        cx=-1
        cy=-1
        if(M['m00']!=0):
                cx=int(M['m10']/M['m00'])
                cy=int(M['m01']/M['m00'])
        return cx,cy

def image_callback(ros_image):
        print('got an image')
```

```
        global bridge
        try:
                image=bridge.imgmsg_to_cv2(ros_image,"bgr8")
                hsv=cv2.cvtColor(image,cv2.COLOR_BGR2HSV)
                cv2.imshow("image",image)
                #cv2.imshow("hsv", hsv)
                #cv2.waitKey(0)
                #cv2.destroyAllWindows()
        except CvBridgeError as e:
                print(e)



        redLower=(150,5,130)
        redUpper=(190,150,180)

        mask=cv2.inRange(hsv,redLower,redUpper)
        gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
        #mask=cv2.adaptiveThreshold(gray,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
cv2.THRESH_BINARY,5,2)
        #_,mask=cv2.threshold(gray,127,255,cv2.THRESH_BINARY)

        cv2.imshow("mask",mask)


        _,                                contours,                                hierarchy=
cv2.findContours(mask,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)

        for c in contours:
                area = cv2.contourArea(c)
                perimeter = cv2.arcLength(c, True)
```

```
                ((x,y),radius)= cv2.minEnclosingCircle(c)
                if (area>40000):
                        cv2.drawContours(image,[c],-1,(150,250,150),2)
                        cv2.drawContours(mask,[c],-1,(150,250,150),2)
                        cx, cy = get_contour_center(c)
                        cv2.circle(image, (cx,cy), (int)(radius), (0,0,255),3)
                        #cv2.circle(mask, (cx,cy), (int)(radius), (0,0,255),3)
                        #cv2.circle(mask, (cx,cy), 5, (0,0,255),-1)
                        print("Area.{},Perimeter.{}".format(area,perimeter))
                        print("number of contours:{}".format(len(contours)))
                        cv2.imshow("RGB", image)
                        cv2.imshow("BW", mask)


                        key=cv2.waitKey(1) & 0xFF
                        if(key==ord('q')):
                                cv2.destroyAllWindows()


def main(args):
        rospy.init_node('image_converter',anonymous=True)
        image_sub=rospy.Subscriber("/usb_cam/image_raw", Image,image_callback)
        try:
                rospy.spin()
        except KeyboardInterrupt:
                print("Shutting down")
                cv2.destroyAllWindows()

if __name__ == '__main__':
        main(sys.argv)
```

1. **Open terminal**

   $roscore

2. **Open new terminal**

   $rosrun usb_cam usb_cam_node _pixel_format:=yuyv

3. **Open new terminal**

   $rosrun sample_opencv_pkg read_opencv.py

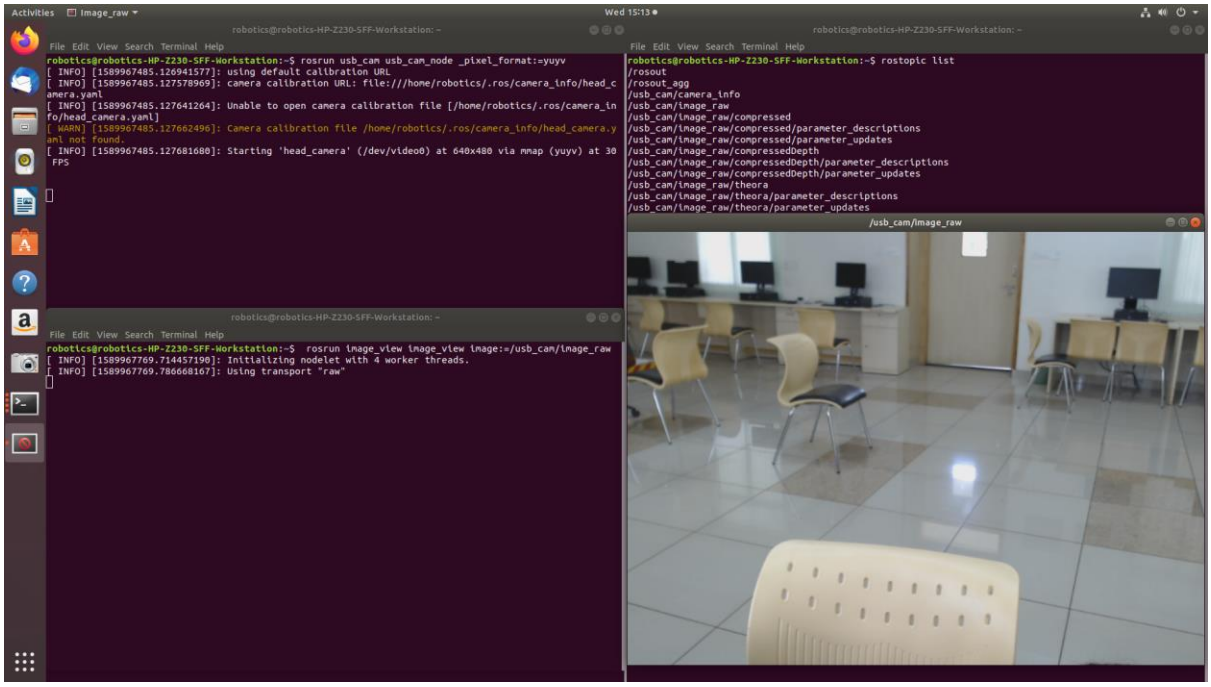
Post Lab exercises

1. Display the location of the ball from the base position using OpenCV + ROS (using RGB image)

2. Display the location of the ball from the base position using OpenCV + ROS (using depth image)

Reference: pyimagesearch.com

docs.opencv.org





5– Depth Camera Image Processing using OpenCV

Installation for Realsense D435

References:

https://www.programmersought.com/article/78655896884/
https://github.com/IntelRealSense/librealsense/blob/master/doc/distribution_linux.md
https://github.com/IntelRealSense/realsense-ros
https://www.programmersought.com/article/55044716391/
https://github.com/IntelRealSense/librealsense/blob/master/doc/installation.md

Steps for instalation of realsense deth camera D435

click on top left corner: software updater->click update->restart now

sudo apt-key adv --keyserver keys.gnupg.net --recv-key F6E65AC044F831AC80A06380C8B3A55A6F3EFCDE || sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-key F6E65AC044F831AC80A06380C8B3A55A6F3EFCDE

sudo add-apt-repository "deb http://realsense-hw-public.s3.amazonaws.com/Debian/apt-repo xenial main" -u

sudo apt-get install librealsense2-dkms

sudo apt-get install librealsense2-utils

sudo apt-get install librealsense2-dev

modinfo uvcvideo | grep "version:"

download source code from https://github.com/IntelRealSense/librealsense/releases/tag/v2.33.1
extract and rename as librealsense.
or
git clone https://github.com/IntelRealSense/librealsense.git

sudo apt-get update

sudo apt-get install git libssl-dev libusb-1.0-0-dev pkg-config libgtk-3-dev

cd librealsense/

sudo cp config/99-realsense-libusb.rules /etc/udev/rules.d/

sudo apt-get update

sudo add-apt-repository ppa:ubuntu-toolchain-r/test

sudo apt-get install gcc

mkdir build && cd build

cmake ../

cmake ../ -DBUILD_EXAMPLES=true

sudo make uninstall && make clean && make && sudo make install

cd ..

```
cd ..

cd catkin_ws/src

git clone https://github.com/IntelRealSense/realsense-ros.git

export ROS_VER=kinetic

sudo apt-get install ros-$ROS_VER-realsense2-camera
sudo apt-get install ros-$ROS_VER-realsense2-description

cd realsense-ros/

git checkout `git tag | sort -V | grep -P "^2.\d+\.\d+" | tail -1`

cd ..
cd ..

catkin_make clean

catkin_make -DCATKIN_ENABLE_TESTING=False -DCMAKE_BUILD_TYPE=Release

catkin_make install

cd ..

echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc

source ~/.bashrc

sudo apt-get install ros-kinetic-rgbd-launch
```

cd catkin_ws

catkin_make

connect camera

open terminal

$roscore

$roslaunch realsense2_camera rs_camera.launch

$roslaunch realsense2_camera rs_rgbd.launch

In a new terminal type

$rostopic list

In a new terminal type

$rosrun image_view image_view image:=/camera/color/image_raw

$rosrun image_view image_view image:=/camera/depth/image_rect_raw

Install astra camera (Kubuki Turtlebot2)

Follow the steps for installation from http://wiki.ros.org/astra_camera

Install astra camera (Kubuki Turtlebot2)

Follow the steps for installation from http://wiki.ros.org/astra_camera

$sudo apt install ros-*-rgbd-launch ros-*-libuvc ros-*-libuvc-camera ros-*-libuvc-ros

cd /catkin_ws/src

$git clone https://github.com/orbbec/ros_astra_camera

Above steps are done only once to install the drivers

After installation

$roscd astra_camera

$./scripts/create_udev_rules

$/catkin_ws

$catkin_make --pkg astra_camera

$roslaunch astra_camera astra.launch

open another terminal and type

$rostopic list

open another terminal and type

$rosrun image_view image_view image:=/camera/rgb/image_raw

rosrun image_view image_view image:=/camera/depth/image

✓ Depth Camera Programming:

$cd catkin_ws/

$cd src

$catkin_create_pkg depth_opencv_pkg sensor_msgs cv_bridge rospy std_msgs

$cd depth_opencv_pkg/src

$mkdir scripts

$gedit depth_camera.py

```
#!/usr/bin/env python
from __future__ import print_function
import rospy
import cv2
```

```
from std_msgs.msg import String
from sensor_msgs.msg import Image, CameraInfo
from cv_bridge import CvBridge,CvBridgeError
import sys
import numpy as np

bridge=CvBridge()

def rgb_image_callback(ros_image):
        print('got an image')
        global bridge
        try:
                rgb_image=bridge.imgmsg_to_cv2(ros_image,"bgr8")
        except CvBridgeError as e:
                print(e)

        rgb_array=np.array(rgb_image,dtype=np.uint8)
        edge=process_image(rgb_array)
        cv2.imshow("RGB Image window",rgb_array)
        cv2.imshow("Edges detected",edge)
        cv2.waitKey(3)

def depth_image_callback(ros_image):
        print('got an image')
        global bridge
        try:
                depth_image=bridge.imgmsg_to_cv2(ros_image,"32FC1")
        except CvBridgeError as e:
                print(e)

        depth_array=np.array(depth_image,dtype=np.float32)
```

```
        cv2.normalize(depth_array,depth_array,0,1,cv2.NORM_MINMAX)
        cv2.imshow("Depth Image window",depth_array)
        cv2.waitKey(3)

def process_image(RGB_image):
        grey=cv2.cvtColor(RGB_image, cv2.COLOR_BGR2GRAY)
        grey=cv2.blur(grey,(7,7))
        edges=cv2.Canny(grey,15.0,30.0)
        return edges

def main(args):
        rospy.init_node('image_converter',anonymous=True)
        image_depth_sub=rospy.Subscriber("/camera/depth/image_raw",
Image,depth_image_callback)
        image_rgb_sub=rospy.Subscriber("/camera/color/image_rect_raw",
Image,rgb_image_callback)
        try:
                rospy.spin()
        except KeyboardInterrupt:
                print("Shutting down")
                cv2.destroyAllWindows()

if __name__ == '__main__':
        main(sys.argv)
```

✓ **In terminal type:**

```
$roscore
$rostopic list
$rosrun image_view image_view
```

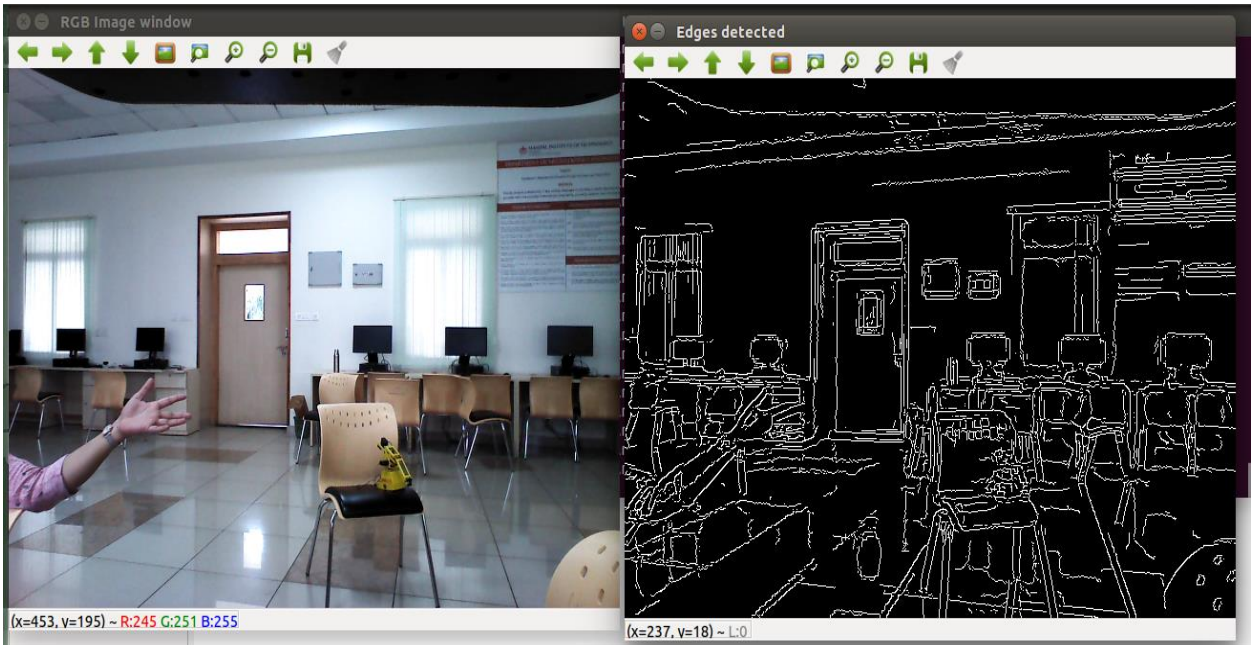
$rosrun depth_opencv_pkg depth_camera.py

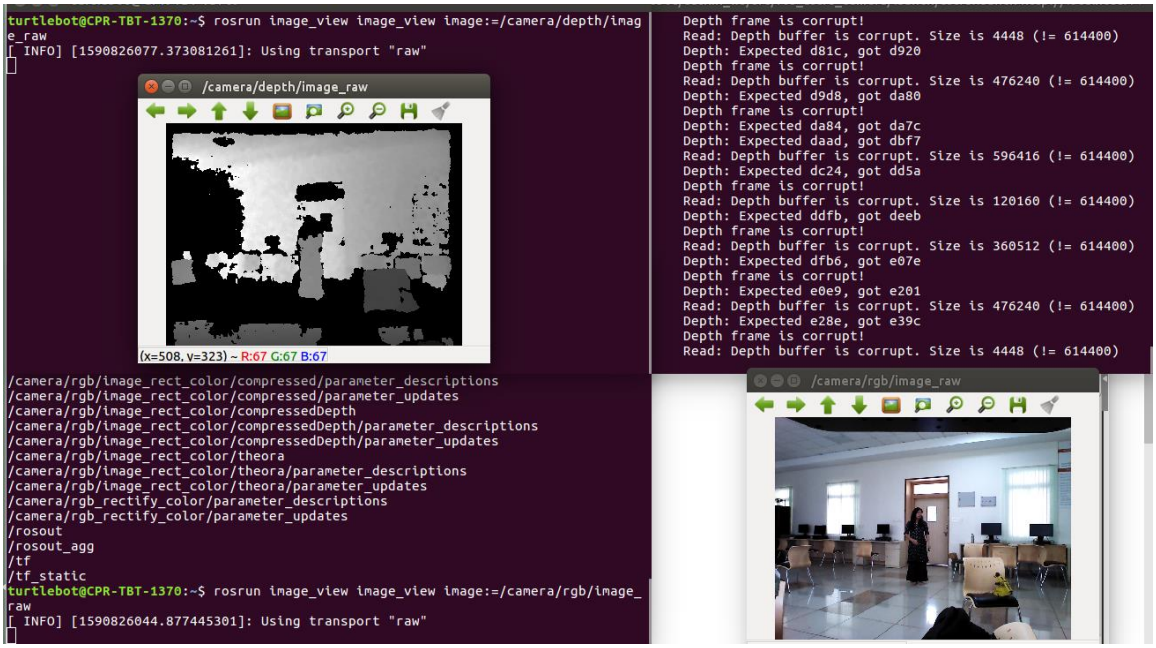
$rosrun depthimage_to_laserscan depthimage_to_laserscan image:=camera/depth/image_raw

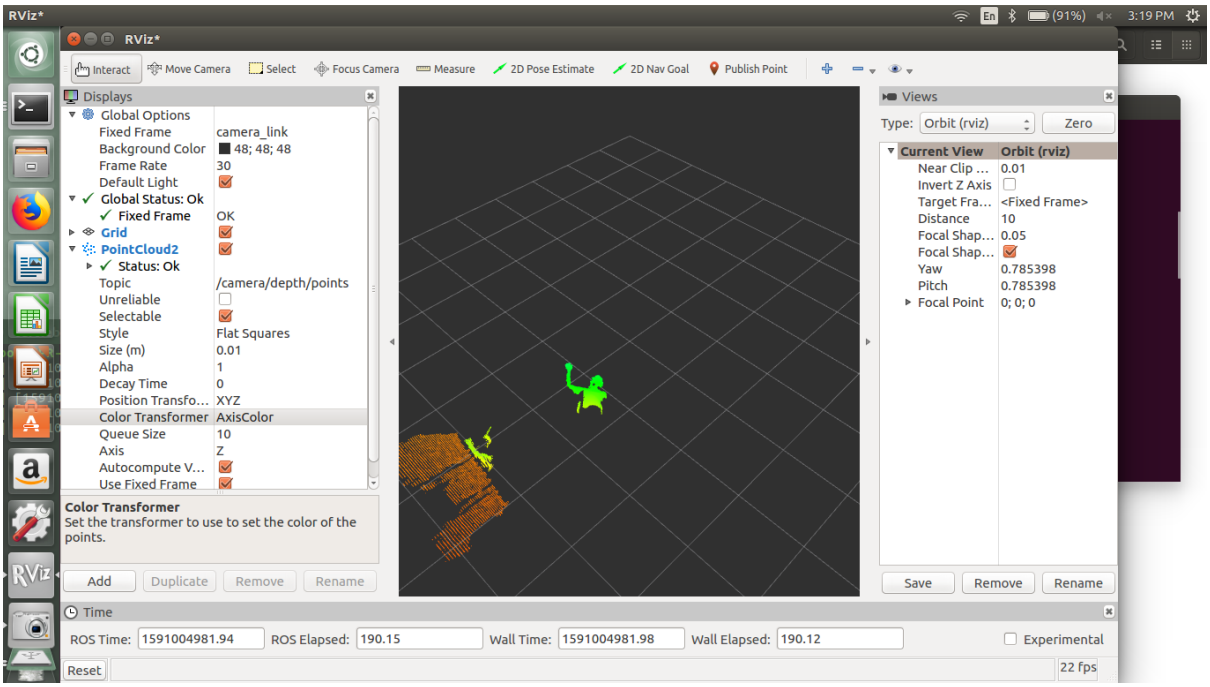
$rostopic list

check /scan

$rostopic echo /scan




turtlebot@CPR-TBT-1370:~$ rosrun image_view image_view image:=/camera/depth/image_raw
[INFO] [1590826077.373081261]: Using transport "raw"
/camera/depth/image_raw
(x=508, y=323) ~ R:67 G:67 B:67
Depth frame is corrupt!
Read: Depth buffer is corrupt. Size is 4448 (!= 614400)
Depth: Expected d81c, got d920
Depth frame is corrupt!
Read: Depth buffer is corrupt. Size is 476240 (!= 614400)
Depth: Expected d9d8, got da80
Depth frame is corrupt!
Depth: Expected da84, got da7c
Depth: Expected daad, got dbf7
Read: Depth buffer is corrupt. Size is 596416 (!= 614400)
Depth: Expected dc24, got dd5a
Depth frame is corrupt!
Read: Depth buffer is corrupt. Size is 120160 (!= 614400)
Depth: Expected ddfb, got deeb
Depth frame is corrupt!
Read: Depth buffer is corrupt. Size is 360512 (!= 614400)
Depth: Expected dfb6, got e07e
Depth frame is corrupt!
Depth: Expected e0e9, got e201
Read: Depth buffer is corrupt. Size is 476240 (!= 614400)
Depth: Expected e28e, got e39c
Depth frame is corrupt!
Read: Depth buffer is corrupt. Size is 4448 (!= 614400)
/camera/rgb/image_rect_color/compressed/parameter_descriptions
/camera/rgb/image_rect_color/compressed/parameter_updates
/camera/rgb/image_rect_color/compressedDepth
/camera/rgb/image_rect_color/compressedDepth/parameter_descriptions
/camera/rgb/image_rect_color/compressedDepth/parameter_updates
/camera/rgb/image_rect_color/theora
/camera/rgb/image_rect_color/theora/parameter_descriptions
/camera/rgb/image_rect_color/theora/parameter_updates
/camera/rgb_rectify_color/parameter_descriptions
/camera/rgb_rectify_color/parameter_updates
/rosout
/rosout_agg
/tf
/tf_static
turtlebot@CPR-TBT-1370:~$ rosrun image_view image_view image:=/camera/rgb/image_raw
[INFO] [1590826044.877445301]: Using transport "raw"
/camera/rgb/image_raw


RViz*
Interact
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
2D Nav Goal
Publish Point
Displays
Global Options
Fixed Frame camera_link
Background Color 48; 48; 48
Frame Rate 30
Default Light
Global Status: Ok
Fixed Frame OK
Grid
PointCloud2
Status: Ok
Topic /camera/depth/points
Unreliable
Selectable
Style Flat Squares
Size (m) 0.01
Alpha 1
Decay Time 0
Position Transfo... XYZ
Color Transformer AxisColor
Queue Size 10
Axis Z
Autocompute V...
Use Fixed Frame
Color Transformer
Set the transformer to use to set the color of the points.
Add
Duplicate
Remove
Rename
Views
Type: Orbit (rviz)
Zero
Current View Orbit (rviz)
Near Clip ... 0.01
Invert Z Axis
Target Fra... <Fixed Frame>
Distance 10
Focal Shap... 0.05
Focal Shap...
Yaw 0.785398
Pitch 0.785398
Focal Point 0; 0; 0
Save
Remove
Rename
Time
ROS Time: 1591004981.94
ROS Elapsed: 190.15
Wall Time: 1591004981.98
Wall Elapsed: 190.12
Experimental
Reset
22 fps

6 – Robot Motion in ROS Gazebo/RViz simulation environment.

Aim:

To familiarize with the turtlebot simulation environment, Gazebo and RViz.

Method:

Gazebo is a simulation platform for robots, where, the physical behavior of the robot can be simulated in parallel with the ROS software that controls it. Rviz is a 3D visualization tool for ROS applications. The following exercises introduces to the Gazebo and Rviz environment.

✓ **Install Rviz and Gazebo**

$sudo apt-get update

Rviz Installation:

$sudo apt-get install ros-kinetic-turtlebot-*

$roslaunch turtlebot_stage turtlebot_in_stage.launch

$ sudo apt-get install ros-kinetic-turtlebot-rviz-launchers

Gazebo Installation:

$sudo apt-get install ros-kinetic-turtlebot-gazebo

1) Exercise 1: to move the robot through keyboard commands in gazebo

✓ Start ROS master:

$roscore

✓ In a separate terminal type the following to open gazebo:

$roslaunch turtlebot_gazebo turtlebot_world.launch

✓ In a separate terminal type the following to control your robot from keyboard:

$roslaunch turtlebot_teleop keyboard_teleop.launch

2) Exercise 2: To move the robot through python code

✓ Create package move_robot in catkin workspace

$cd catkin_ws/src

$ catkin_create_pkg move_robot roscpp rospy std_msgs

✓ Write the Python code move_turtlebot.py in scripts folder in the package

$cd move_robot/src

$mkdir scripts

$cd scripts

$gedit move_turtlebot.py

Write the following code:

```
#!/usr/bin/env python
import rospy
from geometry_msgs.msg import Twist
from nav_msgs.msg import Odometry
import sys

X=0

def pose_callback(msg):
        global  X
        X=msg.pose.pose.position.x
        rospy.loginfo("Robot X=%f\n",X)

def move(lin_vel,ang_vel,distance):
        global X
        rospy.init_node('move_turtlebot',anonymous=False)
        pub=rospy.Publisher('/cmd_vel_mux/input/teleop',Twist,queue_size=10)
```

```
        rospy.Subscriber('/odom',Odometry,pose_callback)
        rate=rospy.Rate(10)
        vel=Twist()
        while not rospy.is_shutdown():
                vel.linear.x=lin_vel
                vel.linear.y=0
                vel.linear.z=0

                vel.angular.x=0
                vel.angular.y=0
                vel.angular.z=ang_vel

                #rospy.loginfo("Linear Vel=%f: Angular Vel=%f",lin_vel,ang_vel)

                if(X>=distance):
                        rospy.loginfo("Robot Reached Destination")
                        rospy.logwarn("Stopping robot")
                        break
                pub.publish(vel)
                rate.sleep()

if __name__=='__main__':
        try:
                move(float(sys.argv[1]),float(sys.argv[2]),float(sys.argv[3]))

        except rospy.ROSInterruptException:
                pass
```

Run using gazebo

✓ **Start ROS master**

$roscore

✓ **Type in a separate terminal to launch gazebo**

$roslaunch turtlebot_gazebo turtlebot_world.launch

✓ **To move the robot, type the following in a separate terminal**

$rosrun move_robot move_turtlebot.py 0.2 0 3

<u>Run using Rviz:</u>

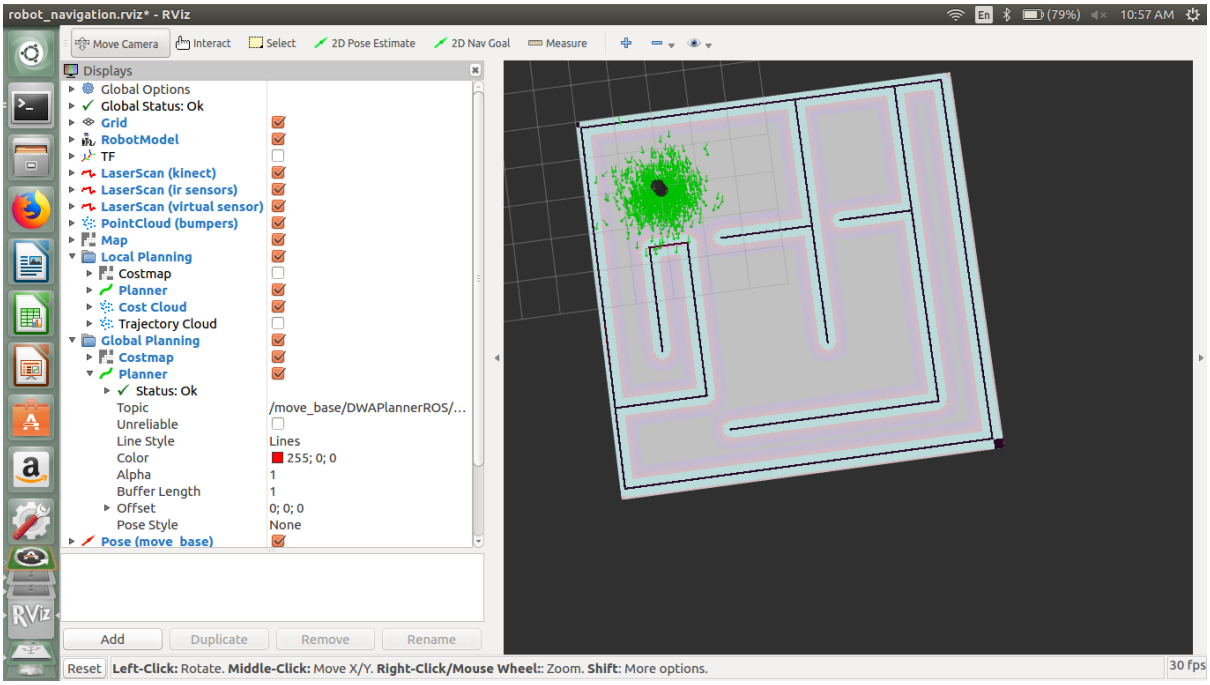
✓ **Start roscore if not already running:**

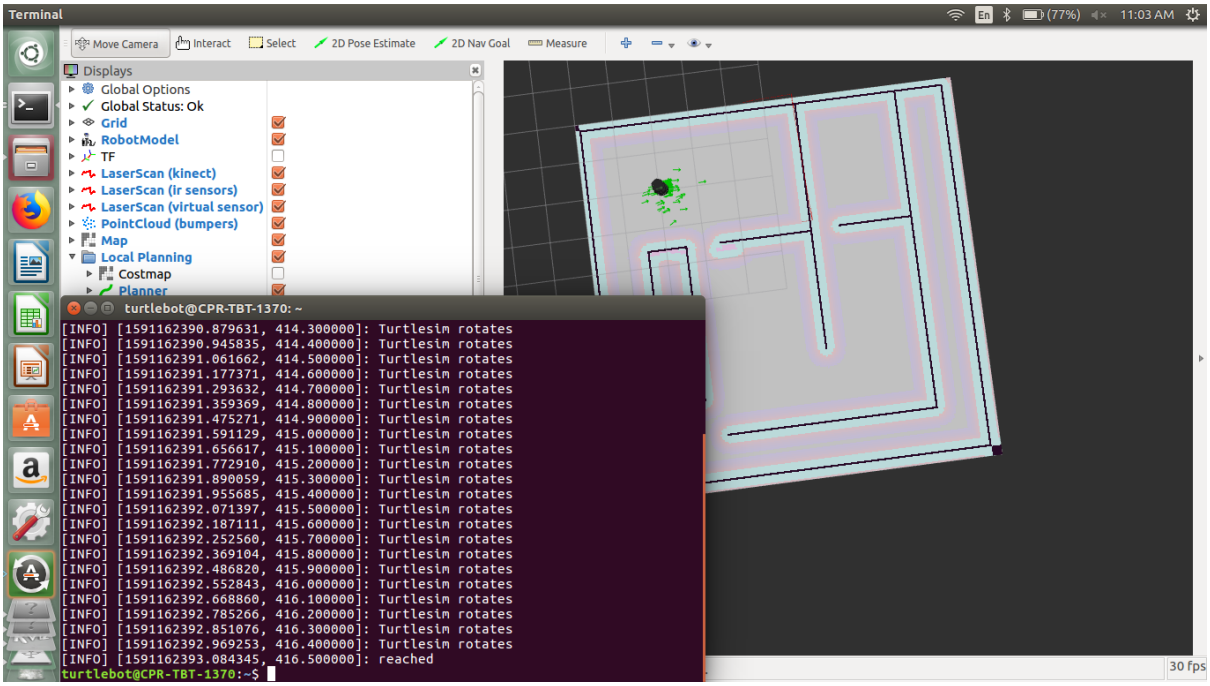
$roscore

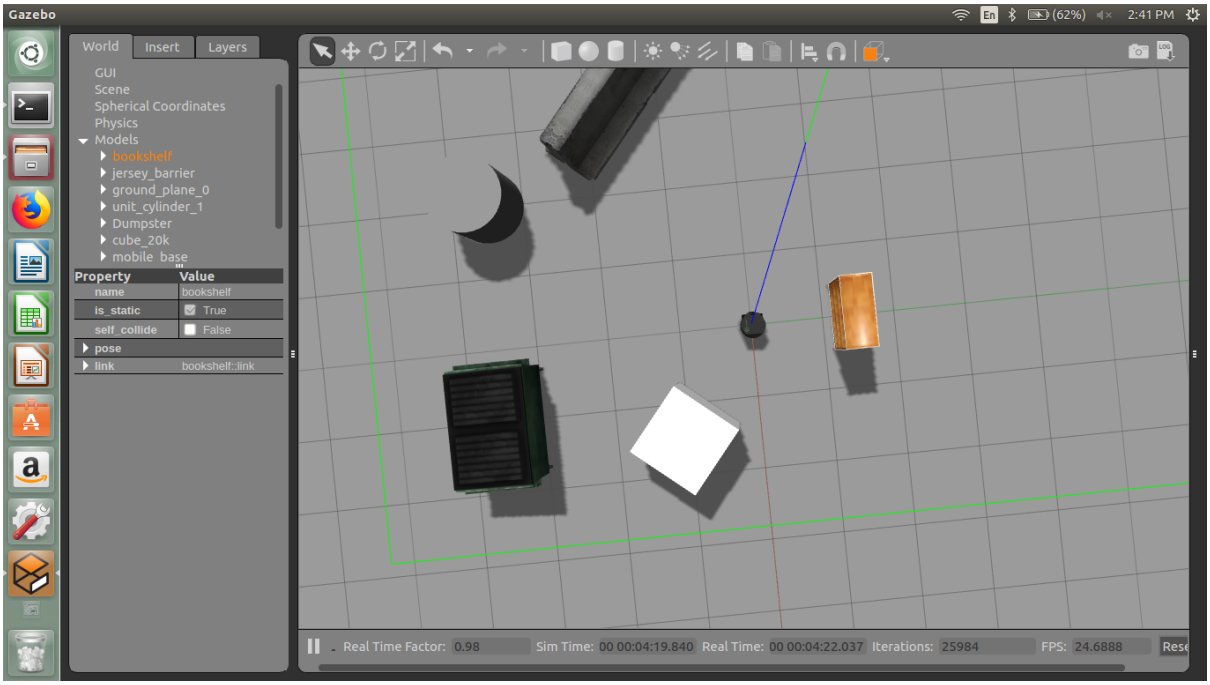
✓ **Type in a separate terminal to launch rviz**

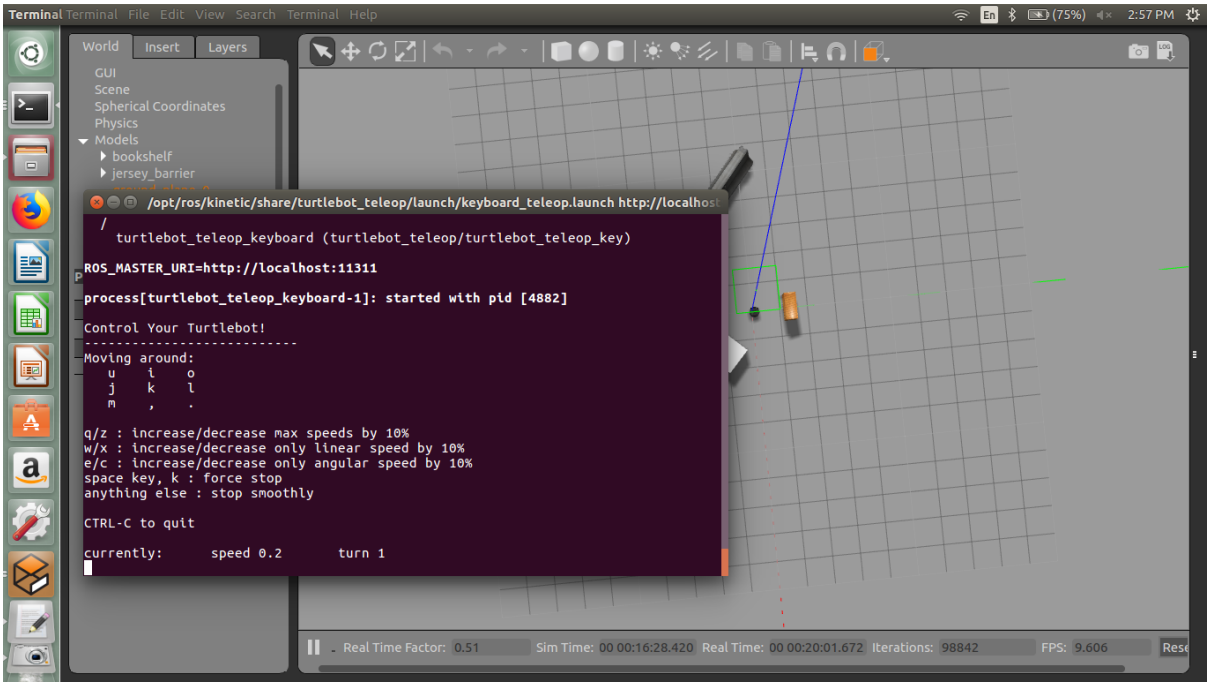
$roslaunch turtlebot_stage turtlebot_in_stage.launch

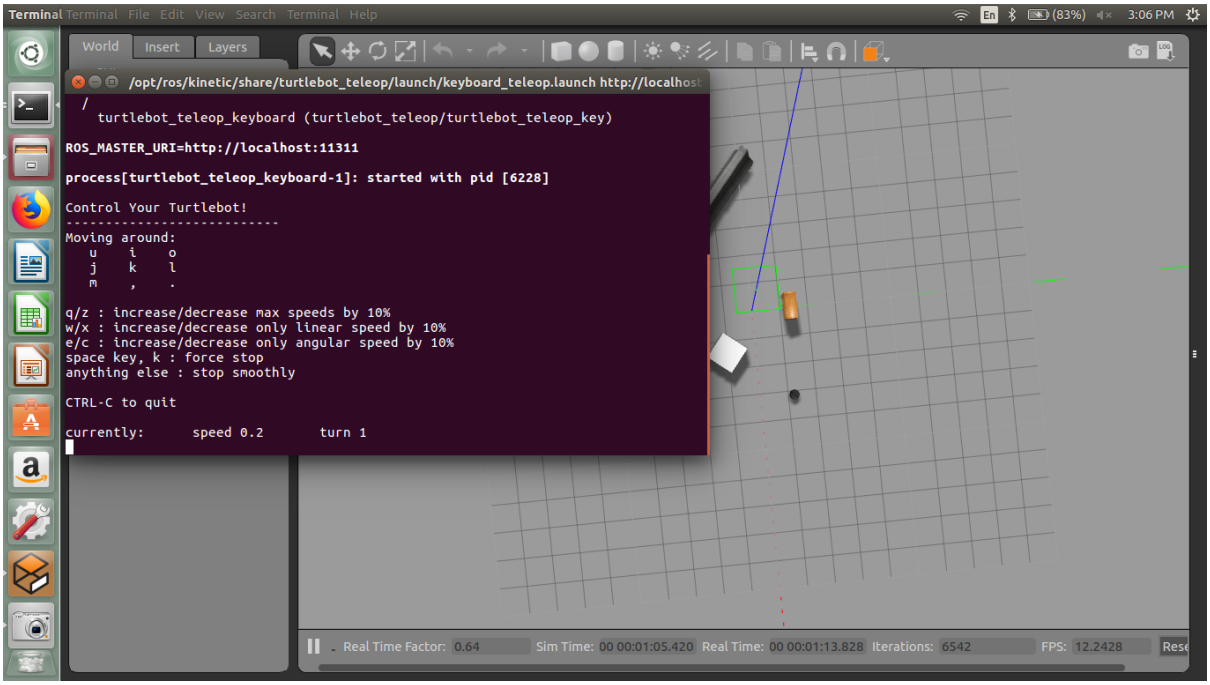
$rosrun move_robot move_turtlebot.py 0.2 0 3

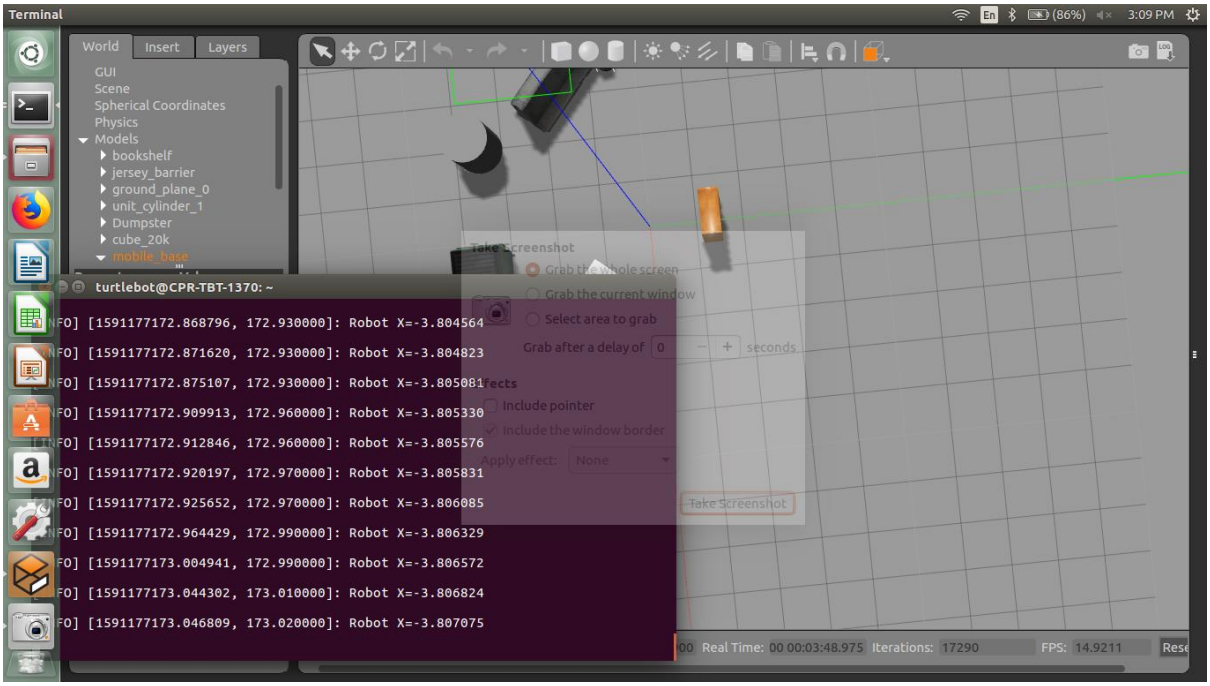



Terminal
(77%) 11:03 AM
Move Camera
Interact
Select
2D Pose Estimate
2D Nav Goal
Measure
Displays
Global Options
Global Status: Ok
Grid
RobotModel
TF
LaserScan (kinect)
LaserScan (ir sensors)
LaserScan (virtual sensor)
PointCloud (bumpers)
Map
Local Planning
Costmap
Planner
turtlebot@CPR-TBT-1370: ~
[INFO] [1591162390.879631, 414.300000]: Turtlesim rotates
[INFO] [1591162390.945835, 414.400000]: Turtlesim rotates
[INFO] [1591162391.061662, 414.500000]: Turtlesim rotates
[INFO] [1591162391.177371, 414.600000]: Turtlesim rotates
[INFO] [1591162391.293632, 414.700000]: Turtlesim rotates
[INFO] [1591162391.359369, 414.800000]: Turtlesim rotates
[INFO] [1591162391.475271, 414.900000]: Turtlesim rotates
[INFO] [1591162391.591129, 415.000000]: Turtlesim rotates
[INFO] [1591162391.656617, 415.100000]: Turtlesim rotates
[INFO] [1591162391.772910, 415.200000]: Turtlesim rotates
[INFO] [1591162391.890059, 415.300000]: Turtlesim rotates
[INFO] [1591162391.955685, 415.400000]: Turtlesim rotates
[INFO] [1591162392.071397, 415.500000]: Turtlesim rotates
[INFO] [1591162392.187111, 415.600000]: Turtlesim rotates
[INFO] [1591162392.252560, 415.700000]: Turtlesim rotates
[INFO] [1591162392.369104, 415.800000]: Turtlesim rotates
[INFO] [1591162392.486820, 415.900000]: Turtlesim rotates
[INFO] [1591162392.552843, 416.000000]: Turtlesim rotates
[INFO] [1591162392.668860, 416.100000]: Turtlesim rotates
[INFO] [1591162392.785266, 416.200000]: Turtlesim rotates
[INFO] [1591162392.851076, 416.300000]: Turtlesim rotates
[INFO] [1591162392.969253, 416.400000]: Turtlesim rotates
[INFO] [1591162393.084345, 416.500000]: reached
turtlebot@CPR-TBT-1370:~$
30 fps


Gazebo
(62%) 2:41 PM
World
Insert
Layers
GUI
Scene
Spherical Coordinates
Physics
Models
bookshelf
jersey_barrier
ground_plane_0
unit_cylinder_1
Dumpster
cube_20k
mobile_base
Property
Value
name
bookshelf
is_static
True
self_collide
False
pose
link
bookshelf::link
Real Time Factor: 0.98
Sim Time: 00 00:04:19.840
Real Time: 00 00:04:22.037
Iterations: 25984
FPS: 24.6888


Terminal Terminal File Edit View Search Terminal Help
2:57 PM
World
Insert
Layers
GUI
Scene
Spherical Coordinates
Physics
Models
bookshelf
jersey_barrier
/opt/ros/kinetic/share/turtlebot_teleop/launch/keyboard_teleop.launch http://localhost
/
turtlebot_teleop_keyboard (turtlebot_teleop/turtlebot_teleop_key)
ROS_MASTER_URI=http://localhost:11311
process[turtlebot_teleop_keyboard-1]: started with pid [4882]
Control Your Turtlebot!
Moving around:
u i o
j k l
m , .
q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%
space key, k : force stop
anything else : stop smoothly
CTRL-C to quit
currently: speed 0.2 turn 1
Real Time Factor: 0.51
Sim Time: 00 00:16:28.420
Real Time: 00 00:20:01.672
Iterations: 98842
FPS: 9.606


Terminal Terminal File Edit View Search Terminal Help
3:06 PM
World
Insert
Layers
/opt/ros/kinetic/share/turtlebot_teleop/launch/keyboard_teleop.launch http://localhost
/
turtlebot_teleop_keyboard (turtlebot_teleop/turtlebot_teleop_key)
ROS_MASTER_URI=http://localhost:11311
process[turtlebot_teleop_keyboard-1]: started with pid [6228]
Control Your Turtlebot!
Moving around:
u i o
j k l
m , .
q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%
space key, k : force stop
anything else : stop smoothly
CTRL-C to quit
currently: speed 0.2 turn 1
Real Time Factor: 0.64
Sim Time: 00 00:01:05.420
Real Time: 00 00:01:13.828
Iterations: 6542
FPS: 12.2428

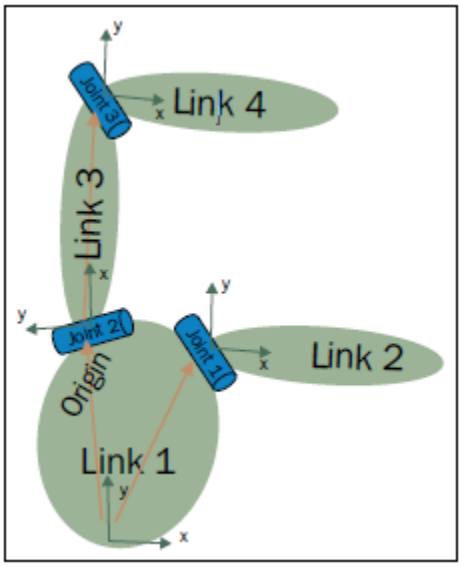

Terminal
3:09 PM
World
Insert
Layers
GUI
Scene
Spherical Coordinates
Physics
Models
bookshelf
jersey_barrier
ground_plane_0
unit_cylinder_1
Dumpster
cube_20k
mobile_base
turtlebot@CPR-TBT-1370: ~
FO] [1591177172.868796, 172.930000]: Robot X=-3.804564
FO] [1591177172.871620, 172.930000]: Robot X=-3.804823
FO] [1591177172.875107, 172.930000]: Robot X=-3.805081
FO] [1591177172.909913, 172.960000]: Robot X=-3.805330
FO] [1591177172.912846, 172.960000]: Robot X=-3.805576
FO] [1591177172.920197, 172.970000]: Robot X=-3.805831
FO] [1591177172.925652, 172.970000]: Robot X=-3.806085
FO] [1591177172.964429, 172.990000]: Robot X=-3.806329
FO] [1591177173.004941, 172.990000]: Robot X=-3.806572
FO] [1591177173.044302, 173.010000]: Robot X=-3.806824
FO] [1591177173.046809, 173.020000]: Robot X=-3.807075
Take Screenshot
Grab the whole screen
Grab the current window
Select area to grab
Grab after a delay of
seconds
Include pointer
Include the window border
Apply effect:
None
Take Screenshot
Real Time:
Iterations:
FPS:

6- Introduction to mobile Robots using URDF

EXERCISE 1

Aim: Build your own customized robot, consisting of a base, one front wheel and two rear wheels, with URDF.

ROS provides packages that can be used to build 3D robot models. It has a meta package called robot_model, which contains important packages to build 3D robot models. One of the important packages inside the robot_model meta package is urdf. The URDF package contains a C++ parser for the **Unified Robot Description Format** (**URDF**), which is an XML file, used to represent a robot model. A robot in URDF is described as a tree-like structure in its links, that is, the robot will have rigid links and will be connected using joints.



Visualization of a robot model having joints and links

Methods:

✓ **Installation:**

$sudo apt-get install python3.5

$sudo apt-get install ros-kinetic-urdf-tutorial

✓ **Skip this if Visual Studio is already installed**

- Download visual studio from https://code.visualstudio.com/Download
- Download deb package suitable to your laptop
- Install as

sudo dpkg -i code_1.50.01602051089_amd64.deb

- Open visual studio code
- Click on extension
- Type xml

xml complete->install

- Type python

python extension pack→install

- terminal-->install

✓ **Create package custom_robot in catkin workspace**

$cd catkin_ws

$cd src

$catkin_create_pkg custom_robot rospy roscpp std_msgs

$cd ..

$catkin_make

$exit

✓ Goto folder custom_robot and open the folder with visual studio code.

✓ In Visual studio, right click on the folder, click on create a new folder and name as ‘urdf’

✓ Create a new file in urdf and name as ‘custom_bot’.

✓ Type the following code in ‘custom_bot’ to create the customized robot.

```
<?xml version="1.0"?>
<robot name="custom_robot">
   <link name="base_link">
      <visual>
         <geometry>
            <box size="0.6 0.3 0.1"/>
```

```
      </geometry>
      <material name="red">
        <color rgba="1 0.0 0.0 1"/>
      </material>
    </visual>
    <collision>
      <geometry>
        <box size="0.6 0.3 0.1"/>
      </geometry>
    </collision>
    <inertial>
      <mass value="1.0"/>
      <inertia ixx="0.015" ixy="0" ixz="0" iyy="0.0375" iyz="0.0" izz="0.0375"/>
    </inertial>

</link>
<link name="front_caster_of_wheel">
    <visual>
      <geometry>
        <box size="0.1 0.1 0.1"/>
      </geometry>
      <material name="green">
        <color rgba="0.0 0.1 0.0 1"/>
      </material>
    </visual>
    <collision>
      <geometry>
        <box size="0.1 0.1 0.1"/>
      </geometry>
    </collision>
    <inertial>
```

```
        <mass value="0.1"/>
        <inertia ixx="0.00083" ixy="0" ixz="0" iyy="0.00083" iyz="0.0" izz="0.000167"/>
    </inertial>
</link>
<joint name="front_caster_of_wheel_joint" type="continuous">
   <axis xyz="0.0 0.0 1"/>
   <parent link="base_link"/>
   <child link="front_caster_of_wheel"/>
   <origin xyz="0.3 0.0 0.0" rpy="0.0 0.0 0.0"/>
</joint>

<link name="front_wheel">
    <visual>
        <geometry>
            <cylinder radius="0.035" length="0.05"/>
        </geometry>
        <material name="black">


        </material>

    </visual>
    <collision>
        <geometry>
            <cylinder radius="0.035" length="0.05"/>
        </geometry>
    </collision>
    <inertial>
        <mass value="0.1"/>
        <inertia ixx="5.1458e-5" ixy="0" ixz="0" iyy="5.1458e-5" iyz="0.0" izz="6.125e-5"/>
    </inertial>
</link>
```

```
    <joint name="front_wheel_joint" type="continuous">
        <axis xyz="0.0 0.0 1"/>
        <parent link="front_caster_of_wheel"/>
        <child link="front_wheel"/>
        <origin xyz="0.05 0.0 -0.05" rpy="-1.5708 0.0 0.0"/>

    </joint>

<link name="right_wheel">
        <visual>
            <geometry>
                <cylinder radius="0.035" length="0.05"/>
            </geometry>
            <material name="black">
                <color rgba="0.0 0.0 0.0 1"/>
            </material>

        </visual>
        <collision>
            <geometry>
                <cylinder radius="0.035" length="0.05"/>
            </geometry>
        </collision>
        <inertial>
            <mass value="0.1"/>
            <inertia ixx="5.1458e-5" ixy="0" ixz="0" iyy="5.1458e-5" iyz="0.0" izz="6.125e-5"/>
</inertial>
    </link>

    <joint name="right_wheel_joint" type="continuous">
```

```
      <axis xyz="0.0 0.0 1"/>
      <parent link="base_link"/>
      <child link="right_wheel"/>
      <origin xyz="-0.2825 -0.125 -0.05" rpy="-1.5708 0.0 0.0"/>

   </joint>
   <link name="left_wheel">
      <visual>
         <geometry>
            <cylinder radius="0.035" length="0.05"/>
         </geometry>
         <material name="black">
            <color rgba="0.0 0.0 0.0 1"/>
         </material>

      </visual>
      <collision>
         <geometry>
            <cylinder radius="0.035" length="0.05"/>
         </geometry>
      </collision>
      <inertial>
         <mass value="0.1"/>
         <inertia ixx="5.1458e-5" ixy="0" ixz="0" iyy="5.1458e-5" iyz="0.0" izz="6.125e-5"/>
</inertial>
   </link>

   <joint name="left_wheel_joint" type="continuous">
      <axis xyz="0.0 0.0 1"/>
      <parent link="base_link"/>
      <child link="left_wheel"/>
```

```
    <origin xyz="-0.2825 0.125 -0.05" rpy="-1.5708 0.0 0.0"/>

  </joint>

  <gazebo>
    <plugin name="differential_drive_controller"
            filename="libgazebo_ros_diff_drive.so">
      <leftJoint>left_wheel_joint</leftJoint>
      <rightJoint>right_wheel_joint</rightJoint>
      <legacyMode>false</legacyMode>

      <robotBaseFrame>base_link</robotBaseFrame>
      <wheelSeparation>0.25</wheelSeparation>
      <wheelDiameter>0.07</wheelDiameter>
      <publishWheelJointState>true</publishWheelJointState>
    </plugin>
  </gazebo>
   <gazebo>
    <plugin name="joint_state_publisher"
            filename="libgazebo_ros_joint_state_publisher.so">
      <jointName>front_caster_of_wheel_joint, front_wheel_joint</jointName>
    </plugin>
  </gazebo>

  </robot>
```

✓ **Create Rviz and gazebo Launch files**

Create new file in 'urdf': custom_bot_rviz.launch and write the following code:

```
<launch>
<!-- values passed by command line input -->
<arg name="model" />
```

```
<arg name="gui" default="False" />
<!-- set these parameters on Parameter Server -->
<param name="robot_description"
textfile="$(find custom_robot)/urdf/$(arg model)" />
<param name="use_gui" value="$(arg gui)"/>
<!-- Start 3 nodes: joint_state_publisher,
robot_state_publisher and rviz -->
<node name="joint_state_publisher"
pkg="joint_state_publisher"
type="joint_state_publisher" />
<node name="robot_state_publisher"
pkg="robot_state_publisher"
type="state_publisher" />
<node name="rviz" pkg="rviz" type="rviz"
args="-d $(find custom_robot)/urdf.rviz"
required="true" />
</launch>
```

Create new file in 'urdf': custom_bot_gazebo.launch and write the following code:

```
<launch>

<param name="robot_description" textfile="$(find custom_robot)/urdf/custom_bot.urdf" />
<include file="$(find gazebo_ros)/launch/empty_world.launch"/>

<!-- Spawn into Gazebo -->

<node name="spawn_urdf" pkg="gazebo_ros" type="spawn_model"
args="-param robot_description -urdf -model custom_bot.urdf" />

</launch>
```

✓ **Launch in gazebo**

$roslaunch custom_robot custom_bot_gazebo.launch

✓ **Install the following to move the custom robot through keyboard**

$sudo apt-get install ros-kinetic-teleop-twist-keyboard

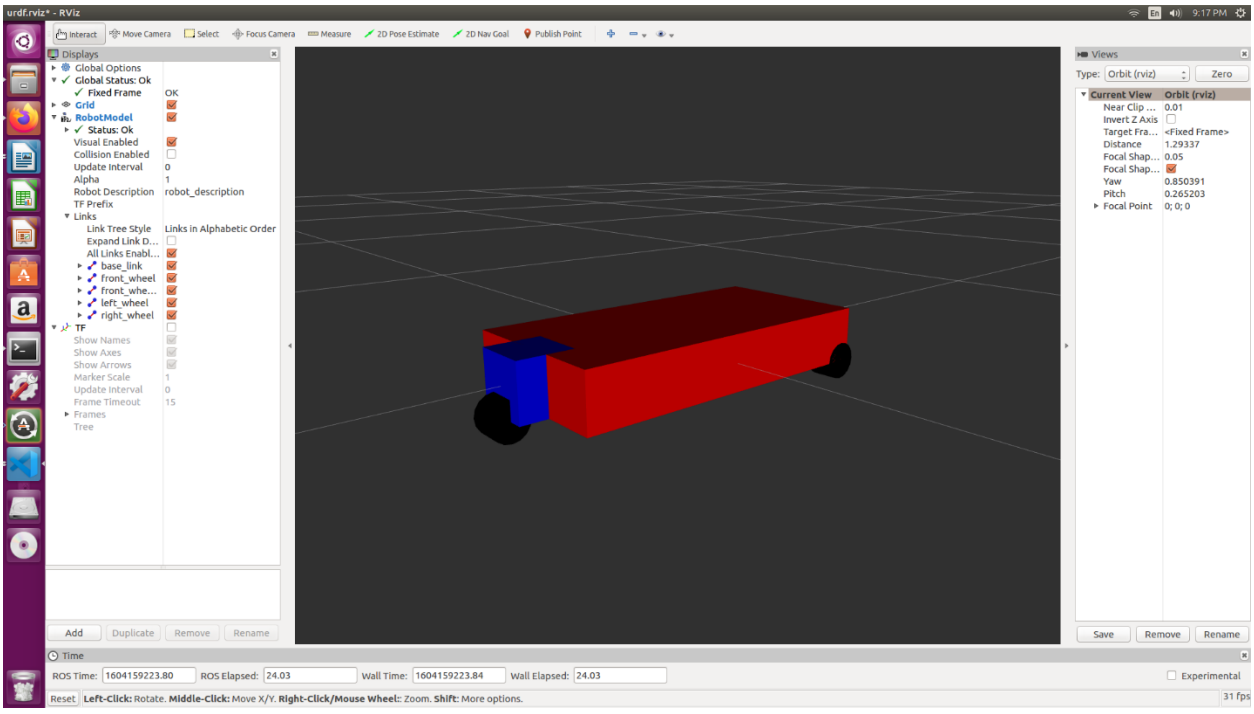
✓ **Move the robot through keyboard:**

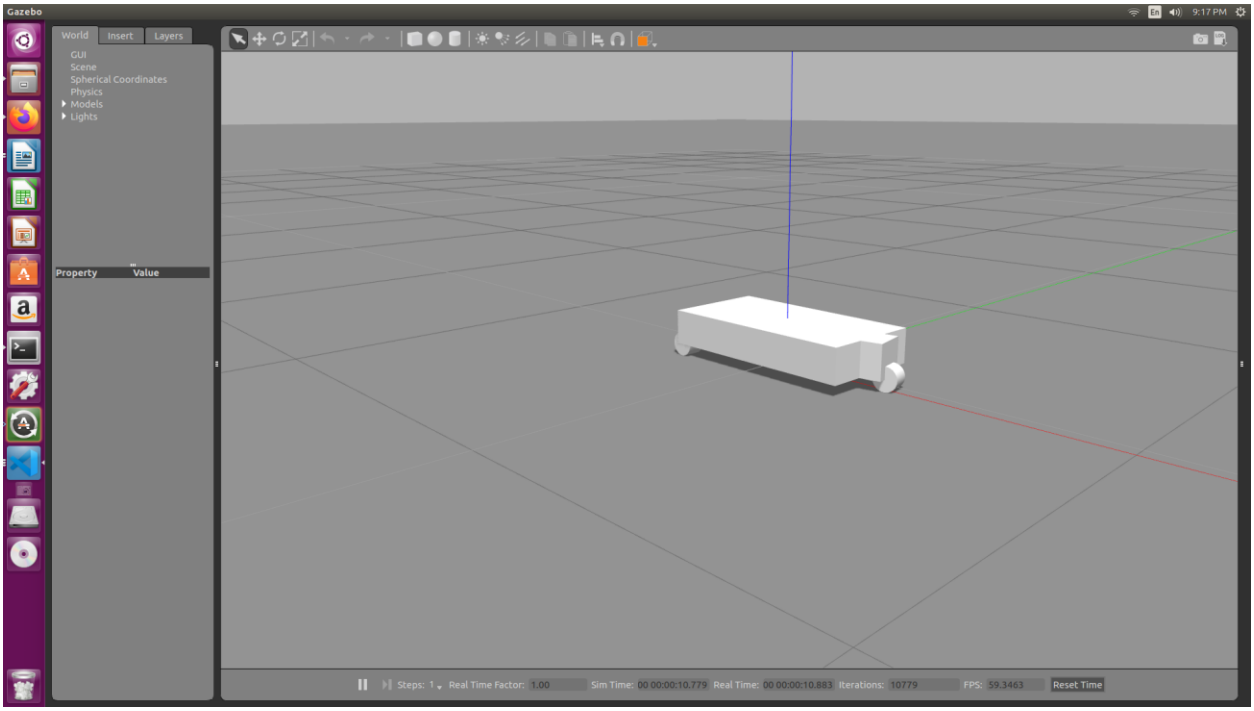
$roscore

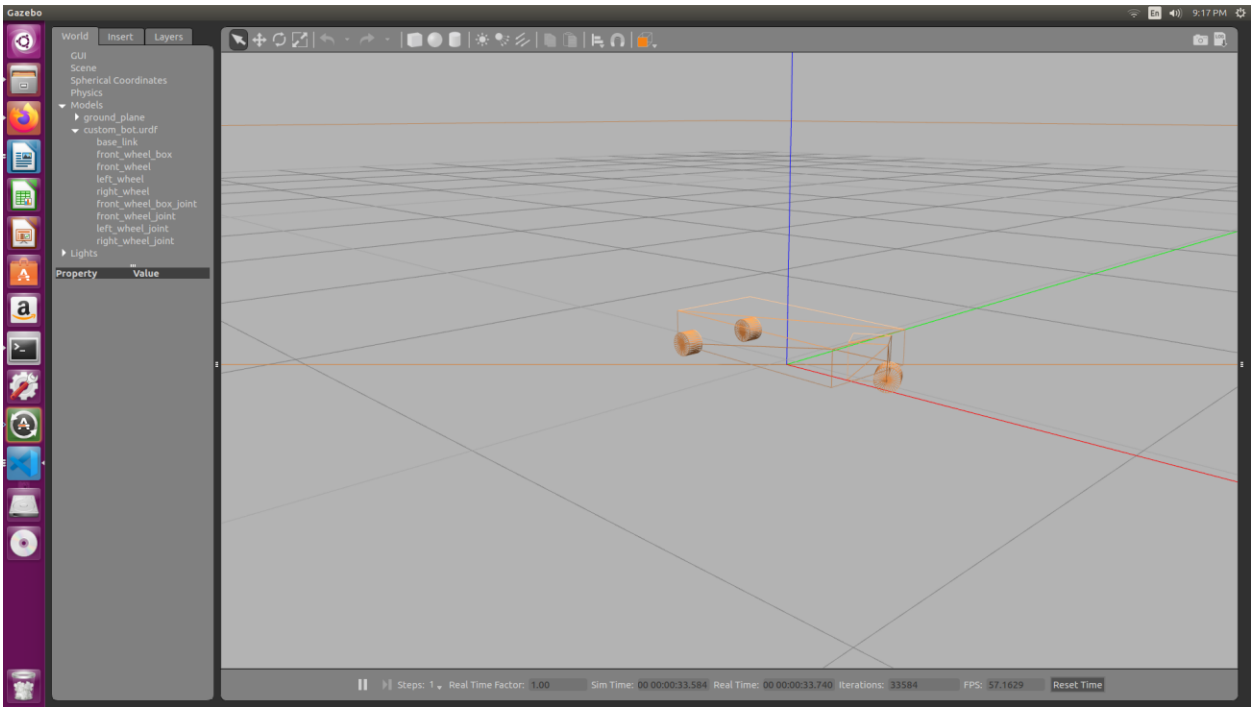
$roslaunch custom_robot custom_bot_rviz.launch model:=custom_bot.urdf

$roslaunch custom_robot custom_bot_gazebo.launch

$rosrun teleop_twist_keyboard teleop_twist_keyboard.py




Gazebo
9:17 PM
World
Insert
Layers
GUI
Scene
Spherical Coordinates
Physics
Models
Lights
Property
Value
Steps: 1
Real Time Factor: 1.00
Sim Time: 00 00:00:10.779
Real Time: 00 00:00:10.883
Iterations: 10779
FPS: 59.3463
Reset Time



2) EXERCISE 2

AIM: ROS simulation experiment on SLAM using Gazebo and Rviz

What is the SLAM problem?

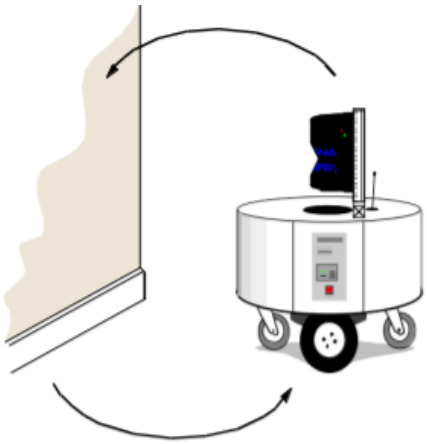
The problem could describe in the following context:

If we leave a robot in an unknown location in an unknown environment can the robot make a satisfactory map while simultaneously being able to find its pose in that map?

The solution to this problem was very significant for the field of mobile robotics.

SLAM is the process by which a robot builds a map of the environment and, at the same time, uses this map to compute its location.

- Localization: inferring location given a map
- Mapping: inferring a map given a location
- SLAM: learning a map and locating the robot simultaneously

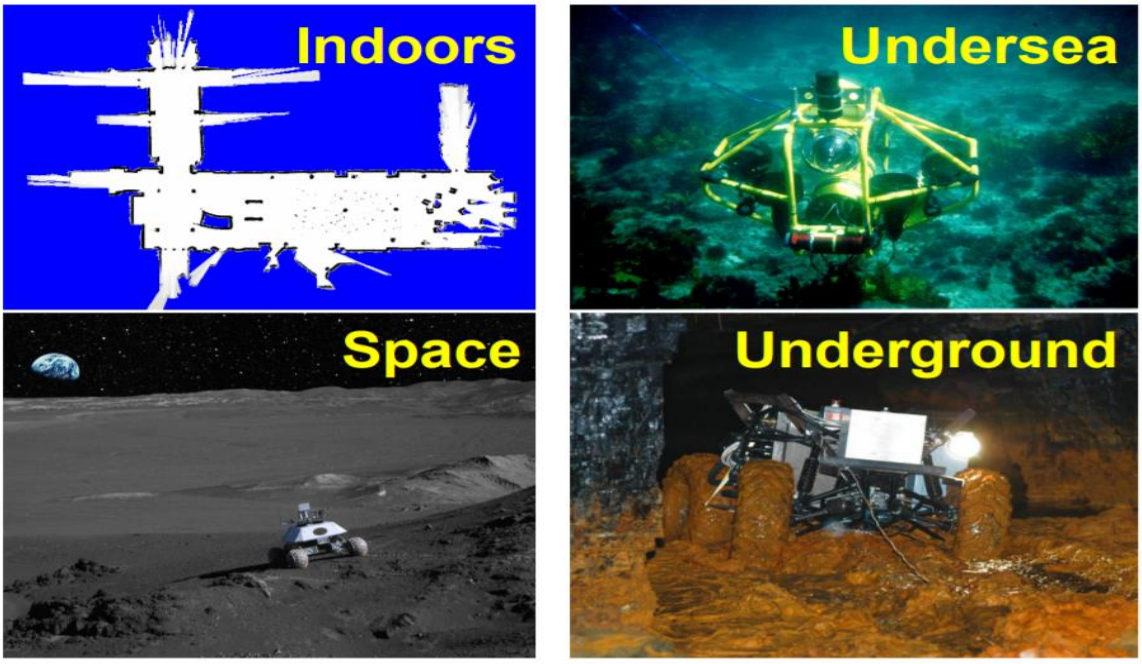


SLAM is a chicken-or-egg problem:

- A map is needed for localizing a robot
- A pose estimate is needed to build a map

SLAM is (regarded as) a hard problem in robotics. Robot path and map are both unknown

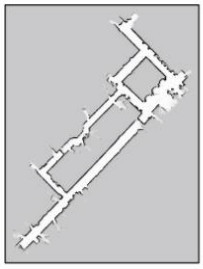
SLAM Applications:



Representations:

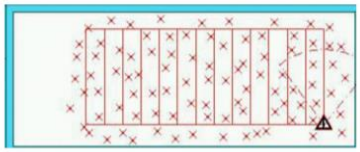
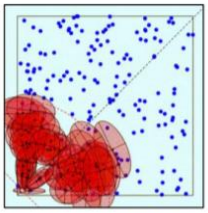
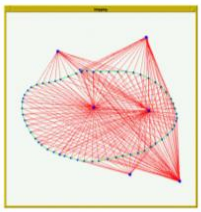
- Grid maps or scans





[Lu & Milios, 97; Gutmann, 98: Thrun 98; Burgard, 99; Konolige & Gutmann, 00; Thrun, 00; Arras, 99; Haehnel, 01;…]

- Landmark-based

Modify the urdf xml code to integrate laser sensors:

```
<?xml version="1.0"?>
<robot name="custom_robot">
  <link name="base_link">
    <visual>
      <geometry>
        <box size="0.6 0.3 0.1"/>
      </geometry>
      <material name="red">
        <color rgba="1 0.0 0.0 1"/>
      </material>
    </visual>
    <collision>
      <geometry>
        <box size="0.6 0.3 0.1"/>
```

```
      </geometry>
    </collision>
    <inertial>
      <mass value="1.0"/>
      <inertia ixx="0.015" ixy="0" ixz="0" iyy="0.0375" iyz="0.0" izz="0.0375"/>
    </inertial>

</link>
<link name="front_caster_of_wheel">
    <visual>
      <geometry>
        <box size="0.1 0.1 0.1"/>
      </geometry>
      <material name="green">
        <color rgba="0.0 0.1 0.0 1"/>
      </material>
    </visual>
    <collision>
      <geometry>
        <box size="0.1 0.1 0.1"/>
      </geometry>
    </collision>
    <inertial>
      <mass value="0.1"/>
      <inertia ixx="0.00083" ixy="0" ixz="0" iyy="0.00083" iyz="0.0" izz="0.000167"/>
    </inertial>
</link>
<joint name="front_caster_of_wheel_joint" type="continuous">
   <axis xyz="0.0 0.0 1"/>
   <parent link="base_link"/>
   <child link="front_caster_of_wheel"/>
```

```
  <origin xyz="0.3 0.0 0.0" rpy="0.0 0.0 0.0"/>
</joint>

<link name="front_wheel">
    <visual>
        <geometry>
            <cylinder radius="0.035" length="0.05"/>
        </geometry>
        <material name="black">

        </material>

    </visual>
    <collision>
        <geometry>
            <cylinder radius="0.035" length="0.05"/>
        </geometry>
    </collision>
    <inertial>
        <mass value="0.1"/>
        <inertia ixx="5.1458e-5" ixy="0" ixz="0" iyy="5.1458e-5" iyz="0.0" izz="6.125e-5"/>
    </inertial>
</link>

<joint name="front_wheel_joint" type="continuous">
    <axis xyz="0.0 0.0 1"/>
    <parent link="front_caster_of_wheel"/>
    <child link="front_wheel"/>
    <origin xyz="0.05 0.0 -0.05" rpy="-1.5708 0.0 0.0"/>

</joint>
```

```
<link name="right_wheel">
        <visual>
            <geometry>
                <cylinder radius="0.035" length="0.05"/>
            </geometry>
            <material name="black">
                <color rgba="0.0 0.0 0.0 1"/>
            </material>

        </visual>
        <collision>
            <geometry>
                <cylinder radius="0.035" length="0.05"/>
            </geometry>
        </collision>
        <inertial>
            <mass value="0.1"/>
            <inertia ixx="5.1458e-5" ixy="0" ixz="0" iyy="5.1458e-5" iyz="0.0" izz="6.125e-5"/>
</inertial>
    </link>

    <joint name="right_wheel_joint" type="continuous">
        <axis xyz="0.0 0.0 1"/>
        <parent link="base_link"/>
        <child link="right_wheel"/>
        <origin xyz="-0.2825 -0.125 -0.05" rpy="-1.5708 0.0 0.0"/>

    </joint>
    <link name="left_wheel">
        <visual>
```

```
            <geometry>
                <cylinder radius="0.035" length="0.05"/>
            </geometry>
            <material name="black">
                <color rgba="0.0 0.0 0.0 1"/>
            </material>

        </visual>
        <collision>
            <geometry>
                <cylinder radius="0.035" length="0.05"/>
            </geometry>
        </collision>
        <inertial>
            <mass value="0.1"/>
            <inertia ixx="5.1458e-5" ixy="0" ixz="0" iyy="5.1458e-5" iyz="0.0" izz="6.125e-5"/>
</inertial>
    </link>

    <joint name="left_wheel_joint" type="continuous">
        <axis xyz="0.0 0.0 1"/>
        <parent link="base_link"/>
        <child link="left_wheel"/>
        <origin xyz="-0.2825 0.125 -0.05" rpy="-1.5708 0.0 0.0"/>

    </joint>

    <gazebo>
        <plugin name="differential_drive_controller"
                filename="libgazebo_ros_diff_drive.so">
            <leftJoint>left_wheel_joint</leftJoint>
```

```
        <rightJoint>right_wheel_joint</rightJoint>
        <legacyMode>false</legacyMode>

        <robotBaseFrame>base_link</robotBaseFrame>
        <wheelSeparation>0.25</wheelSeparation>
        <wheelDiameter>0.07</wheelDiameter>
        <publishWheelJointState>true</publishWheelJointState>
    </plugin>
</gazebo>
  <gazebo>
    <plugin name="joint_state_publisher"
          filename="libgazebo_ros_joint_state_publisher.so">
        <jointName>front_caster_of_wheel_joint, front_wheel_joint</jointName>
    </plugin>
</gazebo>
<link name="laser_scanner">
    <visual>
        <geometry>
            <box size="0.1 0.1 0.1"/>
        </geometry>
    </visual>

    <collision>
        <geometry>
            <box size="0.1 0.1 0.1"/>
        </geometry>
    </collision>

    <inertial>
        <mass value="1e-5"/>
        <inertia ixx="1e-6" ixy="0" ixz="0.0" iyy="1e-6" iyz="0.0" izz="1e-6"/>
```

```
    </inertial>
</link>

<joint name="laser_scanner_joint" type="fixed">
    <axis xyz="0.0 1 0.0"/>
    <parent link="base_link"/>
    <child link="laser_scanner"/>
    <origin xyz="0.0 0.0 0.08" rpy="0.0 0.0 0.0"/>
</joint>

<gazebo reference="laser_scanner">
    <sensor type="ray" name="laser">
        <pose>0 0 0 0 0 0</pose>
        <visualize>true</visualize>
        <update_rate>40</update_rate>
        <ray>
            <scan>
                <horizontal>
                    <samples>720</samples>
                    <resolution>1</resolution>
                    <min_angle>-1.578</min_angle>
                    <max_angle>1.578</max_angle>
                </horizontal>
            </scan>
            <range>
                <min>0.1</min>
                <max>30</max>
                <resolution>0.1</resolution>
            </range>
        </ray>
```

```
        <plugin                                    name="gazebo_ros_head_hokuyo_controller"
filename="libgazebo_ros_laser.so">
            <topicName>/scan</topicName>
            <frameName>laser_scanner</frameName>
        </plugin>
      </sensor>
   </gazebo>

   </robot>
```

✓ **Create bot_model.launch file inside the urdf folder**

```
<launch>
   <param name="robot_description" textfile="$(find custom_robot)/urdf/custom_bot.urdf" />
   <include file="$(find gazebo_ros)/launch/empty_world.launch">

   </include>
   <node name="robot_state_publisher" pkg="robot_state_publisher"
type="robot_state_publisher" />

   <node name="spawn_urdf" pkg="gazebo_ros" type="spawn_model"
      args="-param robot_description -urdf -model custom_bot" />
   1
   <node pkg="teleop_twist_keyboard" type="teleop_twist_keyboard.py"
name="teleop_twist_keyboard" output="screen"/>
   <node pkg="rviz" type="rviz" name="rviz"/>

   <node pkg="gmapping" type="slam_gmapping" name="gmapping">
     <param name="base_frame" value="base_link" />
     <param name="odom_frame" value="odom" />
     <param name="delta" value="0.1" />
   </node>
```

```
    <node pkg="move_base" type="move_base" name="move_base" output="screen">
      <param name="controller_frequency" value="10.0" />
      <rosparam file="$(find custom_robot)/config/costmap_common_params.yaml"
command="load" ns="global_costmap"/>
      <rosparam file="$(find custom_robot)/config/costmap_common_params.yaml"
command="load" ns="local_costmap"/>
      <rosparam file="$(find custom_robot)/config/local_costmap_params.yaml"
command="load" />
      <rosparam file="$(find custom_robot)/config/global_costmap_params.yaml"
command="load" />
      <rosparam file="$(find custom_robot)/config/trajectory_planner.yaml" command="load" />
    </node>


</launch>
```

1. Open the terminal

$roscore

2. Open another terminal

$ roslaunch custom_robot bot_model.launch

Opens Gazebo, Rviz Window.

3. In Gazebo

- edit-->building editor

Use the tools to build the house or environment

- File-->save as

- File--> exit

4. In Rviz Do the following modifications

- Create a folder named config

- **Define all parameters**

create filename: costmap_common_params.yaml

obstacle_range: 6.0
raytrace_range: 8.5
footprint: [[0.3,0.14],[0.3,-0.14],[-0.3,-0.14],[-0.3,0.14]]
map_topic: /map
subscribe_to_update: true
observation_sources: laser_scan_sensor
laser_scan_sensor: {sensor_frame: laser_frame,data_type: LaserScan, topic: scan,marking: true , clearing: true }
global_frame: map
robot_base_frame: base_link
always_send_full_costmap: true

create filename: global_costmap_params.yaml

global_costmap:
  update_frequency: 2.5
  publish_frequency: 2.5
  transform_tolerance: 0.5
  static_map: true
  rolling_window: true
  width: 15

```
  height: 15
  origin_x: -7.5
  origin_y: -7.5
  resolution: 0.1
  inflation_radius: 2.5
```

create filename: local_costmap_params.yaml

```
local_costmap:
  update_frequency: 5
  publish_frequency: 5
  transform_tolerance: 0.25
  static_map: false
  rolling_window: true
  width: 3
  height: 3
  origin_x: -1.5
  origin_y: -1.5
  resolution: 0.1
  inflation_radius: 0.6
```

create filename: trajectory_planner.yaml

```
TrajectoryPlannerROS:
  max_vel_x: 0.2
  min_vel_x: 0.1
  max_vel_theta: 0.35
  min_vel_theta: -0.35
  min_in_place_vel_theta: 0.25
```

acc_lim_theta: 0.25
acc_lim_X: 2.5
acc_lim_Y: 2.5

holonomic_robot: false
meter_scoring: true

xy_goal_tolerance: 0.15
yaw_goal_tolerance: 0.25

- **Click on Global Options**

-->Add-->RobotModel
-->Add-->TF
-->Add-->Laser Scanner-->topic=/scan
-->Add-->Path-->select By topic-->/global path-->Path
-->Add-->Path-->select By topic-->/local-->Path
-->Add-->Path-->select By topic-->/footprint-->Polygon
-->Add-->Map--> topic=/map

- **Change the fixed frame: odom**

Select terminal with use i, l, j, m, k for moving the robot inside the house to get the map.

Use 2D Nav Goal for automation navigation.

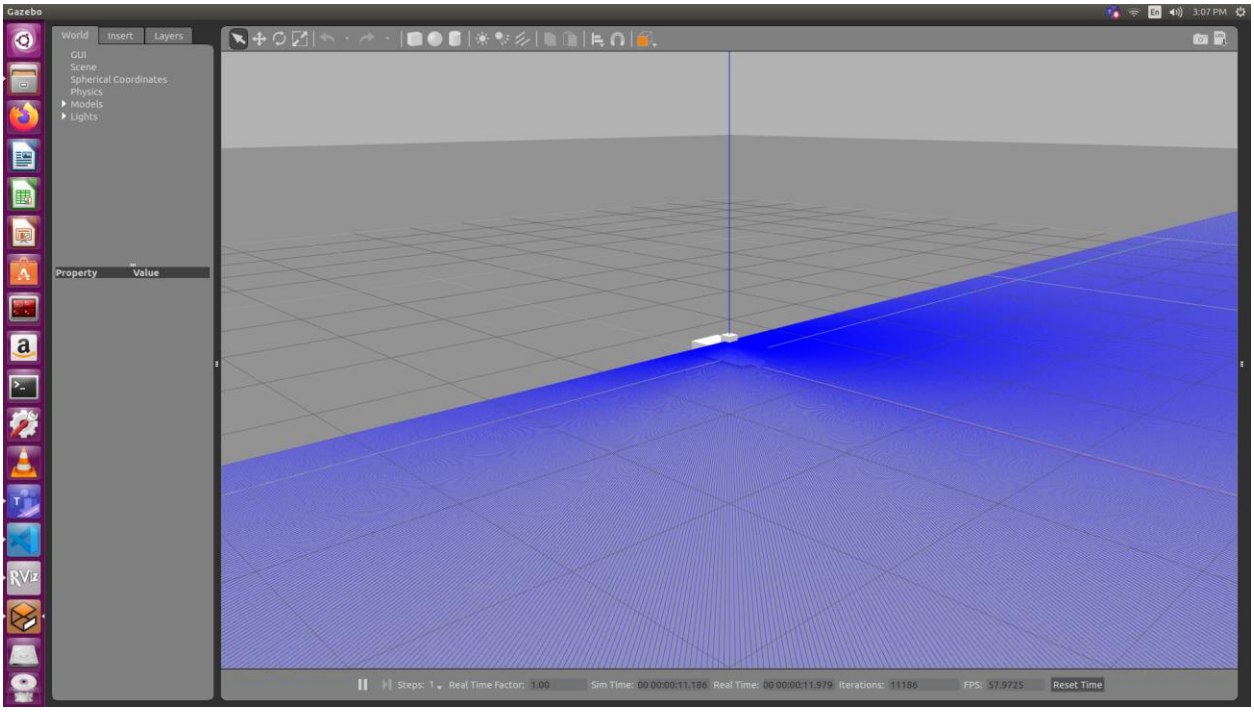
You can keep the object in the middle, and check the movement.

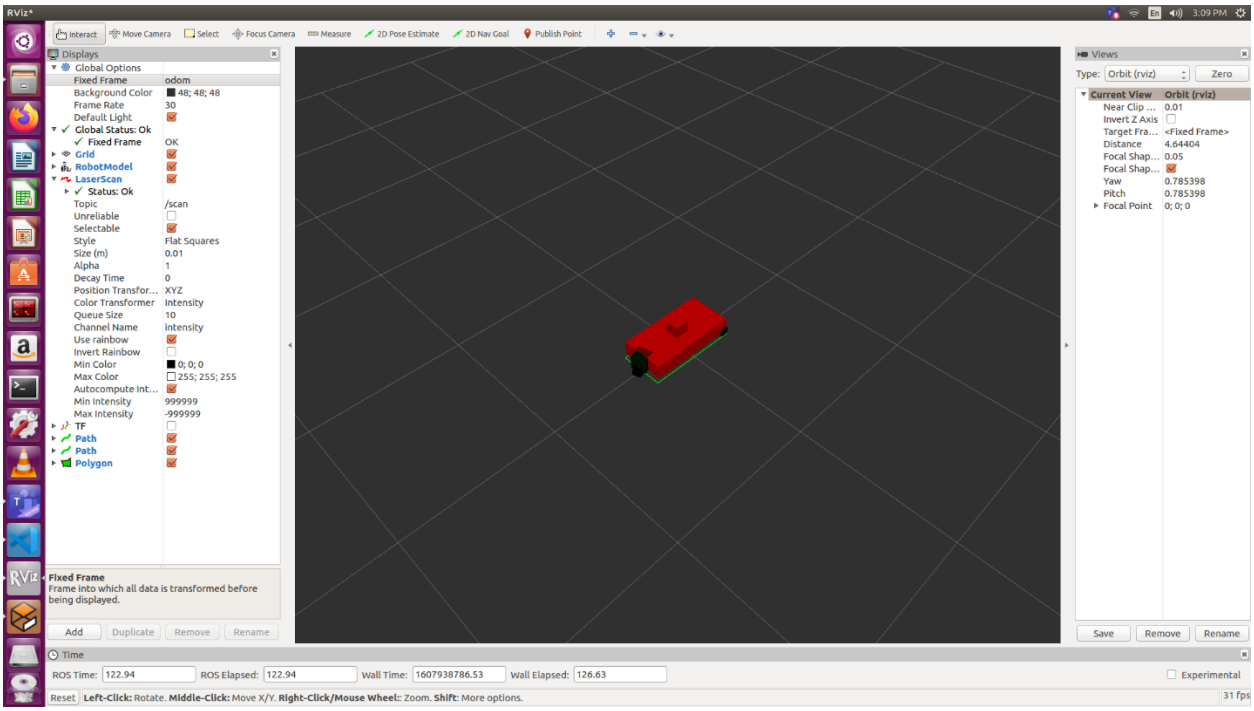
Post Lab exercises

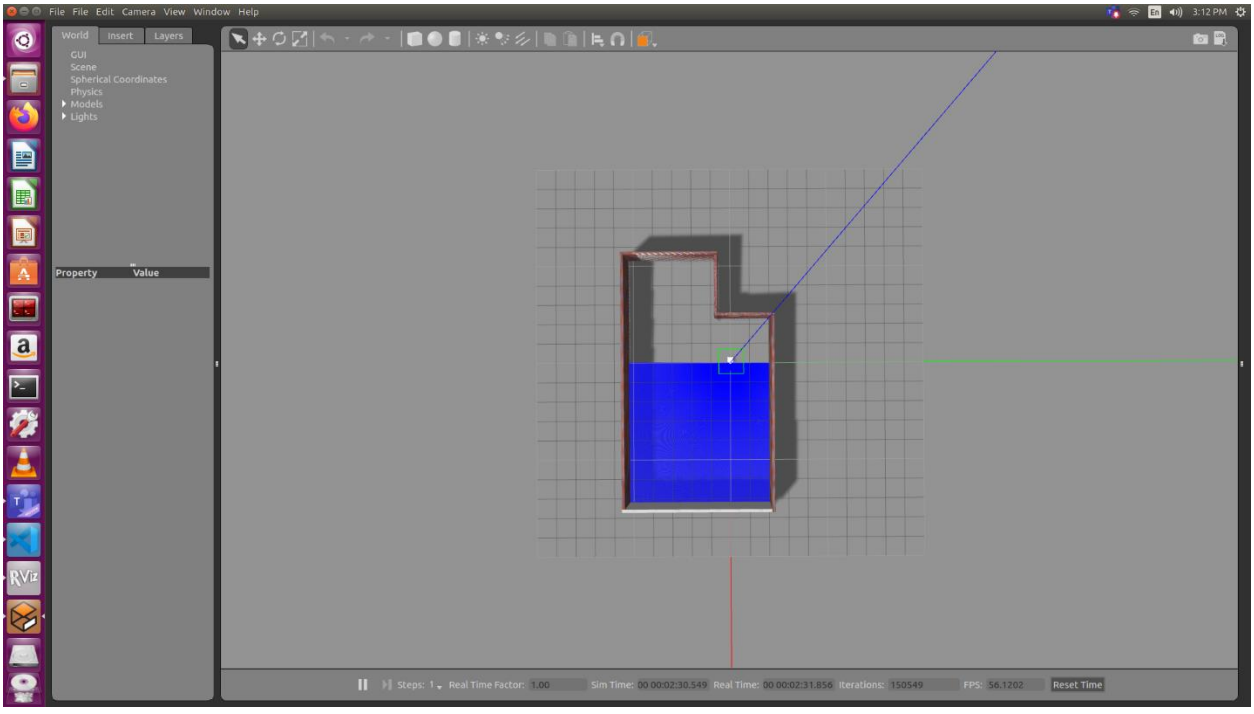
1. Create a 2 wheeled robot with caster wheel.

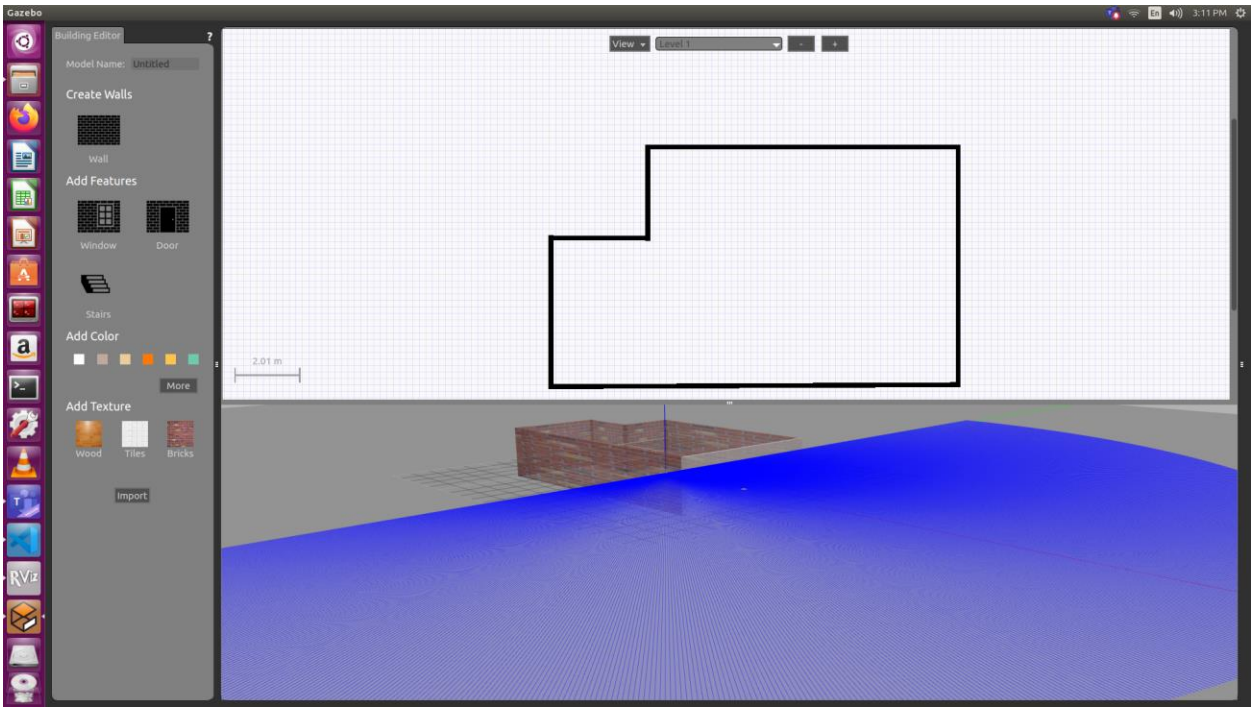
2. Create a 3 joint manipulator with gripper as end tool.

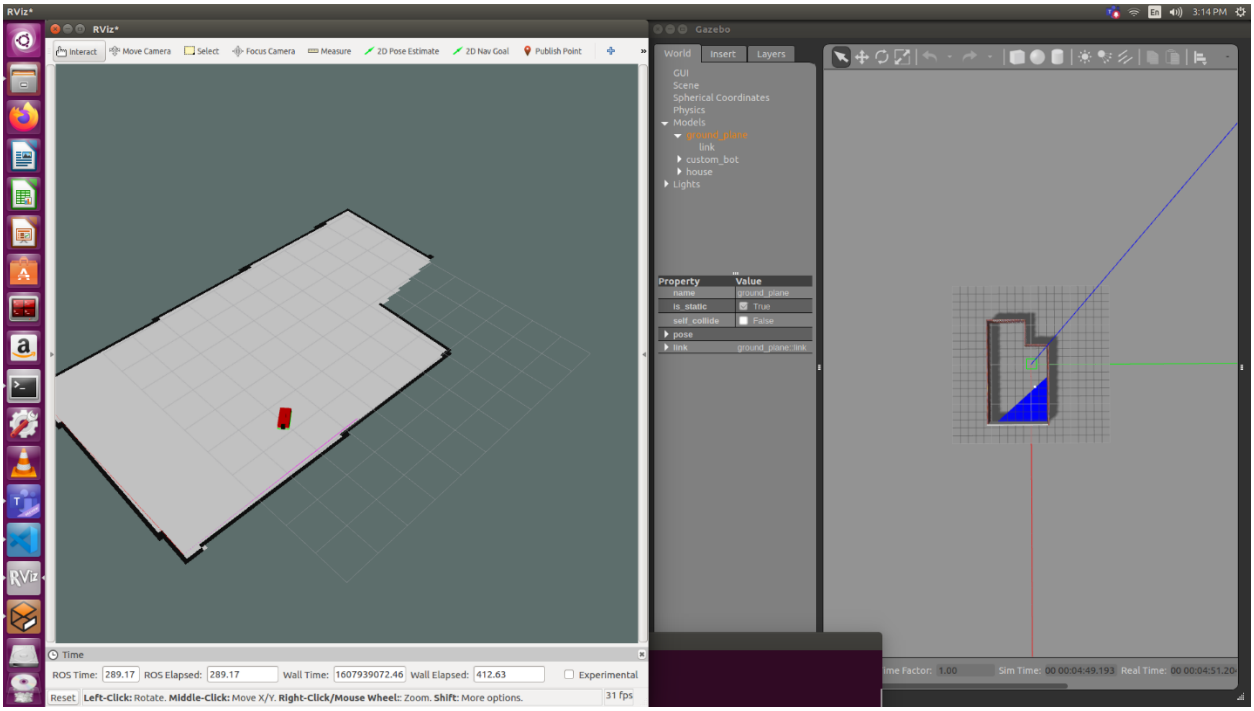
Reference: ROS Robotics by Example : Carol Fairchild

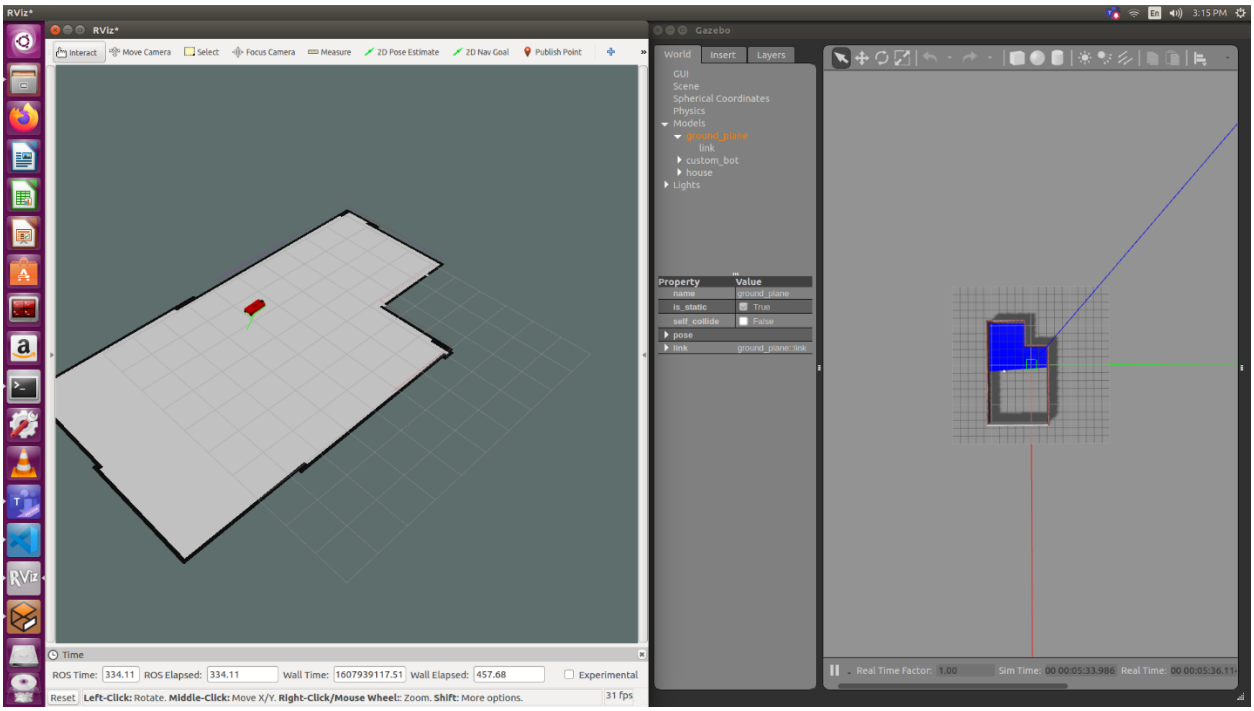



RViz*
3:09 PM
Interact
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
2D Nav Goal
Publish Point
Displays
Global Options
Fixed Frame odom
Background Color 48; 48; 48
Frame Rate 30
Default Light
Global Status: Ok
Fixed Frame OK
Grid
RobotModel
LaserScan
Status: Ok
Topic /scan
Unreliable
Selectable
Style Flat Squares
Size (m) 0.01
Alpha 1
Decay Time 0
Position Transfor... XYZ
Color Transformer Intensity
Queue Size 10
Channel Name intensity
Use rainbow
Invert Rainbow
Min Color 0; 0; 0
Max Color 255; 255; 255
Autocompute Int...
Min Intensity 999999
Max Intensity -999999
TF
Path
Path
Polygon
Fixed Frame
Frame into which all data is transformed before being displayed.
Add
Duplicate
Remove
Rename
Views
Type: Orbit (rviz)
Zero
Current View Orbit (rviz)
Near Clip ... 0.01
Invert Z Axis
Target Fra... <Fixed Frame>
Distance 4.64404
Focal Shap... 0.05
Focal Shap...
Yaw 0.785398
Pitch 0.785398
Focal Point 0; 0; 0
Save
Remove
Rename
Time
ROS Time: 122.94
ROS Elapsed: 122.94
Wall Time: 1607938786.53
Wall Elapsed: 126.63
Experimental
Reset
Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel:: Zoom. Shift: More options.
31 fps


File File Edit Camera View Window Help
3:12 PM
World
Insert
Layers
GUI
Scene
Spherical Coordinates
Physics
Models
Lights
Property
Value
Steps: 1
Real Time Factor: 1.00
Sim Time: 00 00:02:30.549
Real Time: 00 00:02:31.856
Iterations: 150549
FPS: 56.1202
Reset Time


Gazebo
Building Editor
Model Name: Untitled
Create Walls
Wall
Add Features
Window
Door
Stairs
Add Color
More
Add Texture
Wood
Tiles
Bricks
Import
View
Level 1
2.01 m


RViz*
Interact
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
2D Nav Goal
Publish Point
Time
ROS Time: 289.17
ROS Elapsed: 289.17
Wall Time: 1607939072.46
Wall Elapsed: 412.63
Experimental
Reset
Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel: Zoom. Shift: More options.
31 fps
Gazebo
World
Insert
Layers
GUI
Scene
Spherical Coordinates
Physics
Models
ground_plane
link
custom_bot
house
Lights
Property
Value
name
ground_plane
is_static
True
self_collide
False
pose
link
ground_plane::link
Sim Time: 00 00:04:49.193
Real Time: 00 00:04:51.20


RViz*
Interact
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
2D Nav Goal
Publish Point
Time
ROS Time: 334.11
ROS Elapsed: 334.11
Wall Time: 1607939117.51
Wall Elapsed: 457.68
Experimental
Reset
Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel: Zoom. Shift: More options.
31 fps
Gazebo
World
Insert
Layers
GUI
Scene
Spherical Coordinates
Physics
Models
ground_plane
link
custom_bot
house
Lights
Property
Value
name
ground_plane
is_static
True
self_collide
False
pose
link
ground_plane::link
Real Time Factor: 1.00
Sim Time: 00 00:05:33.986
Real Time: 00 00:05:36.11
3:15 PM

8 – INTRODUCTION TO MOBILE ROBOTS

Aim:

To familiarize with and maneuver a Turtle bot according to the applications provided by the programmer.

Theory:

Hardware Specifications:

- Kobuki Base
- Asus Xion Pro Live
- Netbook (ROS Compatible)
- Kinect Mounting Hardware
- TurtleBot Structure
- TurtleBot Module Plate with 1 inch Spacing Hole Pattern

Software Specifications:

- An SDK for the TurtleBot
- A development environment for the desktop
- Libraries for visualization, planning, and perception, control and error handling.

Figure: Turtlebot 2

Steps of Configuration:

1. Connect tutlebot to the laptop.
2. Connect Turtlebot laptop to WiFi.

3. Find out the IP address of the laptop by using the code in terminal window "ifconfig".

4. Close the laptop and place it under the safety sheild provided in turtlebot.

5. Get back to your system connected to local LAN and open terminal window.

6. Connect the PC to turtlebot through the IP address. "ssh turtlebot@[IP]".

7. The Command will prompt for password, type "turtlebot" as password.

8. The system now has provision to program or operate the turtlebot through terminal window.

9. Under turtlebot@CPR-TBT-****:~$, type the following code " roslaunch turtlebot_bringup minimal.launch" and press enter.

10. The following statement will initialize turtlebot with a connecting sound from bot.

11. Now open new terminal window and reconnect that window to turtlebot by using the step no:6.

12. Now as roslaunch a library for turtlebot setup is already running in the contrast we need to check the communication.

13. To try that lets bring a manually operating code.

14. type "roslaunch turtlebot_teleop keyboard_teleop.launch" in the next line in terminal window.

15. Now you are free to communicate with Turtlebot by following simple safety precautions.

16. Pressing CTRL+C can break the condition of teleoperate and brings back the command promt window.

17. This operation can be done to any remote controlling device with their own library calling functions.

Connect camera USB and Kubuki USB and switch on Kubuki.

Switch on the netbook.

Use Remmina to connect to turtlebot from Ubuntu/mtputty from windows

1. See that the turtlebot laptop and remote laptop are connected to same WiFi

2. Find the address of netbook placed on turtlebot
    a. Open terminal and type
    b. $ifconfig
    c. Copy the address inet addr: ----(For example **172.16.65.109**)
3. Open Remmina
4. Establish SSH connection
5. Type Name (for example **turtlebot**)
6. Server Name: **172.16.65.109**
   User name: turtlebot
   Password:turtlebot

1. Open the terminal and type

$roslaunch turtlebot_bringup minimal.launch

2. Open another terminal and type

$roslaunch turtlebot_teleop keyboard_teleop.launch

3. For Robot visualization
Open another terminal and type
$rviz

$cd catkin_ws/src
$catkin_create_pkg turtlebot2 roscpp rospy std_msgs
Make a folder scripts inside the package hello_world/src
$cd catkin_ws
$catkin_make
$cd turtlebot2/src
$mkdir scripts
$cd scripts

Use any editor/Visual Studio to create python file

$gedit turtlebot_square.py

$chmod +x turtlebot_square.py

Example 1:

```
#!/usr/bin/env python
import rospy
from geometry_msgs.msg import Twist
from nav_msgs.msg import Odometry
import sys
robot_x = 0

def pose_callback(msg):
        global robot_x
        #Reading x position from the Odometry message
        robot_x = msg.pose.pose.position.x
        rospy.loginfo("Robot X = %f\n",robot_x)
def move_turtle(lin_vel,ang_vel,distance):
    global robot_x
    rospy.init_node('move_turtlebot', anonymous=False)
    pub = rospy.Publisher('/mobile_base/commands/velocity', Twist, queue_size=10)
    rospy.Subscriber('/odom',Odometry, pose_callback)
    rate = rospy.Rate(10) # 10hz
    vel = Twist()
    while not rospy.is_shutdown():
        vel.linear.x = lin_vel
        vel.linear.y = 0
        vel.linear.z = 0
        vel.angular.x = 0
```

```
            vel.angular.y = 0
            vel.angular.z = ang_vel

        #rospy.loginfo("Linear Vel = %f: Angular Vel = %f",lin_vel,ang_vel)

            if(robot_x >= distance):
                    rospy.loginfo("Robot Reached destination")
                    rospy.logwarn("Stopping robot")
                    break

        pub.publish(vel)
        rate.sleep()

if __name__ == '__main__':
    try:
        move_turtle(float(sys.argv[1]),float(sys.argv[2]),float(sys.argv[3]))
    except rospy.ROSInterruptException:
        pass
```

Example 2:

```
#!/usr/bin/env python
import rospy
import time
from geometry_msgs.msg import Twist
import sys
def move_turtle(lin_vel,ang_vel):
    rospy.init_node('move_turtle', anonymous=True)
    pub = rospy.Publisher('/mobile_base/commands/velocity', Twist, queue_size=10)
    rate = rospy.Rate(1) # 10hz
    vel = Twist()
```

```
    while not rospy.is_shutdown():
        vel.linear.x = .5
        vel.linear.y = 0
        vel.linear.z = 0
        vel.angular.x = 0
        vel.angular.y = 0
        vel.angular.z =0
        rospy.loginfo("Linear Vel = %f: Angular Vel = %f",vel.linear.x,vel.angular.z)
        pub.publish(vel)
        time.sleep(2)
        vel.linear.x = 0
        vel.linear.y = 0
        vel.linear.z = 0
        vel.angular.x = 0
        vel.angular.y = 0
        vel.angular.z = 3
        rospy.loginfo("Linear Vel = %f: Angular Vel = %f",vel.linear.x,vel.angular.z)
        pub.publish(vel)
        time.sleep(1)
        rate.sleep()

if __name__ == '__main__':
    try:
        move_turtle(float(sys.argv[1]),float(sys.argv[2]))
    except rospy.ROSInterruptException:
        pass
```

Example 3:

```
#!/usr/bin/env python
import rospy
from geometry_msgs.msg import Twist
from turtlesim.msg import Pose
from nav_msgs.msg import Odometry
import math
import time
from std_srvs.srv import Empty


def move(speed, time, is_forward):
        t0=rospy.Time.now().to_sec()
        velocity_message=Twist()
        if(is_forward):
                velocity_message.linear.x=abs(speed)
        else:
                velocity_message.linear.x=-abs(speed)

        velocity_message.linear.y=0
        velocity_message.linear.z=0
        velocity_message.angular.x=0
        velocity_message.angular.y=0
        velocity_message.angular.z=0
        loop_rate=rospy.Rate(10)
        cmd_vel_topic='/cmd_vel_mux/input/navi'

        velocity_publisher=rospy.Publisher(cmd_vel_topic, Twist, queue_size=10)
        while(True):
                rospy.loginfo("Turtlesim moves forward")
                velocity_publisher.publish(velocity_message)
```

```
                loop_rate.sleep()
                t1=rospy.Time.now().to_sec()
                if (t1-t0)>10:
                        rospy.loginfo("time crossed")
                        rospy.logwarn("Stopping the Robot")
                        break
        velocity_message.linear.x=0
        velocity_publisher.publish(velocity_message)

def stop():
        velocity_message=Twist()
        velocity_message.linear.x=0
        velocity_publisher.publish(velocity_message)


def rotate(angular_speed_degree, relative_angle_degree,clockwise):
        velocity_message=Twist()
        velocity_message.linear.x=0
        velocity_message.linear.y=0
        velocity_message.linear.z=0
        velocity_message.angular.x=0
        velocity_message.angular.y=0
        velocity_message.angular.z=0
        angular_speed=math.radians(abs(angular_speed_degree))
        if(clockwise):
                velocity_message.angular.z=-abs(angular_speed)
        else:
                velocity_message.angular.z=abs(angular_speed)
        angle_moved=0.0
        loop_rate=rospy.Rate(10)
        and_vel_topic='/cmd_vel_mux/input/navi'
```

```
        velocity_publisher=rospy.Publisher(cmd_vel_topic,Twist,queue_size=10)
        t0=rospy.Time.now().to_sec()
        while(True):
                rospy.loginfo("Turtlesim rotates")
                velocity_publisher.publish(velocity_message)
                t1=rospy.Time.now().to_sec()
                current_angle_degree=(t1-t0)*angular_speed_degree
                loop_rate.sleep()
                if (current_angle_degree>relative_angle_degree):
                        rospy.loginfo("reached")
                        break
        velocity_message.angular.z=0
        velocity_publisher.publish(velocity_message)



if __name__ == '__main__':
        try:
                rospy.init_node('turtlesim_motion_pose',anonymous=True)
                cmd_vel_topic='/cmd_vel_mux/input/navi'
                velocity_publisher = rospy.Publisher(cmd_vel_topic, Twist, queue_size=10)
                for i in range(0,4):

                        move(0.05,2,True)
                        time.sleep(1)
                        rotate(30,125,False)
                stop()

        except rospy.ROSInterruptException:
                rospy.loginfo("node terminated")
```

1. Open the terminal and type
   $roscore
2. Open another terminal and type
   $roslaunch turtlebot_bringup minimal.launch
3. Open another terminal and type
   $rosrun turtlebot turtlebot_square.py
4. Open another terminal and type
   $rostopic list

Exercise:

Maneuver the Turtle bot according to the specifications given as instructions.

- Add LIDAR to the TURTLEBOT
- Implement SLAM and PATH PLANING using TURTLEBOT

PILIDAR Installation:

cd catkin_ws/src

sudo git clone https://github.cim/Slamtec/rplidar_ros.git

cd ..

catkin_make

cd ..

echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bash

source ~/.bashrc

cd catkin_ws

catkin_make

connect both the the USBs of RPLIDAR to PC.

ls -l /dev|grep ttyUSB

sudo chmod 666 /dev/ttyUSB0

roslaunch rplidar_ros view_rplidar.launch

Source: https://medium.com/robotics-with-ros/installing-the-rplidar-lidar-sensor-on-the-raspberry-pi-32047cde9588

9 – INTRODUCTION INDUSTRIAL ROBOTS (COBOT)

Please Refer to Manual for COBOT (File Name:COBOT.pdf)

10. Mini Project

- This experiment is considered as miniproject with your own idea, individual contribution, final submission as a presentation for 10 minutes and a report (as decided by the team leaders).

- Examples for mini project could be (i). implementation of automated robot using raspberry pi (ii). Cobot programming using ROS (iii) URDF programming XACRO + ROS (underwanter robots, fying robots, etc, any other you can think beyond the boundary.) (iii) 3D printed real hardware working project.

- You can use astra, realsense depth camera, Raspberry pi, RGB cameras, PILIDAR, any sensors like ultrasonic, IR sensor, moisture sensor, IP camera,

- Upload the working folder, doc file with steps, screen shots, a short video of 1 minute.

Mini Project Guidelines

Outcome:

- The goal is to provide deeper understanding of the subject and its application in solving industry-based problems. In addition, it also provides hands-on experience with various tools such as ROS, OPENCV-PYTHON, TURTLEBOT, INDUSTRIAL ROBOTS.

- It also helps to improve the project skills and makes students ready for taking up final year major projects. The work includes the project proposal, independent learning, project execution, and presentation.

- Individual or a team of students will choose a mini project from real time problem. Complexity of the work depends on the number of team members.

FORMAT OF MINI PROJECT REPORT

1. **TITLE OF MINI PROJECT** (Font Size 16, Bold, Uppercase, center aligned)
2. **Margins:** Left - 1.25", Right - 1", Top & Bottom - 1".
3. **Line Spacing:** 1.5
4. Title of the report
    - Font: Times New Roman (Bold)
    - Size: 16
    - Alignment: center
5. Section Headings: First level
    - Font: Times New Roman (Bold)
    - Size: 14
    - Alignment: Left
6. Section Headings: Second level
    - Font: Times New Roman (Bold)
    - Size: 12
    - Alignment: Left
7. Text of paragraph:
    - Font: Times New Roman (Normal)
    - Size: 12
    - Alignment: Left and Right justified.
8. Figures and Tables **:**
    - Caption (placed below the figure and above the table)
    - Font : Times New Roman (Bold)
    - Size : 10 point
    - Alignment : Centered
10. Page Numbering (Right)
    - Roman numbered from certificate page to list of figures page
    - Arabic 1, 2 from first chapter (introduction) to end
11. References
    - Spacing : 1.5
    - Font : Times New Roman (normal)

- Size : 12 point
- [Citation number] Author's Name, "Article Title", Journal, Publisher,
- [Citation number] Author's Name, "Title of the Book", Publication, Edition, Year of Printing.

TITLE (16, BOLD)

Provide proper title that suits to your problem.

MAIN HEADING (14, Bold)

Explain the topic in own words, with relevant figures, tables, graphs, etc. (12, Normal). Figures, graphs, tables, etc. should be center aligned.

SUB-HEADING (12, BOLD)

You can use sub-headings such as Experimental Setup, Results & Discussions, Advantages & Discussions, Applications; etc.

Figures must be drawn using draw.io online editor/ paint or anything which looks professional.

CONCLUSION (14, Bold)

Summarize the topic and write concluding remarks in your own words. (12, Normal)

REFERENCES (14, Bold)

[1] List the reference papers in the order in which they are referred, including the main reference paper. (12, Normal)

[2] Author1, Author2, and others, "Title of the paper", Journal/Conference name, Volume No., Serial No., Month, Year.

- Report should be typed and printed back to back of A4-size papers.
- Monochrome printing is suggested.
- Minimum No. of pages: 15, Maximum No. of pages: 40.
- Total No. of copies to be made: 1.
- Before printing, upload the soft-copy to me via MS teams assignment section
- (naming report to be strictly followed as Regno_Name)

The report shall be arranged under the following headings:
Scoring rubrics include the quantity, quality, complexity of the work, results, experimental analysis, clarity of presentation, and depth of understanding displayed. could learn the topic from your report. Reports should contain the following material, organized logically into sections:

Title page

Abstract

Each problem should have a description of your method, solution, and the results of the experiments.

Introduction

Introduction explaining the topic, and motivation. Specific aims of your project, use tables or figures to explain them. Mention why you have chosen this problem, what is the advantage of your work to society/industry. Mention the contribution for the work.

Literature

Discuss the previous works done in the area related to your work. Cite some resources that fits to your work. Describe the work in detail, and explain how it works on your data. Mention the advantages and drawbacks in your words in one paragraph. Tabular format is more appreciated.

- Write the main contribution to the problem statement in your words.
- How it connects to your work.
- What are the strength you observed to choose as base work for your project
- Mention the advantages and limitation of the paper according to authors.

Problem statement & Objectives

State the problem, its need in industry/society. List one or 2 objectives that you are going to achieve.

Methodology

Describe the work you used to solve the problem or to perform the experiments. Explain the methods/algorithms you used for experimentation. Explain the experimental setup, detailed dataset, programing plat form, algorithm. Provide the code used for experimentation with test results.

Result Analysis and Discussion

Present the data used for experimentation, make the list of data in a table. Provide images related to your work. Describe results on images and/or on your test data sets. Put quantitative results in tables or graphs. Use quantitative metrics such as accuracy, mean, standard deviation etc based on your application. If the word is based on industry/health application, then discuss health, safety, risk analysis, risk assessment in the report.

Output/Inference

Provide the output of your experiment and discuss how well it works. In addition, explain where the method fails, how it could be improved.

Conclusion ad Future work

Conclude your work by mentioning the methods you used for the work, result analysis, advantages and limitations, and possible future work.

References.

List of papers cited in the text. Use a consistent citation style - see examples of styles in published journal or conference papers.

APPENDIX

ROS cheat sheet

Filesystem Command-line Tools

rospack/rosstack	A tool inspecting packages/stacks.
roscd	Changes directories to a package or stack.
rosls	Lists package or stack information. roscreate-pkg Creates a new ROS package. roscreate-stack Creates a new ROS stack.
rosdep	Installs ROS package system dependen- cies.
rosmake	Builds a ROS package.
roswtf	Displays a errors and warnings about a running ROS system or launch file.
rxdeps	Displays package structure and depen- dencies.

Usage:

$ rospack find [package]

$ roscd [package[/subdir]]

$ rosls [package[/subdir]]

$ roscreate-pkg [package name]

$ rosmake [package]

$ rosdep install [package]

$ roswtf or roswtf [file]

$ rxdeps [options]

Common Command-line Tools

roscore

A collection of nodes and programs that are pre-requisites of a ROS-based system. You must have a roscore running in order for ROS nodes to communicate.

roscore is currently defined as:

master

parameter server rosout

Usage:

$ roscore

rosmsg/rossrv

rosmsg/rossrv displays Message/Service (msg/srv) data structure definitions.

Commands:

rosmsg show	Display the fields in the msg.
rosmsg users	Search for code using the msg.
rosmsg md5	Display the msg md5 sum.
rosmsg package	List all the messages in a package.
rosnode packages	List all the packages with messages.

Examples:

Display the Pose msg:

$ rosmsg show Pose

List the messages in nav msgs:

$ rosmsg package nav msgs

List the files using sensor msgs/CameraInfo:

$ rosmsg users sensor msgs/CameraInfo

rosrun

rosrun allows you to run an executable in an arbitrary package without having to cd (or roscd) there first.

Usage:

$ rosrun package executable

Example:

Run turtlesim:

$ rosrun turtlesim turtlesim node -

rosnode

Displays debugging information about ROS nodes, including publications, subscriptions and connections.

Commands:

rosnode ping	Test connectivity to node.
rosnode list	List active nodes.
rosnode info	Print information about a node.
rosnode machine	List nodes running on a particular machine.
rosnode kill	Kills a running node.

Examples:

Kill all nodes:

$ rosnode kill -a

List nodes on a machine:

$ rosnode machine aqy.local

Ping all nodes:

$ rosnode ping --all

roslaunch

Starts ROS nodes locally and remotely via SSH, as well as setting parameters on the parameter server.

Examples:

Launch on a different port:

$ roslaunch -p 1234 package filename.launch

Launch a file in a package:

$ roslaunch package filename.launch

Launch on the local nodes:

$ roslaunch --local package filename.launch

rostopic

A tool for displaying debug information about ROS topics, including publishers, subscribers, publishing rate, and messages.

Commands:

rostopic bw	Display bandwidth used by topic.
rostopic echo	Print messages to screen.
rostopic hz	Display publishing rate of topic.
rostopic list	Print information about active topics. rostopic pub Publish data to topic.
rostopic type	Print topic type.
rostopic find	Find topics by type.

Examples:

Publish hello at 10 Hz:

$ rostopic pub -r 10 /topic name std msgs/String hello

Clear the screen after each message is published:

$ rostopic echo -c /topic name

Display messages that match a given Python expression:

$ rostopic echo --filter "m.data==’foo’" /topic name

Pipe the output of rostopic to rosmsg to view the msg type:

$ rostopic type /topic name | rosmsg show

rosparam

A tool for getting and setting ROS parameters on the parameter server using YAML-encoded files.

Commands:

rosparam set	Set a parameter.
rosparam get	Get a parameter.

rosparam load	Load parameters from a file.
rosparam dump	Dump parameters to a file.
rosparam delete	Delete a parameter.
rosparam list	List parameter names.

Examples:

List all the parameters in a namespace:

$ rosparam list /namespace

Setting a list with one as a string, integer, and float:

$ rosparam set /foo "['1', 1, 1.0]"

Dump only the parameters in a specific namespace to file:

$ rosparam dump dump.yaml /namespace

rosservice

A tool for listing and querying ROS services.

rosservice list	Print information about active services.
rosservice node	Print the name of the node providing a service.
rosservice call	Call the service with the given args.
rosservice args	List the arguments of a service.
rosservice type	Print the service type.
rosservice uri	Print the service ROSRPC uri.
rosservice find	Find services by service type.

Examples:

Call a service from the command-line:

$ rosservice call /add two ints 1 2

Pipe the output of rosservice to rossrv to view the srv type:

$ rosservice type add two ints | rossrv show

Display all services of a particular type:

$ rosservice find rospy tutorials/AddTwoInts