**Controlling TurtleBots with Robot Operating System**

**Introduction**

The TurtleBot3 is the 3rd entry in the TurtleBot series. It was released in 2017 with more features than its predecessors, including the ROBOTIS smart actuator DYNAMIXEL for driving. The TurtleBot3 can be customized in various ways with different mechanical parts and sensors. Its core technology is Simultaneous Localization and Mapping (SLAM) which is done using the 360-degree distance sensor mounted on top. This allows the robot to build a map of its surroundings and navigate autonomously. The robot can also be controlled remotely from many devices, including laptops, joysticks, phones, and many more. (TurtleBot3 Overview)

ROS allows robots to be controlled using a distributed system. Every component of the robot and the programs that control it are represented as separate nodes on the computer network. This makes the system highly modular which makes it incredibly robust and maintainable. Every ROS network has a master which controls the robots and acts as an advisor for all communications in the ROS network. The master in this project will be a desktop PC running Ubuntu known as Remote PC. The robots in this project will be Raspberry PI controlled Turtlebot3s which will be known as TurtleBot PCs.

**Setup**

**Remote PC**

1. Download and install Ubuntu 16.04 on Remote PC.

Get the ISO image for Ubuntu from Ubuntu.com. It can be installed in a virtual machine and run on any host operating system like Windows or Mac OS, but it will perform better if the Remote PC is set to dual boot Ubuntu. To do this, download Rufus, a free program that converts USB drives into bootable storage devices. Run Rufus, select the USB device. If the “Create a bootable disk using” option is grayed out, click the “File System” box and select “FAT32”. Click the check box next to the “Create a bootable disk using” option, click the button next to it and select the ISO Image. Click “Start” and “Ok” or “Yes” on the following prompts to download updated SysLinux files, write in ISO Image mode, and destroy all data. Insert the new Ubuntu boot USB drive into the Remote PC and restart it. Access the boot settings, select the new USB drive, and set the computer to allow dual booting off of the USB. Once this is finished, the Remote PC will save the data from the USB and it can be removed.

1. Install ROS on Remote PC.

Start the Remote PC and finish the Ubuntu setup. Open the Terminal by pressing Ctrl+Alt+T on the keyboard. In the Terminal, run the following commands to update the apt-get package manager in Ubuntu:

$ sudo apt-get update

$ sudo apt-get upgrade

$ wget https://raw.githubusercontent.com/ROBOTIS-GIT/robotis\_tools/master/install\_ros\_kinetic.sh && chmod 755 ./install\_ros\_kinetic.sh && bash ./install\_ros\_kinetic.sh

The “$” represents the prompt for the terminal. This means that the Terminal is waiting for keyboard input. Only enter the characters after the “$”. “Sudo” tells the Terminal to run the following commands with higher privileges than granted to the normal user account. This helps prevent users from accidentally rendering their computer unusable. They must be absolutely sure they know what they are doing whenever they run a “Sudo” command. The first time it is run, it will ask for the username and password of the current account.

There are actually three commands in the last line: “wget”, “chmod”, and “bash”. “Wget” is a command that tells the Terminal to fetch files from a webserver. “Chmod 755” allows the specified program “.install\_ros\_kinetic.sh” to be executed. “bash” runs the specified program.

1. Install TurtleBot3 Packages on Remote PC

Run the following commands in the Terminal:

$ sudo apt-get install ros-kinetic-joy ros-kinetic-teleop-twist-joy ros-kinetic-teleop-twist-keyboard ros-kinetic-laser-proc ros-kinetic-rgbd-launch ros-kinetic-depthimage-to-laserscan ros-kinetic-rosserial-arduino ros-kinetic-rosserial-python ros-kinetic-rosserial-server ros-kinetic-rosserial-client ros-kinetic-rosserial-msgs ros-kinetic-amcl ros-kinetic-map-server ros-kinetic-move-base ros-kinetic-urdf ros-kinetic-xacro ros-kinetic-compressed-image-transport ros-kinetic-rqt-image-view ros-kinetic-gmapping ros-kinetic-navigation ros-kinetic-interactive-markers

$ cd ~/catkin\_ws/src/

$ git clone https://github.com/ROBOTIS-GIT/turtlebot3\_msgs.git

$ git clone https://github.com/ROBOTIS-GIT/turtlebot3.git

$ cd ~/catkin\_ws && catkin\_make

“git clone” uses the “git” command to copy files from github.com for the TurtleBot. “catkin\_make” uses catkin, a package manager that is used to build packages for ROS. If all of these commands are successful, then the installation on Remote PC is done.

**TurtleBot PC**

1. Install the latest image of the Ubiquity Robotics OS from <https://www.downloads.ubiquityrobotics.com/pi.html>.

This comes with some ROS packages preinstalled to save time during installation. Download and install Balena Etcher from <https://www.balena.io/etcher/>. Run Balena Etcher and select the Ubiquity Robotics image. Insert the micro SD card and select it as the target, then click “Flash!” Balena Etcher will then flash the image onto the micro SD card. Once this is complete, insert the micro SD card into the Raspberry PI and turn it on.

1. Ensure ROS packages are up to date.

Run the following command in the Terminal on the TurtleBot PC:

$ sudo apt-get update

$ sudo apt-get upgrade

$ wget https://raw.githubusercontent.com/ROBOTIS-GIT/robotis\_tools/master/install\_ros\_kinetic\_rp3.sh && chmod 755 ./install\_ros\_kinetic\_rp3.sh && bash ./install\_ros\_kinetic\_rp3.sh

1. Install TurtleBot3 packages.

Run the following command in the Terminal on the TurtleBot PC:

$ cd ~/catkin\_ws/src

$ git clone https://github.com/ROBOTIS-GIT/hls\_lfcd\_lds\_driver.git

$ git clone https://github.com/ROBOTIS-GIT/turtlebot3\_msgs.git

$ git clone https://github.com/ROBOTIS-GIT/turtlebot3.git

$ sudo apt-get install ros-kinetic-rosserial-python ros-kinetic-tf

1. Delete some unnecessary packages.

$ cd ~/catkin\_ws/src/turtlebot3

$ sudo rm -r turtlebot3\_description/ turtlebot3\_teleop/ turtlebot3\_navigation/ turtlebot3\_slam/ turtlebot3\_example/

1. Build packages.

$ source /opt/ros/kinetic/setup.bash

$ cd ~/catkin\_ws && catkin\_make -j1

1. Allow OpenCR to use the USB port without root permissions.

$ rosrun turtlebot3\_bringup create\_udev\_rules

**OpenCR**

1. Install the latest OpenCR source code.

Run the following commands in the Terminal on TurtleBot PC:

$ export OPENCR\_PORT=/dev/ttyACM0

$ export OPENCR\_MODEL=burger

$ rm -rf ./opencr\_update.tar.bz2

$ wget https://github.com/ROBOTIS-GIT/OpenCR-Binaries/raw/master/turtlebot3/ROS1/latest/opencr\_update.tar.bz2 && tar -xvf opencr\_update.tar.bz2 && cd ./opencr\_update && ./update.sh $OPENCR\_PORT $OPENCR\_MODEL.opencr && cd ..

1. Test the OpenCR board and the dynamixels.

Two test buttons are located on the OpenCR board near the cable that connects a USB port on the Raspberry PI to it. Switch one is located closer to the inside of the board and switch two is closer to the outside. Press and hold switch one for a few seconds to command the robot to move forward 12 inches. Press and hold switch two for a few seconds to command the robot to rotate 180 degrees in place.

**Network Setup**

**Remote PC**

1. Find the IP address for Remote PC.

You can find this by running “$ ifconfig” in the Terminal and looking for the value labelled “inet addr:”. It should look like 192.168.0.200 but with different numbers.

1. Run the following command to open the file “.bashrc”:

$ nano ~/.bashrc

“.bashrc” contains the settings for the Terminal. It is run every time a new Terminal is opened.

Note: if you are unfamiliar with the nano text editor, there is great documentation available online.

1. Set ROS\_MASTER\_URI.

Change the line that begins with “ROS\_MASTER\_URI” to ROS\_MASTER\_URI=http://<IP address of Remote PC.

1. Set ROS\_IP.

Replace the line that begins with “ROS\_HOSTNAME with ROS\_IP=<IP address of Remote PC>.

1. Set TurtleBot3 model

Add “export TURTLEBOT3\_MODEL=burger”

1. Ensure the three updated/new lines are at the bottom of .bashrc.
2. Set host mappings

Run “$ sudo nano /etc/hosts” in the Terminal.

/etc/hosts contains a table of hostname-IP address pairs that allow the computers to connect to each other using the human readable name instead of their IP addresses.

Add this line: “ubuntu@ubiquityrobotics <IP address of TurtleBot PC>”

**TurtleBot PC**

1. Find the IP address for TurtleBot PC.

You can find this by running “$ ifconfig” in the Terminal and looking for the value labelled “inet addr:”. It should look like 192.168.0.200 but with different numbers.

1. Run the following command to open the file “.bashrc”:

$ nano ~/.bashrc

1. Set ROS\_MASTER\_URI.

Change the line that begins with “ROS\_MASTER\_URI” to ROS\_MASTER\_URI=http://<IP address of Remote PC.

1. Set ROS\_IP.

Replace the line that begins with “ROS\_HOSTNAME with ROS\_IP=<IP address of TurtleBot PC>.

1. Set TurtleBot3 model

Add “export TURTLEBOT3\_MODEL=burger”

1. Ensure the three updated/new lines are at the bottom of .bashrc.
2. Set host mappings

Run “$ sudo nano /etc/hosts” in the Terminal.

/etc/hosts contains a table of hostname-IP address pairs that allow the computers to connect to each other using the human readable name instead of their IP addresses.

Add this line: “<Hostname of Remote PC> <IP address of TurtleBot PC>”

**Test Installation and Setup**

1. Ping

Run “$ ping ubuntu@<IP address of TurtleBot>”. This sends a message to the TurtleBot instructing it to send a response message. Ping is used by network administrators to test which machines are reachable on a network. If this is unsuccessful, then the two computers are likely not on the same network.

1. Connect to the TurtleBot PC using ssh.

SSH, or Secure Shell, allows a computer to be controlled remotely by another computer. Ensure both the Remote PC and TurtleBot PC are connected to the same network and run “$ ssh ubuntu@<IP address of TurtleBot PC>” to connect the robot. The Terminal should prompt for the password to the TurtleBot PC. Enter the password and you will see the hostname in that terminal change to that of the TurtleBot PC. You can now run commands and access files on the TurtleBot PC.

1. Netcat messages.

Netcat allows you to send messages between computers and acts as a chat application. Run “$ netcat -l -p 1234” on the Remote PC to have the Remote PC listen for a connection on port 1234. Then, run “$ netcat <IP address of Remote PC> 1234” on Terminal that is connected to the TurtleBot. You should see the cursor move to the next line. It is waiting for you to type a message and press enter. You will see the message appear in both Terminals.

1. Check ROS connection with Rqt\_graph.

First, make sure an instance of roscore is running by running “$ roscore” on the Remote PC. Then run “$ roslaunch turtlebot3\_bringup turtlebot3\_remote.launch” on the Terminal that is connected to the TurtleBot. The Terminal will print a lot of information to the screen. If the last line reads “Calibaration End”, then the TurtleBot started correctly.

Rqt\_graph is a useful tool that allows you to see the active Nodes and connections between them on a ROS network. Install it on the Remote PC by running “$ sudo apt-get install ros-kinetic-rqt”. Run “$rosrun rqt\_graph rqt\_graph”. It will display a GUI containing the Nodes that are active.



**Beginning Control**

1. Ensure an instance of Roscore is running by running “$ roscore” on the Remote PC.
2. Establish an ssh connection with the TurtleBot PC if you have not already and run “$ roslaunch turtlebot3\_bringup turtlebot3\_remote.launch” to bring up the TurtleBot.
3. Run “$ roslaunch turtlebot3\_teleop turtlebot3\_teleop\_key.launch”.
4. You should now be able to control the robot using the keyboard.

Check Rqt\_graph to make sure that the teleop Node is publishing to cmd\_vel and that the TurtleBot is receiving the message. This will show as an arrow connecting teleop to TurtleBot with the label “cmd\_vel.”

Control Your Turtlebot3!

---------------------------

Moving around:

w

a s d

x

w/x : increase/decrease linear velocity

a/d : increase/decrease angular velocity

space key, s : force stop

CTRL-C to quit

**ROS Overview**

As mentioned before, ROS utilizes a distributed system of control for robots. There is no central server that handles all messages sent across the network. Instead, all clients send publish and subscribe messages to a single machine, known as the Master, which establishes the connections for each client. Once the connection is established, the client does not have to send messages to the Master. This makes it easy to add and remove components from the network to make changes while the rest of the components are still functioning normally.

Every component of a ROS network that sends or receives data is known as a Node. Nodes can be anything from sensors and motors, to debugging applications and controllers. They can publish and subscribe to Topics and send Messages across them. You can run “$ rosnode list” in the Terminal to see a list of the nodes that are currently active. Rqt\_graph is another useful utility that shows the Nodes and connections between them in a GUI. You can run “$rosrun rqt\_graph rqt\_graph” to start this utility. Roscore is perhaps the most important node because it allows the Master to facilitate communications between all of the nodes in the network.

Topics are communication channels for Nodes that allow them to send data between them. They have an associated type which creates a contract between the publisher and receiver. For example, if a Topic has the type geometry\_mgs/Twist, then the subscriber must send geometry\_msgs/Twist messages on the Topic and the subscriber must expect geometry\_msgs/Twist messages on the Topic.

Common topics include /rosout, /turtle1/cmd\_vel, and turtle1/pose. /rosout gives text output to the screen, /turtle1/cmd\_vel is a topic that controls the motion of a turtle (from turtle\_sim for this example). You can run “$ rostopic list” to get a list of the published and subscribed topics.

Messages are the pieces of data that are sent by Nodes on Topics. They are used to send sensor readings, commands, and many more. They must have the same type as the Topic that is used to send them.

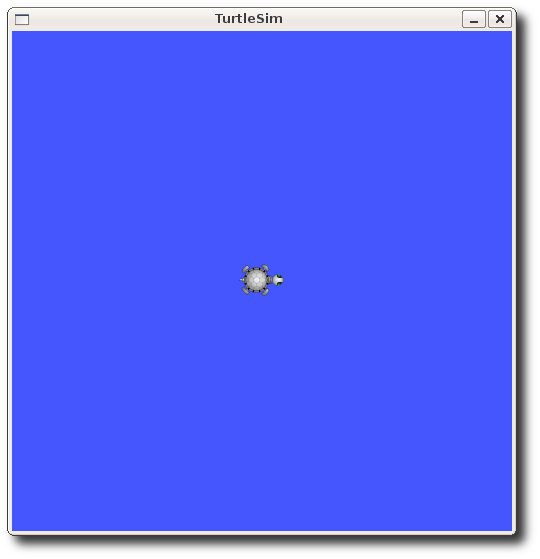
A real-world analogy of the typical ROS network is magazine subscriptions. When a Node wants to receive a specific magazine, it sends a subscription request to Master, which establishes the subscription. Now, when the publisher node releases a new magazine, a copy will be routed to the subscriber node. The publisher node is unaware of how the subscriber node gets the magazine, just like mail can be delivered in many ways, and the subscriber node is unaware of the other subscribers, just like we do not know how many people are also subscribed to our favorite magazines. This separation helps to make ROS networks very robust and maintainable.

**Simulation and SLAM**

A big feature of ROS is the ability to simulate the robots during development. This saves time setting up the robots when you want to run tests and eliminates the possibility of damaging them if something goes wrong. You can simulate the robot with TurtleSim and Gazebl.

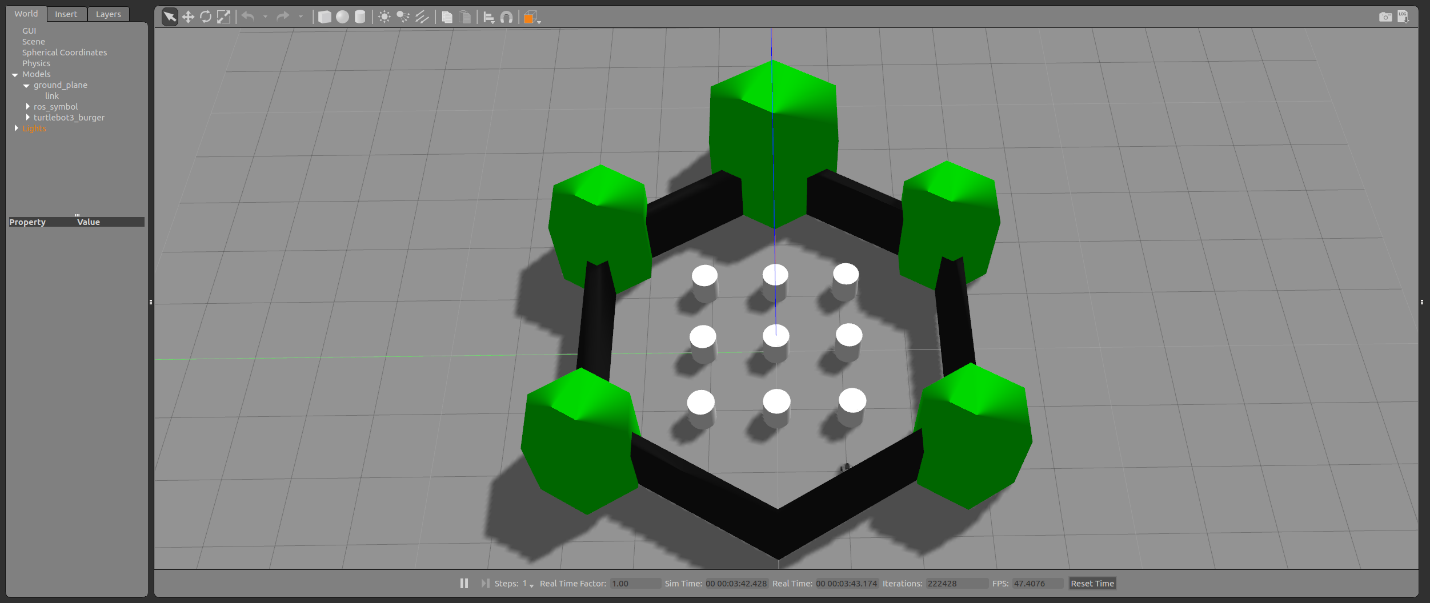
**TurtleSim**

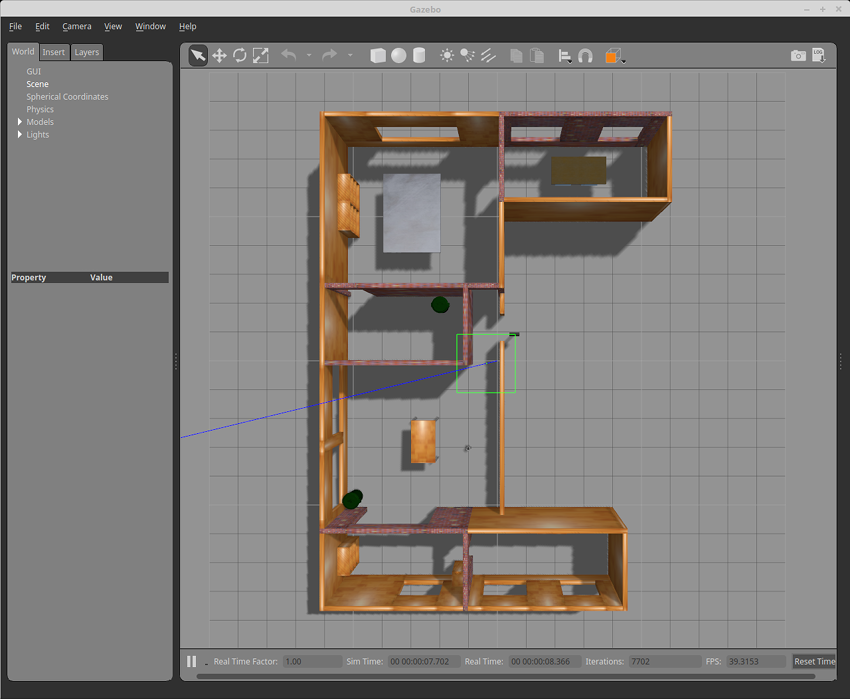
“$ rosrun turtlesim turtlesim\_node” creates a small window with a 2D sprite representing the TurtleBot. It can be controlled using the teleop\_key utility specific to TurtleSim to make it move around the screen. Start this by running “$ rosrun turtlesim turtle\_teleop\_key” in another Terminal.



**Gazebo**

Gazebo creates a 3D world with a replica of the TurtleBot3. There are three worlds available: Empty World, TurtleBot3 World, and TurtleBot3 House. Each of these allows the TurtleBot to drive around the world and interact with any objects if there are any. The TurtleBot3 World and TurtleBot3 house will appear as shown in the images below.





SLAM stands for Simultaneous Localization and Mapping, which is how the Turtlebot3 maps out the environment and navigates within it. It performs this using the laser distance sensor mounted on top. It spins continuously, reading the time it takes for the light to leave the sensor, bounce off of objects, and return to the sensor. This time is used to calculate the distance. You can see the results of SLAM with Rviz. Rviz displays data from any sensors on the TurtleBot3 in real time.

Complete the following steps to view the TurtleBot3’s data in Rviz:

1. Start the TurtleBot3.

Run “$ roslaunch turtlebot3\_bringup turtlebot3\_remote.launch” on the TurtleBot PC.

1. Run Rviz.

Run “$rosrun rviz rviz -d `rospack find turtlebot3\_description`/rviz/model.rviz”

Complete the following steps to perform virtual SLAM in Gazebo:

1. Launch Gazebo.

Run “$ roslaunch turlebot3\_gazebo turtlebot3\_world.launch”

1. Launch SLAM.

Run “$ roslaunch turtlebot3\_slam turtlebot3\_slam.launch slam\_methods:=gmapping”

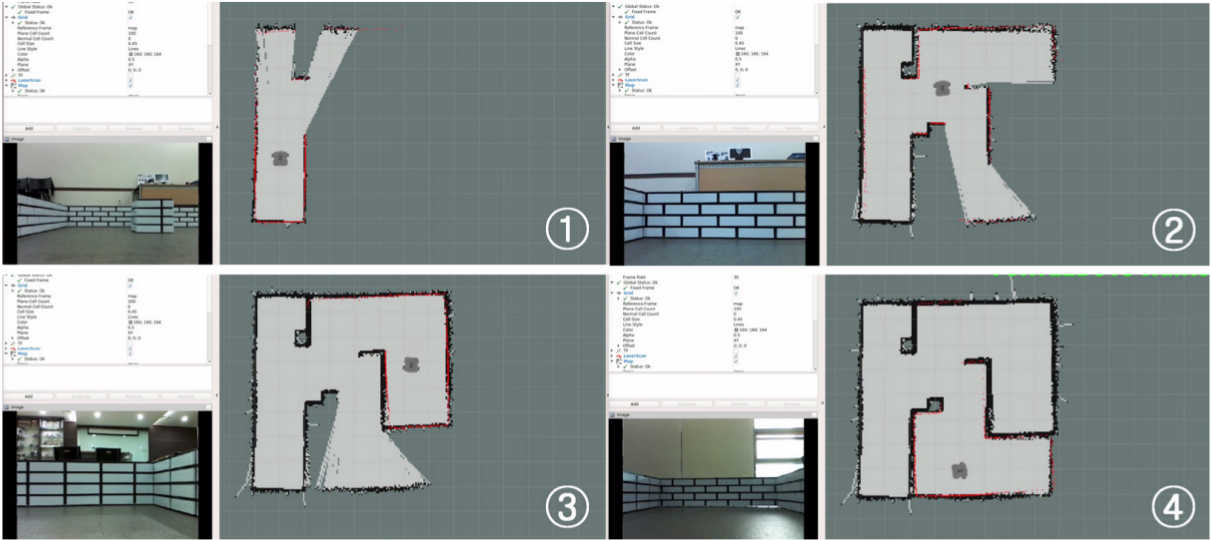
1. Remotely control the TurtleBot3.

Run “$ roslaunch turtlebot3\_teleop turtlebot3\_teleop\_key.launch”.

You should now be able to control the robot with the keyboard. Drive the robot around the environment until it has completed its map. Black areas are solid walls and light gray areas are passable.

1. Save the map.

Run “$ rosrun map\_server map\_saver -f ~/map”

A complete map will look like this: 

**Autonomous Navigation**

Random Navigation

With the map saved, run “$ roslaunch turtlebot3\_navigation turtlebot3\_navigation.launch map\_file:=$HOME/map.yaml”. This will start a new TurtleBot3 in Gazebo that will navigate around the room using the map.

Waypoint Navigation

Complete the following steps to control a TurtleBot3 using waypoints in Rviz:

1. Ensure the TurtleBot3 is running. If not, run the bringup up command.
2. Load the map.

Run “$ roslaunch turtlebot3\_navigation turtlebot3\_navigation.launch map\_file:=$HOME/map.yaml”

1. Start Rviz.
2. Set the initial pose of the robot.

Click the button labelled “2D Pose Estimate” in Rviz. This will create a large green arrow. Move the arrow to the robot’s location on the map. Then hold the left mouse button and drag so that the arrow points in the direction that the robot is facing.

1. Move the robot around with the keyboard. This allows the robot to estimate its actual position in the map.
2. Send Navigation Goal.

Click the button labelled “2D Nav Goal” in Rviz. This will create a large arrow. Move the arrow to a location on the map and set the direction just like you did when setting the initial pose. This specifies the final pose for the robot which the robot will navigate to while avoiding obstacles.

**Programming**

Your first program will create a talker and listener in Python to pass text from one Node to the other.

Your programs will be kept in a new catkin package. The catkin package manager automates the task of compiling program files and linking them together. A catkin package must have a package.xml file and a CMakeLists.txt file, both of which must be in catkin’s format.

Create a new catkin package by running “$ catkin\_create\_pkg <package name> [dependency1] [dependency2] […]. The square brackets mean the contents (the dependencies) are optional. The dependency list lets catkin know which packages it must link to the new package. For example, running “$ catkin\_create\_pkg myPackage rospy std\_msgs” will create a new package called “myPackage” that depends on the rospy and std\_msgs packages. Rospy is required for writing ROS programs in Python, which is what we will be using. Std\_msgs gives us access to the std\_msgs message type which is used to send text between Nodes.

With the new package created, use nano to customize the package.xml file. Give the package a useful description inside the description tag. It will look like this at first: “<description></description>”. This is the description tag. It has a name (description) and it can hold data in its body (in the middle between the ><). A complete description tag could look like this: “<description>My first package</description>”. Do the same for the maintainer tag, making sure to replace the email address placeholder with your email address and enter your name in the body. Also, enter BSD in the license tag.

**Talker-Listener programs**

A good place to start with programming in ROS is creating your own nodes to passing string messages back and forth. This will be done by setting up a simple publisher-subscriber pair in two separate Python files. Use “$ roscd <package name>” to change to your new package’s directory. Use “$ mkdir scripts” to make a folder named “scripts” in your package. Then use “$ cd scripts” to change to the scripts directory and “$ nano talker.py” to create a new python file. Then copy the following code:

#!/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

Talker.py

Use “$ chmod +x talker.py” to make the file executable and you are done with the talker.

“#!/usr/bin/env python” lets the Terminal know how to run the program.

“import rospy” adds the variables and functions from the rospy library to the program and “from std\_msgs.msg import String” imports the String message type into the program.

Inside the talker function, pub is created as a new publisher variable that will publish messages on the topic called “chatter” of type “String” and will hold a maximum of 10 messages that are waiting to be sent. The “init\_node” function from rospy is called to start the node with the name “talker”. Anonymous is set to true to ensure that the node is given a unique name. This allows multiple instances of the talker node to be running at the same time with no conflicts. The rate is set at 10hz, which means the node will send messages at a rate of 10 per second. The while loop will run until the node is closed and will create a string with the current time, log the string, and publish it.

Use “$ nano listener.py” to create a new python file for the listener node and copy the following code:

#!/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()

Listener.py

Use “$ chmod +x listener.py” to make it executable.

The listener has the same imports as the talker but the setup is much simpler. The talker only requires a callback function that is called each time the listener receives a message, to be initialized by rospy, and to be subscribed to the “chatter” topic published to by the talker. “Rospy.spin()” causes the listener to keep listening continuously until it is stopped.

**Starting your first nodes**

Use “$ rosrun <package name> talker.py” to start the talker node.

Use “$ rosrun <package name> listener.py” to start the listener node.

You should see the talker node printing “hello world <date>” and the listener node printing “I heard hello world <date>”.

**Your first program to control the robot**

Create a new python file called “controller.py” and copy the following code into it. Be sure to make the file executable when you are done.

#!/usr/bin/env python

# license removed for brevity

import rospy

from geometry\_msgs.msg import Twist

def reset(vel\_msg):

vel\_msg.linear.x = 0

vel\_msg.linear.y = 0

vel\_msg.linear.z = 0

vel\_msg.angular.x = 0

vel\_msg.angular.y = 0

vel\_msg.angular.z = 0

def move():

# Starts a new node

rospy.init\_node('turtle', anonymous=True)

velocity\_publisher = rospy.Publisher('/cmd\_vel', Twist, queue\_size=10)

vel\_msg = Twist()

move\_speed = 0.01

turn\_speed = 0.01

move\_amount = 0.01

turn\_amount = 0.01

#Since we are moving just in x-axis

reset(vel\_msg)

while not rospy.is\_shutdown():

#Receiveing the user's input

print("Let's move your robot")

str\_in = raw\_input("Input a direction (wasd):")

for dir in str\_in:

reset(vel\_msg)

t0 = rospy.Time.now().to\_sec()

speed = 0

#Setting the current time for distance calculus

current\_distance = 0

current\_angle = 0

if dir == "w" or dir == "s":

if dir == "w": vel\_msg.linear.x = move\_speed; speed = move\_speed

if dir == "s": vel\_msg.linear.x = -move\_speed; speed = -move\_speed

#Loop to move the turtle in an specified distance

while current\_distance < move\_amount:

#Publish the velocity

velocity\_publisher.publish(vel\_msg)

#Takes actual time to velocity calculus

t1=rospy.Time.now().to\_sec()

#Calculates distancePoseStamped

current\_distance = abs(speed)\*(t1-t0)

#After the loop, stops the robot

vel\_msg.linear.x = 0

#Force the robot to stop

velocity\_publisher.publish(vel\_msg)

elif dir == "a" or dir == "d":

if dir == "a": vel\_msg.angular.z = turn\_speed; speed = turn\_speed

if dir == "d": vel\_msg.angular.z = -turn\_speed; speed = -turn\_speed

#Loop to turn the turtle

while current\_angle < turn\_amount:

#Publish the velocity

velocity\_publisher.publish(vel\_msg)

#Takes actual time to velocity calculus

t1=rospy.Time.now().to\_sec()

#Calculates distancePoseStamped

current\_angle = abs(speed)\*(t1-t0)

#After the loop, stops the robot

vel\_msg.angular.z = 0

#Force the robot to stop

velocity\_publisher.publish(vel\_msg)

if \_\_name\_\_ == '\_\_main\_\_':

try:

#Testing our function

move()

except rospy.ROSInterruptException: pass

This program controls the robot because the robot listens for “Twist” messages on the topic “/cmd\_vel”. Twist is a data type that describes motion in three dimensions. If we have a Twist message called cmd\_vel, then cmd\_vel.linear.x tells the robot to move forward by x, and cmd\_vel.angular.z tells the robot to turn by z. You can see the data published on /cmd\_vel by running “$ rostopic echo /cmd\_vel” in another Terminal. If you get an error, ensure that nodes are publishing to the topic and that you used the correct name. You can see the topics using rqt\_graph.

This program asks the user to enter a string of movement commands with WASD. When they hit enter, each of these commands are sent to the robot to make it move. Try it out. Are there any other features you can think of to control the robot?

If the robot does not move after running this program, be sure to check the ROS network using rqt\_graph (“$ rosrun rqt\_graph rqt\_graph” to start). The “turtle” node this program creates should be publishing to cmd/vel and should be connected to the TurtleBot. If not, check the network setup.

**Useful Commands**

* nano <filename> - Open a file in the text editor “Nano”. If the file does not exist, a new file will be created.
* chmod +x <filename> - Make a file executable.
* cd <directory name> - Change directories.
* mkdir <directory name> - Create a new directory.
* roscore – Start the master node.
* rosrun <package name> <filename> - Run a ROS program.
* rosnode list – See a list of active nodes.
* rostopic
  + list – See a list of active topics.
  + echo <topic name> - See data published on a topic.
* roscd <ROS package name> - Change to the base directory of the package.
* roslaunch turtlebot3\_bringup turtlebot3\_remote.launch – Bringup a TurtleBot3 (Run on the TurtleBot PC)

Sources

<https://www.howtogeek.com/howto/linux/create-a-bootable-ubuntu-usb-flash-drive-the-easy-way/>

<http://emanual.robotis.com/docs/en/platform/turtlebot3/overview/>

<http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29>

<http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics#turtle_keyboard_teleoperation>