TechX2019 Robotics Course Documentation

Feb 29, 2020

ROS Tutorial

CHAPTER 1

Installing ROS

Your configuration must be either one from the following:

- System: Ubuntu 16.04
- or
- System: Ubuntu 18.04

If you are using Ubuntu 16.04, you must install ROS Kinetic

Ubuntu 16.04, ROS Kinetic

If you are using Ubuntu 18.04, you must install ROS Melodic

Ubuntu 18.04, ROS Melodic

In this document, we are using Ubuntu 18.04 + ROS Melodic, but if you are using Ubuntu 16.04, please change all the Melodic into Kinetic

Ubuntu 18.04 + ROS Melodic,Ubuntu 16.04, MelodicKinetic

1.1 Setup sources.list

The Tsinghua University Mirror Source (**Recommended**):

```
sudo sh -c '. /etc/lsb-release && echo "deb http://mirrors.tuna.tsinghua.edu.cn/ros/
↪ubuntu/ $DISTRIB_CODENAME main" > /etc/apt/sources.list.d/ros-latest.list'
```

or you can use the default source:

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

1.2 Setup your keys

Run the following code:

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

1.3 Update Package Index

```
sudo apt-get update
```

It may takes some time

1.4 Install ROS

- **Ubuntu 18.04:**

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

- **Ubuntu 16.04:**

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

1.5 Initialize Rosdep

Before you can use ROS, you will need to initialize rosdep.

```
sudo rosdep init
rosdep update
```

1.6 Environment Setup

- **Ubuntu 18.04:**

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

- **Ubuntu 16.04:**

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

1.7 Install More Dependencies

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

CHAPTER 2

Create a workspace

Let's create and build a catkin workspace:

```
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/
catkin_make
echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc
```

CHAPTER 3

Create ROS Package

3.1 Creating a catkin Package

```
cd ~/catkin_ws/src
```

Now use the catkin_create_pkg script to create a new package called 'techx2019' which depends on std_msgs, roscpp, and rospy:

```
catkin_create_pkg techx2019 std_msgs rospy
```

3.2 Building a catkin workspace and sourcing the setup file

Now you need to build the packages in the catkin workspace:

Note: We need to use catkin_make because we create a new package

```
cd ~/catkin_ws
catkin_make
rospack profile
```

CHAPTER 4

Writing a Simple Publisher and Subscriber

4.1 Writing the Publisher Node

Change directory into the techx2019 package

```
roscd techx2019
```

4.1.1 The Code

```
mkdir scripts
cd scripts
touch talker.py
chmod +x talker.py
```

Open the editor

```
gedit talker.py
```

Copy the following Python Code into talker.py:

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

import rospy
from geometry_msgs.msg import Twist

def talker():
    pub = rospy.Publisher('turtle1/cmd_vel', Twist, queue_size=10)
    rospy.init_node('publisher')
    rate = rospy.Rate(10) # 10hz
    while not rospy.is_shutdown():
        msg = Twist()
        msg.linear.x = 1.0
```

(continues on next page)

(continued from previous page)

```
        msg.angular.z = 1.0
        pub.publish(msg)
        rate.sleep()

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

Build the code

```
cd ~/catkin_ws
catkin_make
```

4.1.2 Test Using TurtleSim

Open roscore

```
roscore
```

Open Another Terminal to run TurtleSim

TurtleSim

```
rosrun turtlesim turtlesim_node
```

Open Another Terminal to run the code

Run the code

```
cd ~/catkin_ws/src/techx2019/scripts/
python talker.py
```

You should keep every programs running and move on!!

Open a Terminal!!

4.2 Writing the Subscriber Node

4.2.1 The Code

```
roscd techx2019/scripts/
touch listener.py
chmod +x listener.py
```

Open the editor

```
gedit listener.py
```

Copy the following Python Code into listener.py.

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

import rospy
from turtlesim.msg import Pose

def printMessage(msg):

    print msg.x
    print msg.y
    print msg.theta

def listener():

    rospy.init_node('listener')

    rospy.Subscriber('turtle1/pose', Pose, printMessage)

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

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

4.2.2 Building your nodes

```
cd ~/catkin_ws
catkin_make
```

Save and run the code:

```
roscd techx2019/scripts
python listener.py
```

CHAPTER 5

ROS Filesystem Basics

5.1 Using rospack

rospack allows you to get information about packages. In this tutorial, we are only going to cover the find option, which returns the path to package.

rospack

Usage:

```
rospack find [package_name]
```

Example:

```
rospack find techx2019
```

5.2 Using roscd

It allows you to change directory (cd) directly to a package or a stack.

roscdcdroscdros

Usage:

```
roscd [locationname[/subdir]]
```

Example:

```
roscd techx2019
```

Note: roscd will access the directory output by the rospack find command

5.3 Using rosls

It allows you to ls directly in a package by name rather than by absolute path.

roscdcd

Usage:

```
rosls [locationname[/subdir]]
```

Example:

```
rosls techx2019
```

5.4 Tab Completion(Tab)

Example:

```
roscd tec
```

Press the Tab right now, you will automatically get the command

```
roscd techx2019
```

CHAPTER 6

Roslaunch

6.1 Using roslaunch command

Usage:

```
roslaunch [package] [filename.launch]
```

First go to the techx2019 package we created and built earlier:

```
roscd techx2019
```

Create a launch directory:

```
mkdir launch
cd launch
```

6.2 The Launch File

Now let's create a launch file called turtlemimic.launch

```
touch turtlemimic.launch
gedit turtlemimic.launch
```

Paste the following:

```
<launch>

<group ns="turtlesim">

    <node pkg="turtlesim" name="turtle1" type="turtlesim_node"/>
    <node pkg="turtlesim" name="turtle2" type="turtlesim_node"/>
```

(continues on next page)

(continued from previous page)

```
    <node pkg="techx2019" name="talker" type="talker.py" output="screen"/>

</group>

</launch>
```

Run the launch file:

```
roslaunch techx2019 turtlemimic.launch
```

CHAPTER 7

Debugging Tools

7.1 env | grep ROS

check all the ROS environmental variables

7.2 rqt

rqt is a software framework of ROS that implements the various GUI tools in the form of plugins. One can run all the existing GUI tools as dockable windows within rqt! The tools can still run in a traditional standalone method, but rqt makes it easier to manage all the various windows on the screen at one moment.

```
rqt
```

7.3 rqt_graph

rqt_graph provides a GUI plugin for visualizing the ROS computation graph. Its components are made generic so that other packages where you want to achieve graph representation can depend upon this pkg

```
rqt_graph
```

7.4 rqt_tf_tree

rqt_tf_tree provides a GUI plugin for visualizing the ROS TF frame tree.

```
rqt_tf_tree
```

7.5 rostopic echo

rostopic echo topicname returns the messages being sent from the ROS master about a specific topic, topicname. To stop returning messages, press Ctrl+C. rostopic info topicname returns the message type, publishers, and subscribers for a specific topic, topicname. rostopic type topicname returns the message type for a specific topic.

```
rostopic echo <topic name>
```

7.6 nmap

```
nmap -sP 192.168.50.1/24
```

CHAPTER 8

Use gamepad to control Racecar

8.1 Turn on the robot

- Turn on the battery
- Turn on the ESC

Note: make sure your gamepad's control mode's light is off and the mode is in X

8.2 Connect the TianRacer WiFi

WiFi SSID: TianRacer

Password: www.tianbot.com

8.3 Confirm your Racecar's IP address

192.168.50.10X where X is your team number

X

Example:

If your team number is 9, your Racecar's IP address should be **192.168.50.109**

8.4 Using SSH

Open a terminal

```
ssh -X tianbot@192.168.50.10X
```

where X is your team number

For example, if your team number is 9, you should

```
ssh -X tianbot@192.168.50.109
```

Type in the password: **ros**

Note: you will not see the password on the screen, because the password typing is hidden in linux terminal

8.5 Open Racecar Bringup launch file

Run the following code **ONLY IF** you have logged into your Racecar

Don't run the code on your laptop

```
roslaunch racecar_bringup racecar_bringup.launch
```

CHAPTER 9

Use script to control Racecar

enter your techx2019 package's script folder

```
roscd techx2019/scripts
```

create and edit square.py

```
touch square.py
chmod +x square.py
gedit square.py
```

Configure environment variables on your computer: **Note:yourcarip is the IP address of your car. For example, 192.168.50.101**

```
export ROS_MASTER_URI=http://yourcarip:11311
```

check your computer's ip address

```
ifconfig
```

and then set the environment variable

```
export ROS_IP=yourcomputerip
```

After configuring your computer, run the following code on the car

```
export ROS_MASTER_URI=http://yourcarip:11311
export ROS_IP=yourcarip
```

Copy and paste the following code:

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

import rospy
from geometry_msgs.msg import Twist
```

(continues on next page)

(continued from previous page)

```
import time

def shut_down(pub, rate, msg):
    msg.linear.x = 1500
    msg.angular.z = 90
    pub.publish(msg)
    rate.sleep()

def run_command(linear_x, angular_z, t, rate):
#linear_x should be 1320-1680 where 1500 is stop
#angular_z should be 66-114 where 90 is center
#t is time
    init_time = time.time()

    while(time.time() - init_time < t):
        msg.linear.x = linear_x
        msg.angular.z = angular_z
        pub.publish(msg)
        rate.sleep()

def run(pub, rate, msg):

    run_command(1640, 66, 2, rate)

    shut_down(pub, rate, msg)



if __name__ == '__main__':

    pub = rospy.Publisher('/car/cmd_vel', Twist, queue_size=10)
    rospy.init_node('publisher')
    rate = rospy.Rate(10)
    msg = Twist()

    try:
        run(pub, rate, msg)
    except rospy.ROSInterruptException:
        shut_down(pub, rate, msg)
        pass
```

run the python code

```
python square.py
```

CHAPTER 10

Setup Cartographer

In this section we setup environment for cartographer on your own computer. (If you are running ros2go from USB, you can skip this section).

First install some required package:

```
sudo apt-get update
sudo apt-get install -y python-wstool python-rosdep ninja-build
```

Then Create a new workspace 'carto_ws'.

```
mkdir carto_ws
cd carto_ws
wstool init src
```

Merge the cartographer_ros.rosinstall file and fetch code for dependencies.

```
wstool merge -t src https://raw.githubusercontent.com/tianbot/tianbot_racecar/master/
↪cartographer_ros.rosinstall
wstool update -t src
```

Then install proto3.

```
src/cartographer/scripts/install_proto3.sh
```

Then install deb dependencies. note that the command 'sudo rosdep init' will print an error if you have already executed it since installing ROS. This error can be ignored.

```
sudo rosdep init
rosdep update
rosdep install --from-paths src --ignore-src --rosdistro=${ROS_DISTRO} -y
```

Lastly we build and install.

```
catkin_make_isolated --install --use-ninja
echo "source ~/carto_ws/install_isolated/setup.bash" >> ~/.bashrc
```

Now you can restart all terminal and move to next section.

CHAPTER 11

Mapping

The following steps are running on ROS2GO platform

Open a terminal, connect to your car.

First, set up ROS environment on both your car and your computer.

On your computer, config network variables by modifying your ~/.bashrc file:

```
gedit ~/.bashrc
```

in the text editor, add these two lines to the bottom of the file.

```
ROS_MASTER_URI=http://yourcarip:11311
ROS_IP=`hostname -I`
```

where "yourcarip" is the IP address of your car (192.168.50.10*).

On your car (first ssh -X to your car), config network variables by the same steps.

```
gedit ~/.bashrc
```

in the text editor, add these two lines to the bottom of the file.

```
ROS_MASTER_URI=http://yourcarip:11311
ROS_IP=`hostname -I`
```

where "yourcarip" is the IP address of your car (192.168.50.10*).

You can use the following code to check if your environment variables have been set correctly

```
echo $ROS_MASTER_URI
echo $ROS_IP
```

Get into your computer's SLAM package

```
roscd racecar_slam
```

Update to the latest SLAM package

```
git pull origin master
```

Make the package

```
cd ~/catkin_ws
catkin_make
```

OK, **now close all terminals and open new terminals to do the following steps.**

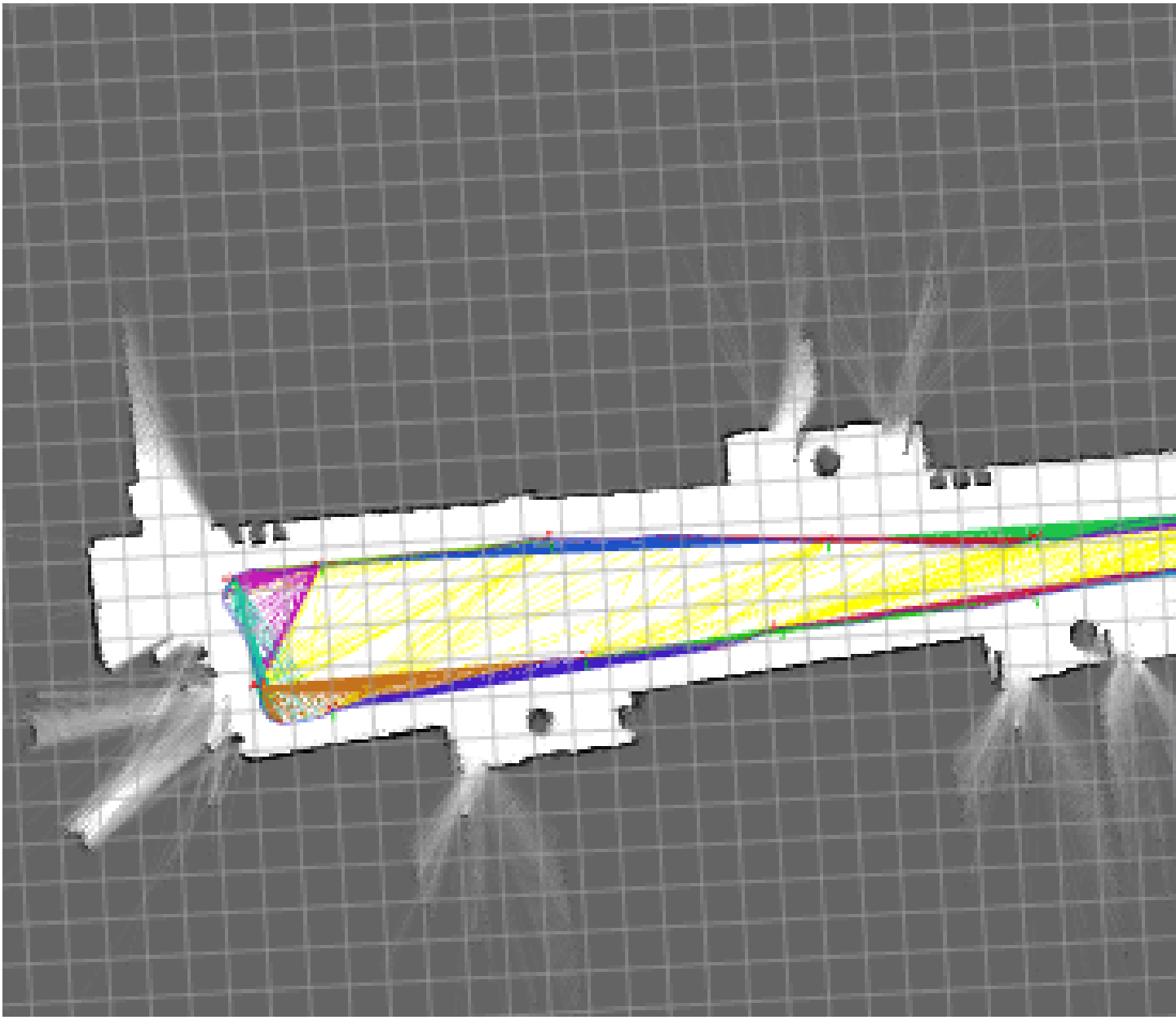
Then ssh to your car and bring up everything on your car

```
roslaunch racecar_bringup racecar_bringup.launch
```

Then go back to your own computer and open a new terminal. Run the mapping launch file to start mapping

```
roslaunch racecar_slam racecar_laser_only_cartographer.launch
```

You should be able to see something like this



Tp save the map, open a new terminal and run

```
rosrun map_server map_saver --occ 51 --free 49 -f test_carto_map
```

CHAPTER 12

Use Rviz to simulate and control MIT Racecar model

In this section we run 2-D simulation in Rviz using the MIT racecar model. Here we control the racecar in two ways:

- Using the gamepad,
- Using custom Publisher and Subscriber.

12.1 Setup

First we need to install additional required packages.

If you are running ubuntu 16.*, do the following:

```
sudo apt-get install ros-kinetic-tf2-geometry-msgs ros-kinetic-ackermann-msgs ros-
↪kinetic-joy ros-kinetic-map-server
```

If you are running ubunut 18.*, do the following:

```
sudo apt-get install ros-melodic-tf2-geometry-msgs ros-melodic-ackermann-msgs ros-
↪melodic-joy ros-melodic-map-server
```

Then we clone the github repo of MIT Racecar in our catkin_ws/src folder.

```
cd ~/catkin_ws/src
git clone https://github.com/mit-racecar/racecar_simulator.git
```

Now go back up one stage and compile the new packages:

```
cd ..
catkin_make
source ~/catkin_ws/devel/setup.bash
```

At this point all the required packages and libraries should be installed.

12.2 Quick Start

To run the simulation, in terminal we run:

```
roslaunch racecar_simulator simulate.launch
```

This will bringup a server that runs everything including roscore, the simulator, a preselected map, a model of the racecar and the joystick server.

In order to visualize the simulation, we open the Rviz simulator by typing

```
rviz
```

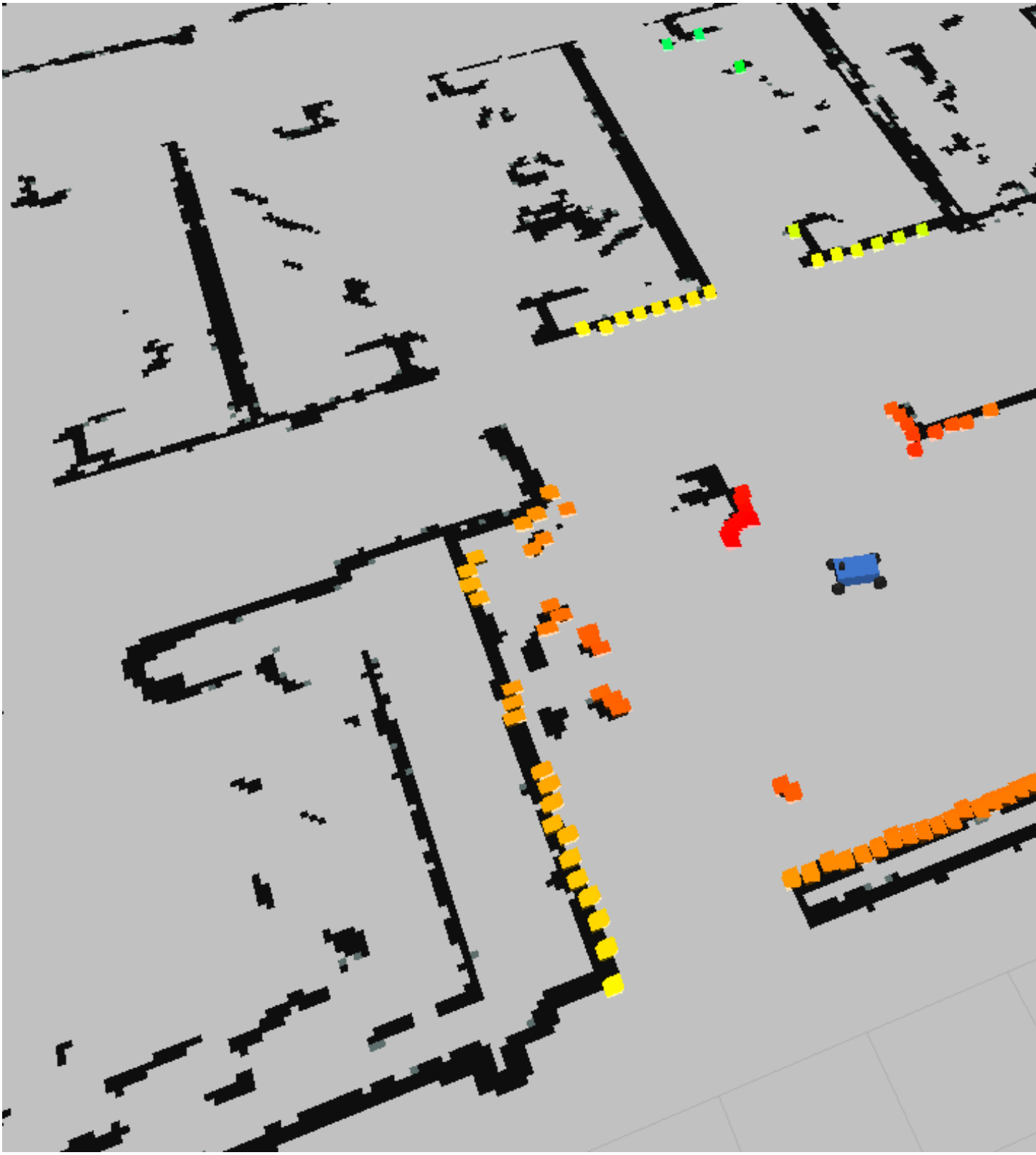
You will see the GUI of rviz popping up, with an empty world. We then click "Add" in the lower-left panel, and select tab "By topic" and add the **/map** topic and then add the **/scan** topic. Under the "By display type" tab, we add the **RobotModel** type.

In the left panel under the newly added LaserScan section, change the size to 0.1 meters for a clearer visualization of the lidar.

12.3 Controlling the car using gamepad

For this part, simply connect the usb receiver of your gamepad to your computer. **Switch the mode of the gamepad to D**, then you should be able to control the car using two joysticks.

The rviz environment with the scanning car should look like this:



12.4 Controlling the car using publisher and subscriber

CHAPTER 13

Navigation

Get into the map folder

```
roscd racecar_slam/maps/
```

Download the map data and save it into this folder

Get into the Navigation folder and change the linear velocity data

```
roscd racecar_navigation/launch/includes/
gedit tianbot_move_base.launch.xml
```

find the line of baseSpeed and change it into

```
<param name="baseSpeed" value="1600"/> <!-- pwm for motor constant speed, 1480: stop,␣
↪1440: ~0.5m/s, 1430: ~1.5m/s -->
```

Change the Navigation Launch file

```
roscd racecar_navigation/launch/
gedit racecar_amcl_nav.launch
```

Find the line of map selection and change it into

```
<arg name="map_file" default="$(find racecar_slam)/maps/first_floor.yaml" />
```

start the Navigation program

```
roslaunch racecar_navigation racecar_amcl_nav.launch
```

Open another terminal to run rViz

```
rviz
```

Add the map data inside rviz

CHAPTER 14

Setup Opencv and run tracking algorithm on local computer

In this section we install opencv on Ubuntu and implement multi-object tracking using built-in algorithm.

14.1 Setup

First we need to setup opencv environment in ubuntu.

If you are running ubuntu 18.04, then do the following

First we install pip:

```
sudo apt-get install python-pip
```

then install libopencv-dev:

```
sudo apt-get install libopencv-dev
```

then install opencv-python as follows:

```
sudo pip install opencv-python
```

If you have reached this point, you can verify your installation in terminal and check the version as follows:

```
$ python
Python 2.7.15+ (default, Nov 27 2018, 23:36:35)
[GCC 7.3.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import cv2
>>> cv2.__version__
>>> '4.1.0'
```

14.2 Multi-object tracking algorithm in opencv

Let's make a new directory under techx19 for cv-related code(you may need to change the path to go into the directory you want):

```
cd ~/catkin_ws/src/techx2019/script
mkdir cv_local
cd cv_local
touch cv_multi_test.py
gedit cv_multi_test.py
```

Then copy the following code and save it.

```
from __future__ import print_function
import sys
import cv2
from random import randint

trackerTypes = ['BOOSTING', 'MIL', 'KCF','TLD', 'MEDIANFLOW', 'GOTURN', 'MOSSE', 'CSRT
↪']

def createTrackerByName(trackerType):
  # Create a tracker based on tracker name
  if trackerType == trackerTypes[0]:
    tracker = cv2.TrackerBoosting_create()
  elif trackerType == trackerTypes[1]:
    tracker = cv2.TrackerMIL_create()
  elif trackerType == trackerTypes[2]:
    tracker = cv2.TrackerKCF_create()
  elif trackerType == trackerTypes[3]:
    tracker = cv2.TrackerTLD_create()
  elif trackerType == trackerTypes[4]:
    tracker = cv2.TrackerMedianFlow_create()
  elif trackerType == trackerTypes[5]:
    tracker = cv2.TrackerGOTURN_create()
  elif trackerType == trackerTypes[6]:
    tracker = cv2.TrackerMOSSE_create()
  elif trackerType == trackerTypes[7]:
    tracker = cv2.TrackerCSRT_create()
  else:
    tracker = None
    print('Incorrect tracker name')
    print('Available trackers are:')
    for t in trackerTypes:
      print(t)

  return tracker

# Create a video capture object to read videos
cap = cv2.VideoCapture(0)

# Read first frame
success, frame = cap.read()
# quit if unable to read the video file
if not success:
  print('Failed to read video')
  sys.exit(1)
```

(continues on next page)

(continued from previous page)

```
## Select boxes
bboxes = []
colors = []

# OpenCV's selectROI function doesn't work for selecting multiple objects in Python
# So we will call this function in a loop till we are done selecting all objects
while True:
  # draw bounding boxes over objects
  # selectROI's default behaviour is to draw box starting from the center
  # when fromCenter is set to false, you can draw box starting from top left corner
  bbox = cv2.selectROI('MultiTracker', frame)
  bboxes.append(bbox)
  colors.append((randint(0, 255), randint(0, 255), randint(0, 255)))
  print("Press q to quit selecting boxes and start tracking")
  print("Press any other key to select next object")
  k = cv2.waitKey(0) & 0xFF
  if (k == 113):  # q is pressed
    break

print('Selected bounding boxes {}'.format(bboxes))

# Specify the tracker type
trackerType = "CSRT"

# Create MultiTracker object
multiTracker = cv2.MultiTracker_create()

# Initialize MultiTracker
for bbox in bboxes:
  multiTracker.add(createTrackerByName(trackerType), frame, bbox)

# Process video and track objects
while cap.isOpened():
  success, frame = cap.read()
  if not success:
    break

  # get updated location of objects in subsequent frames
  success, boxes = multiTracker.update(frame)

  # draw tracked objects
  for i, newbox in enumerate(boxes):
    p1 = (int(newbox[0]), int(newbox[1]))
    p2 = (int(newbox[0] + newbox[2]), int(newbox[1] + newbox[3]))
    cv2.rectangle(frame, p1, p2, colors[i], 2, 1)

  # show frame
  cv2.imshow('MultiTracker', frame)


  # quit on ESC button
  if cv2.waitKey(1) & 0xFF == 27:  # Esc pressed
    break
```

Save the file, go back to terminal and excute the program:

```
python cv_multi_test.py
```

follow the instruction in terminal, and you should be able to track multiple objects.

CHAPTER 15

Use the USB-camera and run opencv on racecar

In this section we bring up the USB-camera on the racecar and run opencv for simple application. We monitor the process by streaming the video to our local computer.

15.1 Setup and bring up the USB_cam

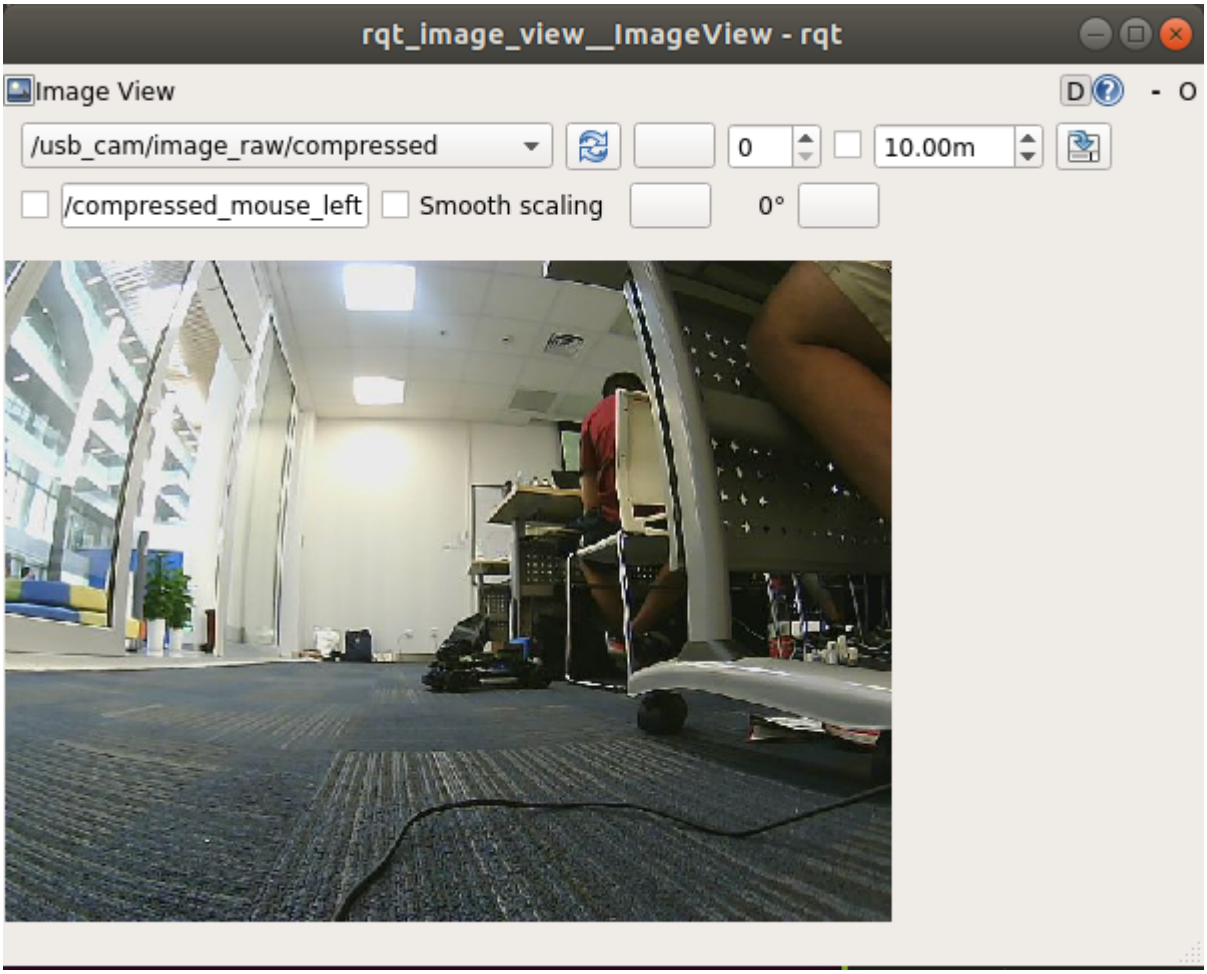
First sonnect to the car using ssh, with "-X" option, which enables graphics interface.

```
ssh -X tianbot@192.168.50.10*
```

Then simply running command:

```
roslaunch usb_cam usb_cam-test.launch
```

A window will pop up, select the menu on the top-left corner for /usb_cam/image_raw/compressed, then we should be able to see the video from the camera.



15.2 Running opencv

Opencv has already been installed on the Jetson Nano board on the racecar. To see it's version, we simply test:

```
$ python
Python 2.7.15rc1 (default, Nov 12 2018, 14:31:15)
    [GCC 7.3.0] on linux2
    Type "help", "copyright", "credits" or "license" for more information.
>>> import cv2
>>> cv2.__version__
>>> '3.3.1'
```

Unfortunately, for opencv version 3.3.1, the MultiTracker class(see the last tutorial that we run on the local computer) has not been supported. Here we demonstrate simple image processing techniques as follows:

15.2.1 Open USB_cam in opencv

Here we demonstrate how to stream the USB_cam using opencv. Note that we don't need to launch any node to read the camera signal. First create a python file "streaming.py"

```
gedit streaming.py
```

then copy and paste the following:

```
import cv2

cap = cv2.VideoCapture(0)

while True:
    success, frame = cap.read()
    cv2.imshow("streaming",frame)
    if cv2.waitKey(1) & 0xFF == 27:  # Esc pressed
        break
```

15.2.2 Using Color filter

Here we demonstrate how to do simple color filtering.

First create another file called "filtering.py"

```
gedit filtering.py
```

Then copy and paste the following:

```
import numpy as np
import cv2

cap = cv2.VideoCapture(0)

def nothing(x):
    pass

cv2.namedWindow('image',cv2.WINDOW_NORMAL)
cv2.resizeWindow('image',(600,200))
cv2.createTrackbar('minH', 'image', 0, 255, nothing)
cv2.createTrackbar('minS', 'image', 0, 255, nothing)
cv2.createTrackbar('minV', 'image', 0, 255, nothing)
cv2.createTrackbar('maxH', 'image', 0, 255, nothing)
cv2.createTrackbar('maxS', 'image', 0, 255, nothing)
cv2.createTrackbar('maxV', 'image', 0, 255, nothing)

while True:
    _, frame = cap.read()
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV) # convert to hsv encoding for better␣
↪processing

    minH = cv2.getTrackbarPos('minH', 'image')
    minS = cv2.getTrackbarPos('minS', 'image')
    minV = cv2.getTrackbarPos('minV', 'image')
    maxH = cv2.getTrackbarPos('maxH', 'image')
    maxS = cv2.getTrackbarPos('maxS', 'image')
    maxV = cv2.getTrackbarPos('maxV', 'image')

    a = cv2.waitKey(5) & 0xFF
    if a == ord('p'):
        print('minH: ', minH, '\nmaxH: ', maxH,'\nminS : ', minS, '\nmaxS : ', maxS\
```

(continues on next page)

(continued from previous page)

```
          ,'\nminV : ',minV,'\nmaxV : ',maxV)
    lowerpink = np.array([minH,minS,minV])
    upperpink = np.array([maxH,maxS,maxV])
    # print(lowerpink + '\n' + upperpink)

    mask = cv2.inRange(hsv,lowerpink, upperpink)
    res = cv2.bitwise_and(frame,frame, mask = mask)

    # median = cv2.bilateralFilter(res,15,75,75)

    # cv2.imshow('median',descale(median,3))
    cv2.imshow('frame',frame)
    cv2.imshow('mask',mask)
    cv2.imshow('res',res)

    k = cv2.waitKey(5) & 0xFF
    if k == 27:
        break

cv2.destroyAllWindows()
cap.release()
```

You shuold be able to use the slider and select the color you want.

15.2.3 Edge detection

Here we demonstrate how to do simple edge detection.

First create another file called “edge.py”

```
gedit edge.py
```

Then copy and paste the following:

```
import cv2
import numpy as np

cap = cv2.VideoCapture(0)

def nothing(x):
    pass

cv2.namedWindow('image',cv2.WINDOW_NORMAL)
cv2.resizeWindow('image',(600,200))
cv2.createTrackbar('th1', 'image', 0, 255, nothing)
cv2.createTrackbar('th2', 'image', 0, 255, nothing)
while True:
    _, frame = cap.read()

    th1 = cv2.getTrackbarPos('th1', 'image')
    th2 = cv2.getTrackbarPos('th2', 'image')

    laplacian = cv2.Laplacian(frame,cv2.CV_64F)
    sobelx = cv2.Sobel(frame,cv2.CV_64F, 1, 0, ksize = 5)
    sobely = cv2.Sobel(frame,cv2.CV_64F, 0, 1, ksize = 5)
    edges = cv2.Canny(frame, th1, th2)
```

(continues on next page)

(continued from previous page)

```
    newedges = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
    ret,thresh = cv2.threshold(newedges,127,255,0)
    cv2.imshow('thresh', thresh)
    contours, hierarchy = cv2.findContours(edges,cv2.RETR_TREE,cv2.CHAIN_APPROX_
↪SIMPLE)[-2:]
    cv2.drawContours(frame,contours,-1,(0,0,255),1)
    # print(len(contours))

    # cv2.imshow('original',frame)
    # cv2.imshow('laplacian',laplacian)
    # cv2.imshow('sobelx',sobelx)
    # cv2.imshow('sobely', sobely)
    cv2.imshow('edges', edges)
    cv2.imshow('frame', frame)

    k = cv2.waitKey(5) & 0xFF
    if k == 27:
        break

cv2.destroyAllWindows()
cap.release()
```

You should be able to run the file and do simple edge detection.

CHAPTER 16

Setting up Turtlebot3

Turtlebot3 is only available in **ROS Kinetic** and **ROS Melodic**

The installation methods for ROS Kinetic and ROS Melodic are **different**:

16.1 Download & Install Turtlebot3 Package

Please choose only one from the following installation method according to your environment

- **ROS Kinetic (Ubuntu 16.04)**

```
sudo apt install ros-kinetic-turtlebot3-*
cd ~/catkin_ws
catkin_make
source ~/catkin_ws/devel/setup.bash
rospack profile
```

- **ROS Melodic (Ubuntu 18.04)**

```
cd ~/catkin_ws/src
git clone https://github.com/ROBOTIS-GIT/turtlebot3.git
git clone https://github.com/ROBOTIS-GIT/turtlebot3_msgs.git
git clone https://github.com/ROBOTIS-GIT/turtlebot3_simulations.git
git clone https://github.com/ROBOTIS-GIT/turtlebot3_autorace.git
cd ..
rosdep install --from-paths src -i -y
catkin_make
source ~/catkin_ws/devel/setup.bash
rospack profile
```

16.2 Turtlebot3 Model Config

There are two kinds of model for Turtlebot3. If you don't choose one of them, the program **will not** run. For general purpose, we choose to use "burger" model.

```
echo "export TURTLEBOT3_MODEL=burger" >> ~/.bashrc
source ~/.bashrc
```

16.3 Close 3D acceleration

if you are using **Ubuntu in Virtual Machine**

You **dont't** have to do so if you are using native Ubuntu

```
echo "export SVGA_VGPU10=0" >> ~/.bashrc
source /.bashrc
```

16.4 Test Launch

If everything is set up correctly, you should get a opened simulator with a small robot inside it.

```
roslaunch turtlebot3_gazebo turtlebot3_world.launch
```

You can use **Ctrl+C** to exit the program

CHAPTER 17

Writing Waypoint Nevigation Algorithm

Enter **techx2019** package scripts directory

```
roscd techx2019/scripts
```

Create a blank python file and edit it

```
touch waypoint.py
gedit waypoint.py
```

Paste the following code into it:

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

import rospy
import tf
import numpy as np
import matplotlib.pyplot as plt
from geometry_msgs.msg import Twist
from nav_msgs.msg import Odometry
from math import pi, sqrt, atan2

WAYPOINTS = [[0.5,0],[1,0],[1,0],[1,0.5],[1,1],[1,1],[0.5,1],[0,1],[0,1],[0,0.5],[0,
↪0]]

class PID:
    """
    Discrete PID control
    """
    def __init__(self, P=0.0, I=0.0, D=0.0, Derivator=0, Integrator=0, Integrator_
↪max=10, Integrator_min=-10):
        self.Kp = P
        self.Ki = I
        self.Kd = D
        self.Derivator = Derivator
```

(continues on next page)

(continued from previous page)

```
        self.Integrator = Integrator
        self.Integrator_max = Integrator_max
        self.Integrator_min = Integrator_min
        self.set_point = 0.0
        self.error = 0.0

    def update(self, current_value):
        self.error = self.set_point - current_value
        if self.error > pi:  # specific design for circular situation
            self.error = self.error - 2*pi
        elif self.error < -pi:
            self.error = self.error + 2*pi
        self.P_value = self.Kp * self.error
        self.D_value = self.Kd * ( self.error - self.Derivator)
        self.Derivator = self.error
        self.Integrator = self.Integrator + self.error
        if self.Integrator > self.Integrator_max:
            self.Integrator = self.Integrator_max
        elif self.Integrator < self.Integrator_min:
            self.Integrator = self.Integrator_min
        self.I_value = self.Integrator * self.Ki
        PID = self.P_value + self.I_value + self.D_value
        return PID

    def setPoint(self, set_point):
        self.set_point = set_point
        self.Derivator = 0
        self.Integrator = 0

    def setPID(self, set_P=0.0, set_I=0.0, set_D=0.0):
        self.Kp = set_P
        self.Ki = set_I
        self.Kd = set_D

class turtlebot_move():
    def __init__(self):
        rospy.init_node('turtlebot_move', anonymous=False)
        rospy.loginfo("Press CTRL + C to terminate")
        rospy.on_shutdown(self.stop)

        self.x = 0.0
        self.y = 0.0
        self.theta = 0.0
        self.pid_theta = PID(0,0,0)  # initialization

        self.odom_sub = rospy.Subscriber("odom", Odometry, self.odom_callback)
        self.vel_pub = rospy.Publisher('cmd_vel', Twist, queue_size=10)
        self.vel = Twist()
        self.rate = rospy.Rate(10)
        self.counter = 0
        self.trajectory = list()

        # track a sequence of waypoints
        for point in WAYPOINTS:
            self.move_to_point(point[0], point[1])
            rospy.sleep(1)
        self.stop()
```

(continues on next page)

(continued from previous page)

```
        rospy.logwarn("Action done.")

        # plot trajectory
        data = np.array(self.trajectory)
        np.savetxt('trajectory.csv', data, fmt='%f', delimiter=',')
        plt.plot(data[:,0],data[:,1])
        plt.show()


    def move_to_point(self, x, y):
        # Compute orientation for angular vel and direction vector for linear vel
        diff_x = x - self.x
        diff_y = y - self.y
        direction_vector = np.array([diff_x, diff_y])
        direction_vector = direction_vector/sqrt(diff_x*diff_x + diff_y*diff_y)  #␣
↪normalization
        theta = atan2(diff_y, diff_x)

        # We should adopt different parameters for different kinds of movement
        self.pid_theta.setPID(1, 0, 0)     # P control while steering
        self.pid_theta.setPoint(theta)
        rospy.logwarn("### PID: set target theta = " + str(theta) + " ###")

        # Adjust orientation first
        while not rospy.is_shutdown():
            angular = self.pid_theta.update(self.theta)
            if abs(angular) > 0.2:
                angular = angular/abs(angular)*0.2
            if abs(angular) < 0.01:
                break
            self.vel.linear.x = 0
            self.vel.angular.z = angular
            self.vel_pub.publish(self.vel)
            self.rate.sleep()

        # Have a rest
        self.stop()
        self.pid_theta.setPoint(theta)
        #self.pid_theta.setPID(1, 0, 0)   # PI control while moving
        self.pid_theta.setPID(1, 0.02, 0.2)  # PID control while moving

        # Move to the target point
        while not rospy.is_shutdown():
            diff_x = x - self.x
            diff_y = y - self.y
            vector = np.array([diff_x, diff_y])
            linear = np.dot(vector, direction_vector) # projection
            if abs(linear) > 0.2:
                linear = linear/abs(linear)*0.2

            angular = self.pid_theta.update(self.theta)
            if abs(angular) > 0.2:
                angular = angular/abs(angular)*0.2

            if abs(linear) < 0.01 and abs(angular) < 0.01:
                break
            self.vel.linear.x = linear
```

(continues on next page)

(continued from previous page)

```
            self.vel.angular.z = angular
            self.vel_pub.publish(self.vel)
            self.rate.sleep()

        self.stop()


    def stop(self):
        self.vel.linear.x = 0
        self.vel.angular.z = 0
        self.vel_pub.publish(self.vel)
        rospy.sleep(1)


    def odom_callback(self, msg):
        # Get (x, y, theta) specification from odometry topic
        quarternion = [msg.pose.pose.orientation.x,msg.pose.pose.orientation.y,\
                       msg.pose.pose.orientation.z, msg.pose.pose.orientation.w]
        (roll, pitch, yaw) = tf.transformations.euler_from_quaternion(quarternion)
        self.theta = yaw
        self.x = msg.pose.pose.position.x
        self.y = msg.pose.pose.position.y

        # Make messages saved and prompted in 5Hz rather than 100Hz
        self.counter += 1
        if self.counter == 20:
            self.counter = 0
            self.trajectory.append([self.x,self.y])
            rospy.loginfo("odom: x=" + str(self.x) + ";  y=" + str(self.y) + ";
↪theta=" + str(self.theta))


if __name__ == '__main__':
    try:
        turtlebot_move()
    except rospy.ROSInterruptException:
        rospy.loginfo("Action terminated.")
```

Save the file and exit it

To run the waypoint nevigation algorithm, first **open a terminal** and run the turtlebot world

```
roslaunch turtlebot3_gazebo turtlebot3_world.launch
```

IMPORTANT: make the python file executable

```
chmod +x waypoint.py
```

Open another terminal, run the python code

```
python waypoint.py
```

You can see the turtlebot moving in the world, however, it would bump into the barrier. We will discuss about this later.

CHAPTER 18

Using roslaunch to simplify the program

We have learned how to use roslaunch. NOw we can use it on our turtlebot waypoint nevigation program.

First, get into the package launch directory

```
roscd techx2019/launch/
```

Create a new blank launch file called **turtlebot3_waypoint.launch** and edit it

```
touch turtlebot3_waypoint.launch
gedit turtlebot3_waypoint.launch
```

Copy and paste the following launch file code:

```
<launch>
<arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger, waffle,␣
↪waffle_pi]"/>
<arg name="x_pos" default="-2.0"/>
<arg name="y_pos" default="-0.5"/>
<arg name="z_pos" default="0.0"/>

<include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find turtlebot3_gazebo)/worlds/turtlebot3_world.
↪world"/>
    <arg name="paused" value="false"/>
    <arg name="use_sim_time" value="true"/>
    <arg name="gui" value="true"/>
    <arg name="headless" value="false"/>
    <arg name="debug" value="false"/>
</include>

<param name="robot_description" command="$(find xacro)/xacro --inorder $(find␣
↪turtlebot3_description)/urdf/turtlebot3_$(arg model).urdf.xacro" />

<node pkg="gazebo_ros" type="spawn_model" name="spawn_urdf"  args="-urdf -model␣
↪turtlebot3_$(arg model) -x $(arg x_pos) -y $(arg y_pos) -z $(arg z_pos) -param␣
↪robot_description" />
```

(continues on next page)

(continued from previous page)

```
<node pkg="techx2019" name="waypoint" type="waypoint.py" output="screen"/>
</launch>
```

Save and close the file

Run the launch file

```
roslaunch turtlebot3_gazebo turtlebot3_waypoint.launch
```

CHAPTER 19

Get in touch with SLAM

19.1 Start the turtlebot3 world

```
roslaunch turtlebot3_gazebo turtlebot3_world.launch
```

19.2 Start the SLAM RViz

If your system is **Ubuntu 16.04, run this code to install SLAM algorithm:**

```
sudo apt install ros-kinetic-gmapping
roslaunch turtlebot3_slam turtlebot3_slam.launch
```

If your system is **Ubuntu 18.04:**

```
roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=karto
```

19.3 Open teleop keyboard control

```
rosrun turtlebot3_teleop turtlebot3_teleop_key
```

CHAPTER 20

Edit Hosts file

20.1 Check your Hosts file

```
cat /etc/hosts
```

The returned result should be **something like this**:

```
127.0.0.1   localhost
127.0.1.1   chris-Inspiron-7373

# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

20.2 Edit the file

open the file editor

```
sudo gedit /etc/hosts
```

Copy and paste the following code to the end of the file:

```
192.168.50.101   tianbot-01
192.168.50.102   tianbot-02
192.168.50.103   tianbot-03
192.168.50.104   tianbot-04
192.168.50.105   tianbot-05
192.168.50.106   tianbot-06
```

(continues on next page)

(continued from previous page)

```
192.168.50.107   tianbot-07
192.168.50.108   tianbot-08
192.168.50.109   tianbot-09
192.168.50.110   tianbot-10
```

make sure that your file is looking **something like this** right now: **Note: Don't copy paste the following code.**

```
127.0.0.1   localhost
127.0.1.1   chris-Inspiron-7373

# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters

192.168.50.101   tianbot-01
192.168.50.102   tianbot-02
192.168.50.103   tianbot-03
192.168.50.104   tianbot-04
192.168.50.105   tianbot-05
192.168.50.106   tianbot-06
192.168.50.107   tianbot-07
192.168.50.108   tianbot-08
192.168.50.109   tianbot-09
192.168.50.110   tianbot-10
```

Save the file and close it

CHAPTER 21

Useful Links:

21.1 Piazza

Piazza.

21.2 ROS Wiki

ROSWiki.