# 

# Table of Content

[**Objective**](#_gw1z78c0r527) **3**

[**Platform**](#_wr12wwaynw0u) **3**

[**Before we start**](#_iyl10qgwrpip) **4**

[**ROS Intro**](#_8ny1n1i95h35) **4**

[**What is ROS?**](#_qq4i62q4eagb) **4**

[**Setup ROS Kinetic and Turtlebot 2 packages**](#_ctkvcsizhys8) **4**

[**Packages**](#_qy3nnusuiejz) **6**

[1.face\_rec](#_f3xl5sy1hr11) 6

[2. find\_face](#_6h4n2ptswgj7) 8

[3. robot\_control](#_8b8t10gw0zf0) 10

[4. user\_input](#_fu07fu99iifo) 14

[**Control the servos arm**](#_z59hcvskhpk) **16**

[**References**](#_jxez12rqz0kb) **22**

# 

# Objective

Following:

1. Robot must be able to move forward, backward, left, right, and turn around.
2. Robot must have some simple body gestures like greeting, hello, goodbye, shake hand, etc.
3. Robot must have speech capabilities: word or sentence recognition and text to speech.
4. Robot must have some vision capabilities: recognize human face (not who this person is, only that human is present and where) , recognize object (like big red box or yellow round object etc), or recognize another robot (which one).
5. Robot must use some type of Machine Learning, Evolutionary algorithm, neural net or fuzzy logic. Take it from previous projects and integrate.
6. You have to deliver 3 items: (1) report in Word, (2) presentation in PowerPoint, (3) video. Report must be inclusive and the role of each student from the team should be clear. PPT presentation must allow you to deliver a good speech in front of class, what did you achieve, what else you want to learn.

# Platform

This project is for the [TurtleBot2](https://www.turtlebot.com/turtlebot2/). (From [Wikipedia](https://en.wikipedia.org/wiki/TurtleBot)) TurtleBot is a low-cost, personal robot kit with open source software. TurtleBot was created at Willow Garage by Melonee Wise and Tully Foote in November 2010. The TurtleBot kit consists of a mobile base, 3D Sensor, laptop computer, and the TurtleBot mounting hardware kit. In addition to the TurtleBot kit, users can download the TurtleBot SDK from the ROS wiki.

The TurtleBot2 uses Robot Operating System [ROS Kinetic](http://wiki.ros.org/kinetic/Installation) as its software platform.



Figure 1

# Before we start

The contents in this document is not original. We documented steps and information from various sources (e.g., tutorials and setup) along the way as we work for the demos. This is mainly for educational purposes not beneficial. Throughout the project, we have learned and worked with the ROS programming to control many feature that include in the turtlebot.

# ROS Intro

I recommend reading at least the first five pages of Gentle introduction to ROS, digital version available [here](http://wiki.ros.org/ROS/Installation). Also, check out the ROS wiki [here](http://wiki.ros.org/ROS/Introduction#What_is_ROS.3F). Just remember to not worry yet if the next two sections mean nothing to you right know; I will try to explain why they are relevant in the following sections. Also, they will be covered again in class along with other topics in greater detail. If you need more help you can also use the reference boot left in the lab.

# What is ROS?

From the ROS wiki: ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly used functionality, message passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers.

The ROS runtime "graph" is a peer-to-peer network of processes (potentially distributed across machines) that are loosely coupled using the ROS communication infrastructure. ROS implements several different styles of communication, including synchronous RPC-style communication over services, asynchronous streaming of data over topics, and storage of data on a Parameter Server. These are explained in greater detail in our Conceptual Overview.

# Setup ROS Kinetic and Turtlebot 2 packages

First, We are going to install ROS kinetic so we will be using Ubuntu 16.04 LTS. This version of ubuntu is compatible with ROS kinetic, anything newer or older version of Ubuntu may not work.

1. To install turtlebot 2 packages, It is quite tricky because there are so many versions of packages out there. We have tried many packages for turtlebot2, but we also have failed many times and have to reinstall a new Ubuntu and ROS again. Luckily, we found a very neat and convenient piece of code for this setup. The author is Damien Jade Duff, he has everything include in the script. If you want to install these package the hard way or having doubt about the script, please skip this step and go to straight step b.

$ git clone git clone <https://github.com/Phasor2/ROS_and_Turtlebot2_PKG/>

$ cd ROS\_and\_Turtlebot2\_PKG

$ chmod +x ros\_turtlebot2\_pkg.sh

$ ./ros\_turtlebot2\_pkg.sh

and follow the instruction in the shell. It took around 15 minutes on virtual machine.

B. To install **ROS Kinetic** on Ubuntu 16.04 run command:

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

**Rosdep** for dependencies

sudo rosdep init  
 rosdep update

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

C. When we are done installing all of the required packages for ROS, we start creating packages.

* Create a workspace
  + $ mkdir -p ~/catkin\_ws/src  
    $ cd ~/catkin\_ws/  
    $ catkin\_make
* Make sure ROS\_PACKAGE\_PATH environment variable includes the directory you're in.
  + $ echo $ROS\_PACKAGE\_PATH
* Create a package
  + $ cd ~/catkin\_ws/src
  + $ catkin\_create\_pkg beginner\_tutorials std\_msgs rospy roscpp
  + $ cd ~/catkin\_ws
  + $ catkin\_make
* To add the workspace to your ROS environment you need to source the generated setup file:
  + $ . ~/catkin\_ws/devel/setup.bash

# Packages

For this project we created four packages:

1. face\_rec
2. find\_face
3. robot\_control
4. user\_input

Now let’s talk about each package in details.

## 1.face\_rec

The python script in this package deals the image that the Astra camera captures. We when run the camera, we get X and Y values. After getting the image, the node publishes the X value; which is the center of the face to find\_face.

**Code:**

#!/usr/bin/env python

# BEGIN ALL

import rospy

from std\_msgs.msg import String

from sensor\_msgs.msg import Image

import cv2, cv\_bridge

from time import sleep

face\_cascade =cv2.CascadeClassifier('/home/phong/facial\_rec/haarcascade\_frontalface\_default.xml')

eye\_cascade = cv2.CascadeClassifier('/home/phong/facial\_rec/haarcascade\_eye.xml')

class face\_rec:

def \_\_init\_\_(self):

self.bridge = cv\_bridge.CvBridge()

self.image\_sub = rospy.Subscriber('camera/rgb/image\_raw',Image, self.image\_callback)

def image\_callback(self, msg):

rate = rospy.Rate(10) # 10hz

image = self.bridge.imgmsg\_to\_cv2(msg,"bgr8")

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

faces = face\_cascade.detectMultiScale(gray, scaleFactor=1.5, minNeighbors=5)

for (x,y,w,h) in faces:

cv2.rectangle(image, (x,y), (x+w, y+h),(255,0,0), 2)

roi\_gray = gray[y:y+h, x:x+w]

roi\_color = image[y:y+h, x:x+w]

eyes =eye\_cascade.detectMultiScale(roi\_gray)

rospy.loginfo("x: %d and y: %d", x, y)

pub = rospy.Publisher('face\_rec/pos\_x',String, queue\_size=1)

pub.publish(str(x))

rate.sleep()

for (ex,ey,ew,eh) in eyes:

cv2.rectangle(roi\_color, (ex,ey), (ex+ew, ey+eh), (0,255,0),2)

cv2.imshow('afterimg',image)

cv2.waitKey(3)

rospy.init\_node('face\_rec')

Face\_rec = face\_rec()

rospy.spin()

# END ALL

###############################################################

Figure 2

Figure 2 shown above, shows in details how the nodes are connected together and it shows how /camera/rgb/image\_raw captures the image. And, /face\_rec node publishes the X value to topic /face\_rec/pos\_x and to find\_face node.

We can obtain this graph by $ rosrun rqt\_graph rqt\_graph

## **2. find\_face**

In this package, we get the position of the face from face\_rec. As the person in front of the camera moves, the node sends commands to either turn right or left to the topic /keyop\_vel\_smoother/raw\_cmd\_vel. This node is a subscriber (when getting the X value from face\_rec).

**Code:**

#!/usr/bin/env python

import rospy

from std\_msgs.msg import String

from geometry\_msgs.msg import Twist

from math import radians

from time import sleep

pup = rospy.Publisher('/keyop\_vel\_smoother/raw\_cmd\_vel', Twist, queue\_size=1)

def callback(data):

x=int(data.data)

rospy.loginfo("string to int %d",x)

#movement here

r = rospy.Rate(50);

#let's turn at 45 deg/s

turn\_cmd = Twist()

turn\_cmd.linear.x = 0

#if x is small which face is at the left keep turn left until (x center 250) fuzzy logic

#range 240-260

if 220 <= x <= 280 :

rospy.loginfo("STOP TURNING")

r.sleep()

elif x < 220 :

#turn left

rospy.loginfo("Turning left")

turn\_cmd.angular.z = radians(10);

pup.publish(turn\_cmd)

r.sleep()

elif x>280 :

#turn right

rospy.loginfo("Turning right")

turn\_cmd.angular.z = radians(-10);

pup.publish(turn\_cmd)

r.sleep()

#two keep going back and forth. Go forward for 2 seconds (10 x 5 HZ) then turn for 3 second

''' while not rospy.is\_shutdown():

for x in range(0,6):

rospy.loginfo("Turning")

self.cmd\_vel.publish(turn\_cmd)

r.sleep()

'''

def find\_face():

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

rospy.Subscriber("face\_rec/pos\_x",String, callback)

rospy.spin()

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

try:

find\_face()

except:

rospy.loginfo("node terminated.")

###############################################################

## 3. robot\_control

This package is all about getting the base to move around. So, we created 8 different motions. Each motion can be performed based on a user input. It’s very self explanatory as we can see in the code below, if the user enters a number between 1-8, the robot will perform one of the eight motions.

**Code:**

#!/usr/bin/env python

import rospy

from geometry\_msgs.msg import Twist

from math import radians

from std\_msgs.msg import Int32

from time import sleep

pub = rospy.Publisher('/keyop\_vel\_smoother/raw\_cmd\_vel', Twist, queue\_size=1)

def callback(data):

rospy.loginfo("Command numer: %d", data.data)

x=data.data

r = rospy.Rate(5);

go\_cmd =Twist()

turn\_cmd =Twist()

if x==1:

rospy.loginfo("GO STRAIGHT")

go\_cmd.linear.x=0.3

pub.publish(go\_cmd)

r.sleep()

elif x==2:

rospy.loginfo("TURN LEFT")

turn\_cmd.linear.x=0

turn\_cmd.angular.z=radians(180)

pub.publish(turn\_cmd)

r.sleep()

elif x==3:

rospy.loginfo("TURN RIGHT")

turn\_cmd.linear.x=0

turn\_cmd.angular.z=radians(-180);

pub.publish(turn\_cmd)

r.sleep()

#draw a square

elif x==4:

y = 0

while y !=1 :

for y in range(0,15):

rospy.loginfo("GO STRAIGHT")

go\_cmd.linear.x=0.2

pub.publish(go\_cmd)

r.sleep()

sleep(1)

#draw a triangle

elif x==5:

y = 0

while y !=1 :

for y in range(0,10):

rospy.loginfo("GO STRAIGHT")

go\_cmd.linear.x=0.3

go\_cmd.angular.z = 0

pub.publish(go\_cmd)

r.sleep()

sleep(1)

for y in range (0,15):

rospy.loginfo("TURNING")

turn\_cmd.linear.x=0

turn\_cmd.angular.z=radians(45)

pub.publish(turn\_cmd)

r.sleep()

sleep(1)

#back and fort

elif x==6:

y = 0

while y !=1 :

for y in range(0,15):

rospy.loginfo("GO STRAIGHT")

go\_cmd.linear.x=0.3

go\_cmd.angular.z = 0

pub.publish(go\_cmd)

r.sleep()

sleep(1)

#turn 180

for y in range (0,14):

rospy.loginfo("TURNING")

turn\_cmd.linear.x=0

turn\_cmd.angular.z=radians(83)

pub.publish(turn\_cmd)

r.sleep()

sleep(1)

#draw a circle Counterclockwise

elif x==7:

y = 0

while y !=1 :

rospy.loginfo("GO circle counter clockwise")

go\_cmd.linear.x=0.3

go\_cmd.angular.z = radians(90)

pub.publish(go\_cmd)

r.sleep()

#draw a circle Clockwise

elif x==8:

y = 0

while y !=1 :

rospy.loginfo("GO circle counter clockwise")

go\_cmd.linear.x=0.3

go\_cmd.angular.z = radians(-90)

pub.publish(go\_cmd)

r.sleep()

def robot\_control():

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

#SUBSRIBE TO TOPIC

rospy.Subscriber("motion\_num", Int32, callback)

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

rospy.spin()

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

try:

robot\_control()

except:

rospy.loginfo("node terminated.")

###############################################################

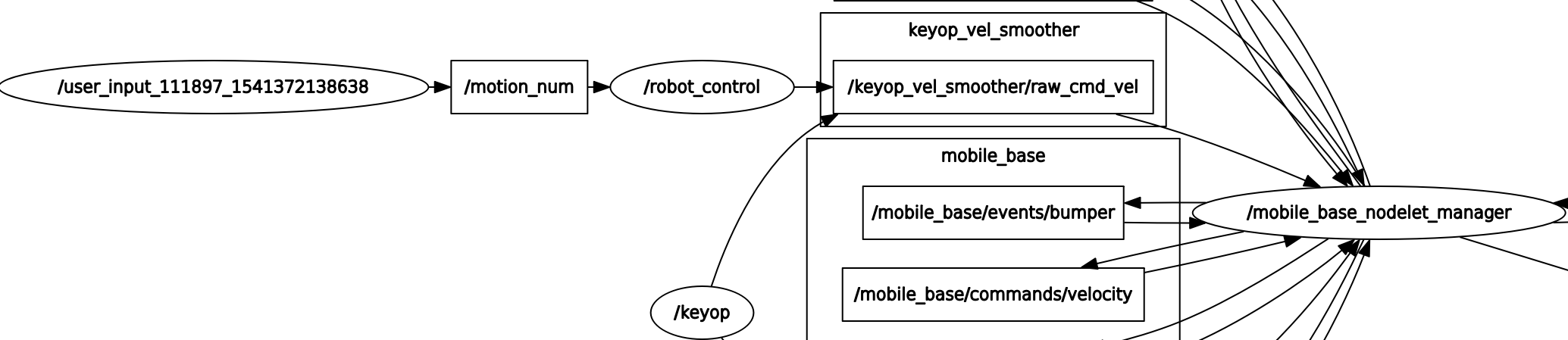


Figure 3

In figure 3 shown above, node /user\_input\_111897\_1541372138638 is publishing the value of X to the topic /motion\_num; which is the user input. And the node /robot\_control is subscribing to that topic to basically get the value of X.

## 4. user\_input

This package is very straightforward as we can see in the code below. This script basically gets the X value from the user, and prompt the user with help instructions. We used argparse which makes it easy to write user-friendly command-line interfaces.

**Code:**

#!/usr/bin/env python

# BEGIN ALL

import rospy

import sys

from std\_msgs.msg import Int32

from sensor\_msgs.msg import Image

import argparse, textwrap

def User\_input():

# DECLARE TOPIC

pub = rospy.Publisher('motion\_num', Int32, queue\_size=10)

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

parser = argparse.ArgumentParser(description='Robot\_controller',usage='use "roslaunch user\_input user\_input.launch -num" for more information -help', formatter\_class=argparse.RawTextHelpFormatter)

parser.add\_argument("num", type=int, default=1,help=textwrap.dedent('''\

1 = Go forward

2 = Spin Left

3 = Spin Right

4 = Draw a Square

5 = Draw a Triangle

6 = Go back and Fort

7 = Circle Counterclockwise

8 = Circle Clockwise

'''))

args = parser.parse\_args()

rate = rospy.Rate(10) # 10hz

while not rospy.is\_shutdown():

pub.publish(args.num)

rate.sleep()

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

try:

User\_input()

except rospy.ROSInterruptException:

Pass

# Control the servos arm

There are 3 files in the package that we need to focus on. They are

controller\_manager.launch

start\_tilt\_controller.launch

tilt.yaml

So contoller\_manager.launch this is where you set the usb port and max id for the motor

<!-- -\*- mode: XML -\*- -->

<launch>

<node name="dynamixel\_manager" pkg="dynamixel\_controllers" type="controller\_manager.py" required="true" output="screen">

<rosparam>

namespace: dxl\_manager

serial\_ports:

pan\_tilt\_port:

port\_name: "/dev/ttyUSB1"

baud\_rate: 1000000

min\_motor\_id: 1

max\_motor\_id: 25

update\_rate: 20

</rosparam>

</node>

</launch>

Start\_tilt\_controller.launch this is where you declare the name of the controller that going to have topics. Make sure the name is match with tilt.yaml we are going to see next

<launch>

<!-- Start tilt joint controller -->

<rosparam file="$(find my\_dynamixel\_tutorial)/tilt.yaml" command="load"/>

<node name="tilt\_controller\_spawner" pkg="dynamixel\_controllers" type="controller\_spawner.py"

args="--manager=dxl\_manager

--port pan\_tilt\_port

tilt\_controller1

tilt\_controller2

tilt\_controller3

tilt\_controller4

tilt\_controller5

tilt\_controller6

tilt\_controller7

tilt\_controller8

tilt\_controller9

tilt\_controller10"

output="screen"/>

</launch>

Tilt.yaml is the find that will define the values of each servos. We took note the id and limit from Roboplus software then get the limits for the servos here

tilt\_controller1:

controller:

package: dynamixel\_controllers

module: joint\_position\_controller

type: JointPositionController

joint\_name: claw\_1f

joint\_speed: 1.0

motor:

id: 1

init: 512

min: 409

max: 818

tilt\_controller2:

controller:

package: dynamixel\_controllers

module: joint\_position\_controller

type: JointPositionController

joint\_name: claw\_1f

joint\_speed: 1.0

motor:

id: 2

init: 512

min: 341

max: 648

tilt\_controller3:

controller:

package: dynamixel\_controllers

module: joint\_position\_controller

type: JointPositionController

joint\_name: claw\_1f

joint\_speed: 1.0

motor:

id: 3

init: 512

min: 307

max: 716

tilt\_controller4:

controller:

package: dynamixel\_controllers

module: joint\_position\_controller

type: JointPositionController

joint\_name: claw\_1f

joint\_speed: 1.0

motor:

id: 4

init: 512

min: 341

max: 648

tilt\_controller5:

controller:

package: dynamixel\_controllers

module: joint\_position\_controller

type: JointPositionController

joint\_name: claw\_1f

joint\_speed: 1.0

motor:

id: 5

init: 512

min: 205

max: 614

tilt\_controller6:

controller:

package: dynamixel\_controllers

module: joint\_position\_controller

type: JointPositionController

joint\_name: claw\_1f

joint\_speed: 1.0

motor:

id: 6

init: 512

min: 409

max: 682

tilt\_controller7:

controller:

package: dynamixel\_controllers

module: joint\_position\_controller

type: JointPositionController

joint\_name: claw\_1f

joint\_speed: 1.0

motor:

id: 7

init: 512

min: 614

max: 1023

tilt\_controller8:

controller:

package: dynamixel\_controllers

module: joint\_position\_controller

type: JointPositionController

joint\_name: claw\_1f

joint\_speed: 1.0

motor:

id: 8

init: 512

min: 409

max: 682

tilt\_controller9:

controller:

package: dynamixel\_controllers

module: joint\_position\_controller

type: JointPositionController

joint\_name: claw\_1f

joint\_speed: 1.0

motor:

id: 9

init: 512

min: 358

max: 665

tilt\_controller10:

controller:

package: dynamixel\_controllers

module: joint\_position\_controller

type: JointPositionController

joint\_name: claw\_1f

joint\_speed: 1.0

motor:

id: 10

init: 512

min: 375

max: 614

Now we can launch the servos

Starting by a new terminal

$roslaunch controller\_manager.launch

And second terminal

$roslaunch start\_tilt\_controller.launch

We should get something like this

[INFO] [1541392073.543571]: pan\_tilt\_port controller\_spawner: All services are up, spawning controllers...

[INFO] [1541392073.866271]: Controller tilt\_controller1 successfully started.

[INFO] [1541392073.958678]: Controller tilt\_controller2 successfully started.

[INFO] [1541392074.037563]: Controller tilt\_controller3 successfully started.

[INFO] [1541392074.277643]: Controller tilt\_controller4 successfully started.

[INFO] [1541392074.438679]: Controller tilt\_controller5 successfully started.

[INFO] [1541392074.571134]: Controller tilt\_controller6 successfully started.

[INFO] [1541392074.643010]: Controller tilt\_controller7 successfully started.

[INFO] [1541392074.691523]: Controller tilt\_controller8 successfully started.

[INFO] [1541392074.757360]: Controller tilt\_controller9 successfully started.

[INFO] [1541392074.917674]: Controller tilt\_controller10 successfully started.

Unhandled exception in thread started by

Then we check the rostopic if they are running

$rostopic list

/diagnostics

/motor\_states/pan\_tilt\_port

/rosout

/rosout\_agg

/tilt\_controller1/command

/tilt\_controller1/state

/tilt\_controller10/command

/tilt\_controller10/state

/tilt\_controller2/command

/tilt\_controller2/state

/tilt\_controller3/command

/tilt\_controller3/state

/tilt\_controller4/command

/tilt\_controller4/state

/tilt\_controller5/command

/tilt\_controller5/state

/tilt\_controller6/command

/tilt\_controller6/state

/tilt\_controller7/command

/tilt\_controller7/state

/tilt\_controller8/command

/tilt\_controller8/state

/tilt\_controller9/command

/tilt\_controller9/state

Now we can publish values for these topics example

rostopic pub -1 /tilt\_controller10/command std\_msgs/Float64 -- 1.5

# References

<http://wiki.ros.org/ROS/Tutorials/CreatingPackage>

<http://wiki.ros.org/catkin/Tutorials/create_a_workspace>

Damien Jade Duff:

<https://bitbucket.org/damienjadeduff/456_kinetic_turtlebot/src/b8eacebf174b?at=master>