**Project 2 - DIM**

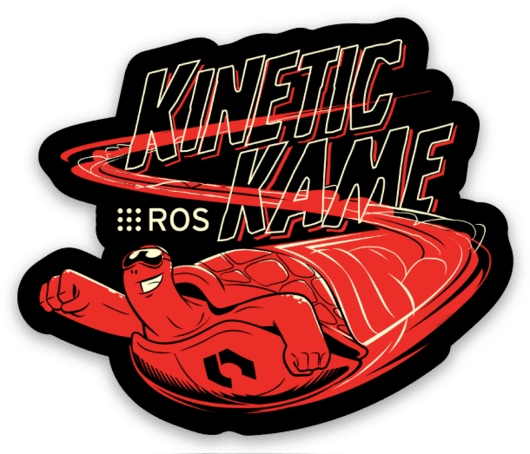
Stephen Poanessa

Himanshu Devi

Adel Alkharraz

ECE 478 - Winter 2018

**ROS Kinetic Installation (Ubuntu 16.04)**

1. Setup source.list to allow pi to accept software from packages.ros.org:

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

2. Setup the keys:

sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116

*Note: If you experience issues connecting to the keyserver, you can try substituting hkp://pgp.mit.edu:80 or hkp://keyserver.ubuntu.com:80 in the previous command.*

3.Ensure Debian package is up to date:

sudo apt-get update

4. Install ROS:

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

5. Initialized ROS Dependencies:

sudo rosdep init

rosdep update

**ROS - Robot Operating System**

**Making a ROS Workspace**

ROS requires a workspace in which to build and run projects. The name of the workspace does not matter but the common convention is to use the name “catkin\_ws”. The following terminal commands can be used in the in order to create a catkin workspace named “catkin\_ws”:

$ mkdir -p ~/catkin\_ws/

$ cd ~/catkin\_ws/

$ catkin\_make

$ source devel/setup.bash

*Note: Anytime you open a new terminal to work on a ROS project, you must run the “source devel/setup.bash” command in the workspace. ROS commands will not work if you forget to do this.*

**Making a ROS Package**

1. Navigate to the src file within the given catkin\_ws.

$catkin\_create\_pkg nameOfPackage std\_msgs Int32 rospy roscpp

The command shown above specifies support of std\_msgs and Int32 data types. “rospy” specifies the ability to compile python code while “roscpp” specifies C++ capabilities.

**Project Specific Workspace - DIM\_WS**

For implementation of Project 2 requirements, a ROS workspace named “dim\_ws” was created using the methods shown in the sections above. Within this workspace a package called “service\_tut” was created. Within the package a folder exists called “scripts” which contains the project Python source files.

**Hardware for Speech and Sound**

**Microphone**

A USB microphone (MAONO AU-410) was purchased for use as an audio input for Google Diologflow and Amazon Polly. The device has the following specifications:

* Transducer: Electret condenser.
* Frequency Response: 30Hz to 18kHz.
* Sensitivity: -30dB+/-3dB(0dB=1V/Pa@1kHz).
* Signal/Noise Ratio: 74dB Sample rate, 16bit/24bi.
* Recording resolution:up to 96KHz/48KHz/ 44.1KHz.
* Connector: Standard USB type A ( Plug-and-play for PC/MAC).

Testing of the purchase microphone has yielded excellent results; good audio quality, appropriate sensitivity, and great background noise rejection.

*Note: Google Diologflo requires mono audio files to convert and this microphone is dual channel. In order to address this issue, the output of the recording is converted to FLAC format before sending to google’s network service.*



**Speakers:**

A compact set of USB speakers (BeBomBasics SP20) were purchased for outputting audio. These speakers were selected due to the low purchase price and as a result of the aesthetic similarities to DIM. Quality of these speakers are somewhat disappointing, with a notable whine present when plugged into the Pi.

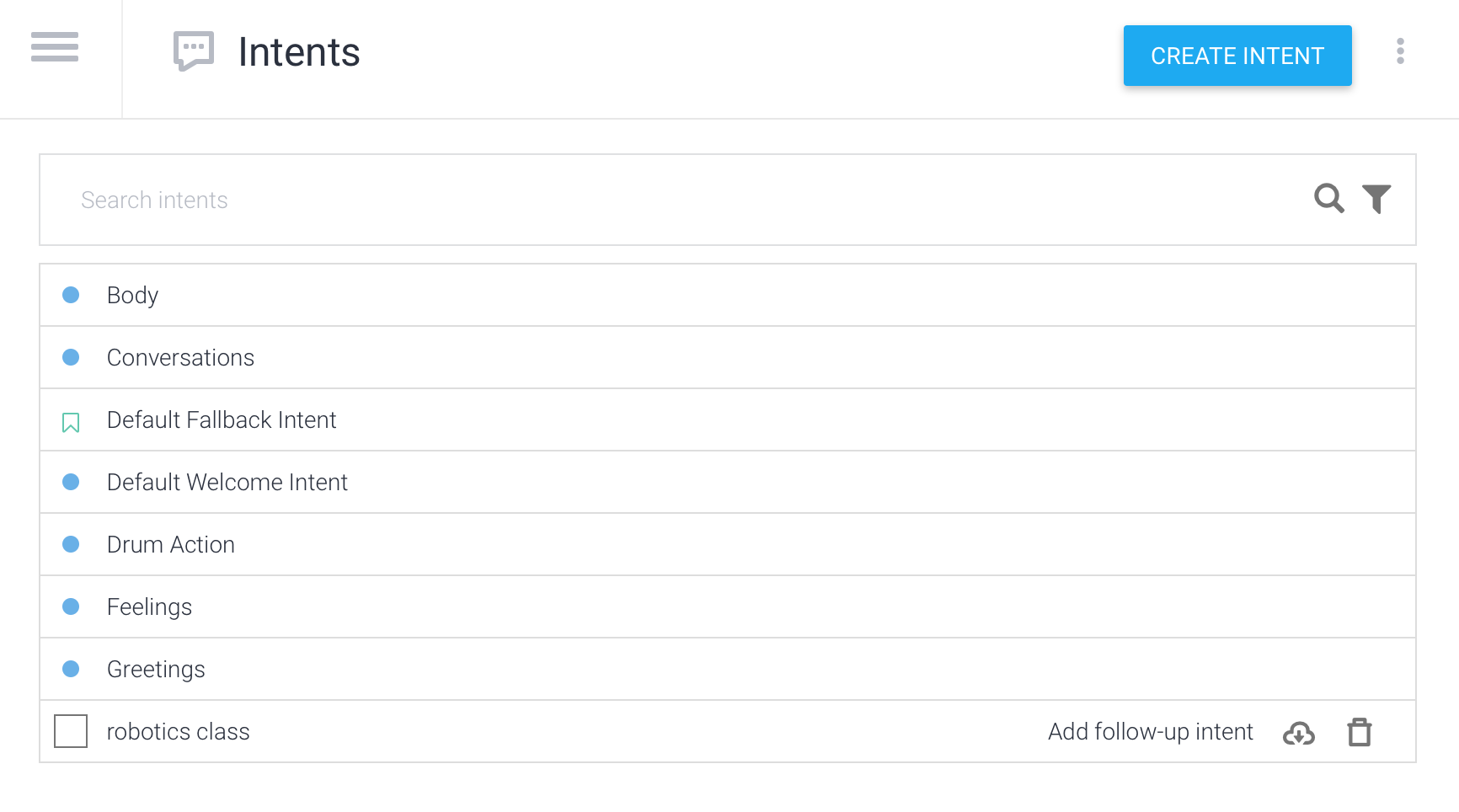
**Google Dialogflow**

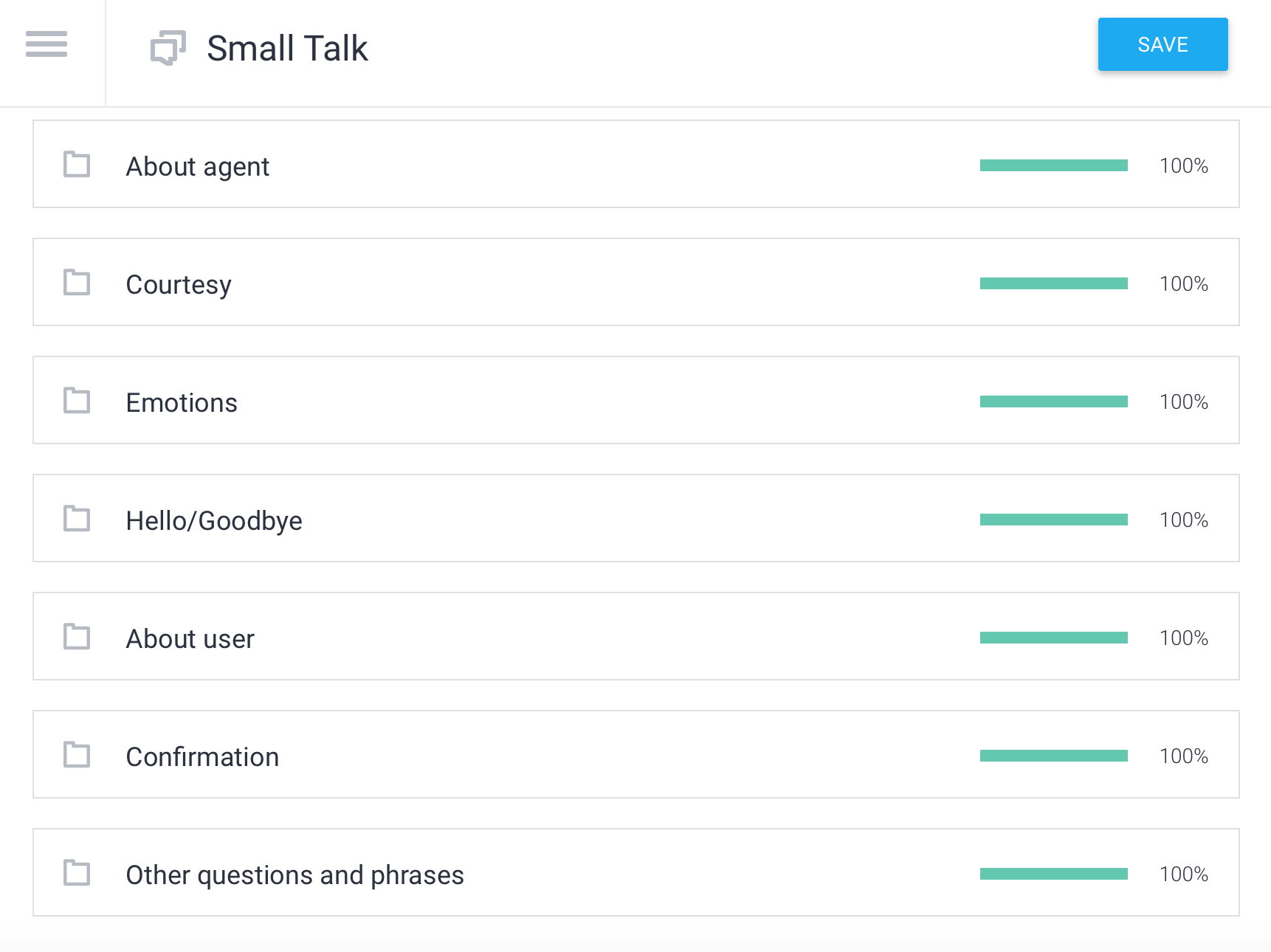
In Google Dialogflow, the first step is to create an Agent (in our case our agent named Dimbot). In the Intents section where it is used for defining a specific action that the user can invoke by using one of the defined terms in the Dialogflow console. Where in our Dimbot has the following intents:

* Body
* Feelings
* Drum action
* Greeting
* Robotic class.
* Conversations

The Entities section can be thought of as a recognizable parameter that is used by Dialogflow to respond to a given user input. In the case of our project, one entity was built for the purpose of allowing the user to verbally request for DIM to play a specific drumming routine.

In the training section of Dialogflow we can add and observe the training data where the agents had learned so far, while the history page shows a basic version of the conversation our agent got engaged in.

****

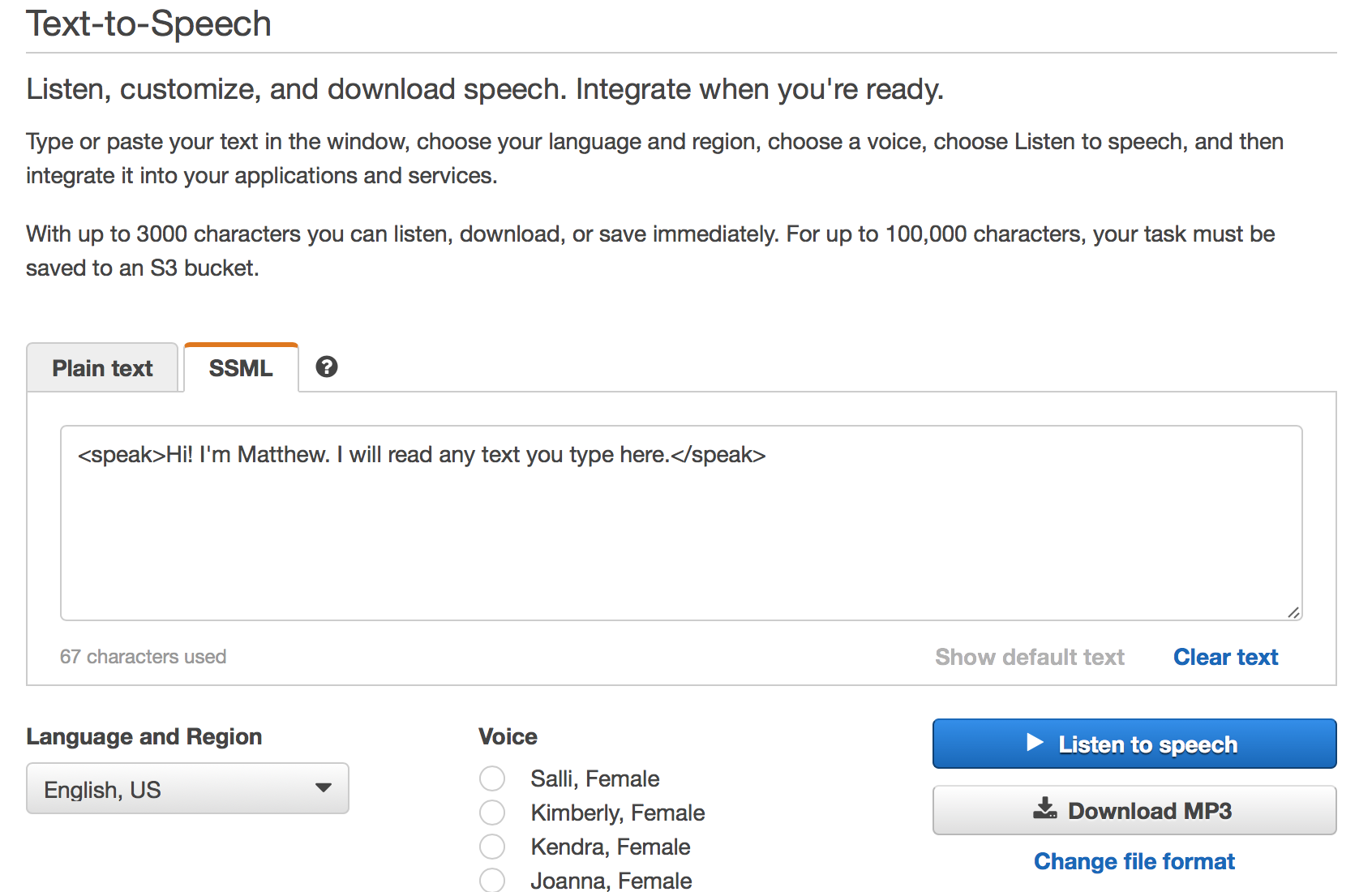
****

**Amazon Polly**

**Create an amazon poly accounts**

1. In the text-to-speech section select SSML, Language and Region select English, US and then select the desired voice (Matthew, Male is selected for our robot).
2. In the input field type the desired sentence to be converted to speech. “Tags” can also be used to better mimic the desired speech by varying parameters such as sound level, rate, pitch, breaks, etc.
3. After testing the speech it will be downloaded as mp3 file. Amazon Polly is used in this project to transfer each line for theater play script (Paul’s role) from text to speech and downloaded each line in a single mp3 file which converted to wav files so it can be used in ROS.

Amazon Polly is also used in conjunction with Google DialogFlow using ROS. A subscriber receives character response from Google DialogFlow and then publishes it to Amazon Polly, which converts it to audio. After this conversion process, the ROS script plays the received audio.

****

**Robot Theatre**

For the theater performance, DIM has been programmed to interact with two other robots:turtlebot and Einsteinbot. In order arbitrate lines, a publishing node with a line incrementing counter was implemented. All of the robots have their own nodes and are set to subscribe the the master node that is incrementing the line counter. When a line matches a given robot’s line number, that robot will respond by playing the Amazon Polly recording of the specified line. The program is designed in such a way that the counter will not increment until the given robot has completed the speech of the line.

**Documentation**

**ROS Scripts:**

/home/pi/dim\_ws/src/dim/scripts

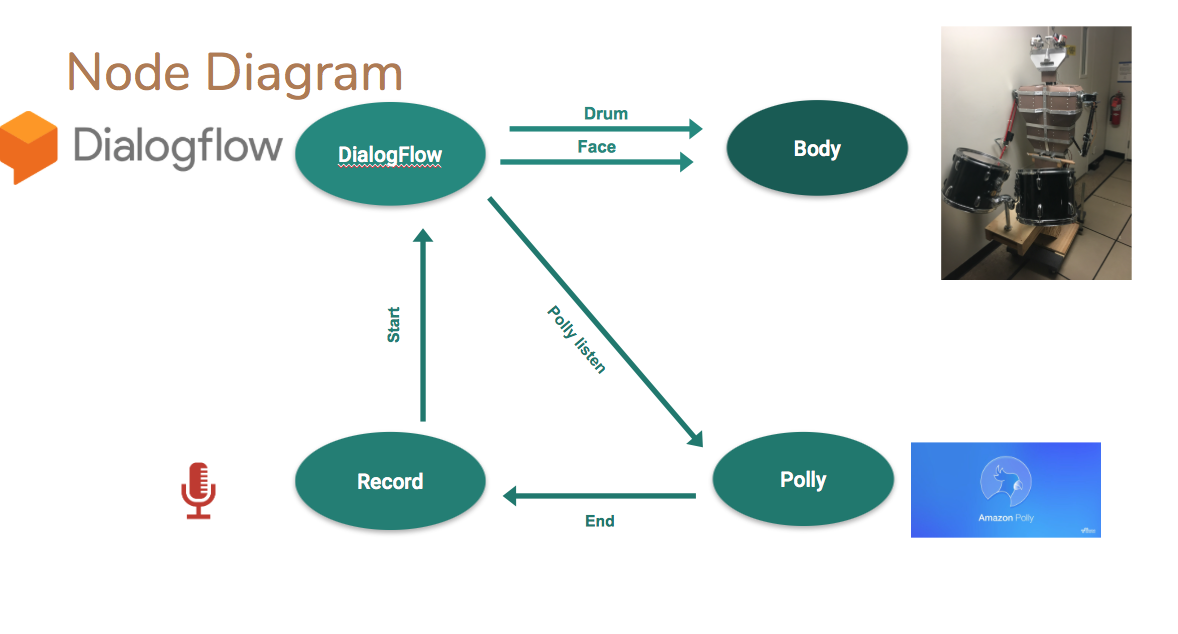
**Amazon Polly Theater Recordings:**

/home/pi/dim\_ws/src/dim/scripts/Act 1

/home/pi/dim\_ws/src/dim/scripts/Act 2

All amazon polly files can located on Github along with the source code used to generate them.

**Node Diagram**

  
**Code:**

**Record.py**

#!/usr/bin/env python

import pyaudio

import wave

import sys

import os

import rospy

from std\_msgs.msg import String

from service\_tut.srv import \*

CHUNK = 512

FORMAT = pyaudio.paInt16

CHANNELS = 1

RATE = 44100

RECORD\_SECONDS = 5

WAVE\_OUTPUT\_FILENAME = "output.wav"

# file path for the .wav file

homedir = os.environ['HOME']

filepath = homedir + "/dim\_ws/src/service\_tut/scripts"

user\_input = os.path.join(filepath, WAVE\_OUTPUT\_FILENAME)

start = rospy.Publisher('start',String,queue\_size = 10)

def record():

p = pyaudio.PyAudio()

stream = p.open(format=FORMAT,

channels=CHANNELS,

rate=RATE,

input=True,

frames\_per\_buffer=CHUNK)

print("\* recording")

frames = []

for i in range(0, int(RATE / CHUNK \* RECORD\_SECONDS)):

data = stream.read(CHUNK)

frames.append(data)

print("\* done recording")

stream.stop\_stream()

stream.close()

p.terminate()

wf = wave.open(WAVE\_OUTPUT\_FILENAME, 'wb')

wf.setnchannels(CHANNELS)

wf.setsampwidth(p.get\_sample\_size(FORMAT))

wf.setframerate(RATE)

wf.writeframes(b''.join(frames))

wf.close()

start.publish("start")

def init\_record():

p = pyaudio.PyAudio()

stream = p.open(format=FORMAT,

channels=CHANNELS,

rate=RATE,

input=True,

frames\_per\_buffer=CHUNK)

print("\*init recording")

frames = []

for i in range(0, int(RATE / CHUNK \* RECORD\_SECONDS)):

data = stream.read(CHUNK)

frames.append(data)

print("\* done init recording")

stream.stop\_stream()

stream.close()

p.terminate()

wf = wave.open(WAVE\_OUTPUT\_FILENAME, 'wb')

wf.setnchannels(CHANNELS)

wf.setsampwidth(p.get\_sample\_size(FORMAT))

wf.setframerate(RATE)

wf.writeframes(b''.join(frames))

wf.close()

start.publish("start")

def end\_callback(data):

record()

def recorder():

rospy.init\_node("record",anonymous = True)

print("Record node init")

init\_record()

rospy.Subscriber('end',String,end\_callback)

rospy.spin()

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

try:

recorder()

except KeyboardInterrupt:

Pass

**Dialogflow.py:**

#!/usr/bin/env python

import dialogflow\_v2

import os

import wave

from subprocess import call

import rospy

from std\_msgs.msg import String

global WAVE\_OUTPUT\_FILENAME

polly = rospy.Publisher('polly\_listen',String,queue\_size = 10)

drum = rospy.Publisher('drum',String,queue\_size = 10)

face = rospy.Publisher('face',String,queue\_size = 10)

def detect\_intent\_audio(project\_id, session\_id, audio\_file\_path,

language\_code):

session\_client = dialogflow\_v2.SessionsClient()

audio\_encoding = dialogflow\_v2.enums.AudioEncoding.AUDIO\_ENCODING\_LINEAR\_16

sample\_rate\_hertz = 44100

session = session\_client.session\_path(project\_id, session\_id)

with open(audio\_file\_path, 'rb') as audio\_file:

input\_audio = audio\_file.read()

audio\_config = dialogflow\_v2.types.InputAudioConfig(

audio\_encoding=audio\_encoding, language\_code=language\_code,

sample\_rate\_hertz=sample\_rate\_hertz)

query\_input = dialogflow\_v2.types.QueryInput(audio\_config=audio\_config)

response = session\_client.detect\_intent(

session=session, query\_input=query\_input,

input\_audio=input\_audio)

return response.query\_result.fulfillment\_text

# file path for the .wav file

def init\_callback(data):

WAVE\_OUTPUT\_FILENAME = "output.wav"

print("init callback")

homedir = os.environ['HOME']

filepath = homedir + "/dim\_ws/"

user\_input = os.path.join(filepath, WAVE\_OUTPUT\_FILENAME)

result = detect\_intent\_audio("dimbot-309c3", "1-1-1-1-1", user\_input, 'en-US')

print result

polly.publish(result)

drum.publish(result)

face.publish(result)

def processor():

rospy.init\_node("dialogflow",anonymous = True)

print("dialogflow init")

rospy.Subscriber('start',String,init\_callback)

rospy.spin()

if \_\_name\_\_ == "\_\_main\_\_":

try:

processor()

except KeyboardInterrupt:

pass

**Polly.py:**

#!/usr/bin/env python

import boto3

import rospy

#import actionlib

from std\_msgs.msg import String, Bool

import time

from pygame import mixer

Text\_Input=""

p=""

end = rospy.Publisher('end',String,queue\_size = 10)

def Speak\_callback(data):

global Text

print("Session Created")

polly\_client = boto3.Session(

aws\_access\_key\_id="AKIAIYIA4XOXYYJNSJLA",

aws\_secret\_access\_key="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",

region\_name='us-west-2').client('polly')

print("Waiting for Callback")

Text\_Input=data.data

response = polly\_client.synthesize\_speech(VoiceId='Matthew',

OutputFormat='mp3',

#Text = 'Robotics Sample Text.')

Text = Text\_Input)

file = open('speech.mp3', 'w')

file.write(response['AudioStream'].read())

file.close()

time.sleep(2)

mixer.init()

mixer.music.load('/home/pi/dim\_ws/speech.mp3')

mixer.music.play()

print("File Played")

time.sleep(2)

end.publish("end")

print("end publish")

def polly():

global Text

# Initializing the ROS node "polly\_speech"

rospy.init\_node("polly", anonymous=True)

print("polly node init")

# Creating Subscriber topics for Listen

rospy.Subscriber("polly\_listen",String,Speak\_callback)

rospy.spin()

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

try:

polly()

except rospy.ROSInterruptexception:

pass

**Body.py:**

#!/usr/bin/env python

# Simple demo of of the PCA9685 PWM servo/LED controller library.

# This will move channel 0 from min to max position repeatedly.

# Author: Tony DiCola

# License: Public Domain

from \_\_future\_\_ import division

import time

import rospy

from std\_msgs.msg import String

from std\_msgs.msg import Int32

# Import the PCA9685 module.

import Adafruit\_PCA9685

#import cv2

import sys

import os

# Initialise the PCA9685 using the default address (0x40).

pwm = Adafruit\_PCA9685.PCA9685()

##Pulse length to degrees

degree\_0 = 102

degree\_30 = 171

degree\_40 = 194

degree\_45 = 206

degree\_50 = 220

degree\_60 = 240

degree\_70 = 263

degree\_80 = 286

degree\_85 = 297

degree\_90 = 310

degree\_100 = 333

degree\_110 = 356

degree\_120 = 379

degree\_130 = 400

degree\_135 = 414

degree\_140 = 430

degree\_150 = 448

degree\_160 = 471

degree\_180 = 505

#pwm channel number on PWM Driver

pwm\_channel\_0 = 0 #Left Eyebrow

pwm\_channel\_1 = 1 #Right Eyebrow

pwm\_channel\_2 = 2 #Left Eye Lid

pwm\_channel\_3 = 3 #Right Eye Lid

pwm\_channel\_4 = 4 #Left and Right Horizontal

pwm\_channel\_5 = 5 #Left and Right Vertical

pwm\_channel\_6 = 6 #Mouth

pwm\_channel\_7 = 7 #Left Shoulder joint

pwm\_channel\_8 = 8 #Right Shoulder joint

pwm\_channel\_9 = 9 #Left Arm\_side ways

pwm\_channel\_10 = 10 #Right Arm\_side ways

pwm\_channel\_11 = 11 #Left Elbow

pwm\_channel\_12 = 12 #right Elbow

pwm\_channel\_13 = 13

pwm\_channel\_14 = 14

pwm\_channel\_15 = 15

class Face:

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

print ("Face init")

def Right\_Eyebrow(self,channel,degree):

pwm.set\_pwm(channel,0,degree)

def Left\_Eyebrow(self,channel,degree):

pwm.set\_pwm(channel,0,degree)

def Eye\_Center(self,channel,degree):

#110

pwm.set\_pwm(channel,0,degree)

def Mouth(self,channel,degree):

#60 open | 0 close

pwm.set\_pwm(channel,0,degree)

def Left\_Eye\_Lid(self,channel,degree):

#150 close | 90 open

pwm.set\_pwm(channel,0,degree)

def Right\_Eye\_Lid(self,channel,degree):

#60 close | 120 open

pwm.set\_pwm(channel,0,degree)

def Eye\_Vertical(self,channel,degree):

#60 up | 100 down

pwm.set\_pwm(channel,0,degree)

def Eye\_Horizontal(self,channel,degree):

#160 left | 80 Right

pwm.set\_pwm(channel,0,degree)

def Face\_Reset(self):

self.Right\_Eyebrow(pwm\_channel\_1,degree\_120)

self.Left\_Eyebrow(pwm\_channel\_0,degree\_120)

self.Mouth(pwm\_channel\_6,degree\_0)

self.Left\_Eye\_Lid(pwm\_channel\_2,degree\_150)

self.Right\_Eye\_Lid(pwm\_channel\_3,degree\_60)

self.Eye\_Vertical(pwm\_channel\_5,degree\_100)

self.Eye\_Horizontal(pwm\_channel\_4,degree\_120)

print("Reset is done")

def Excited(self):

Robo\_face.Mouth(pwm\_channel\_6,degree\_60)

Robo\_face.Left\_Eyebrow(pwm\_channel\_0,degree\_135)

Robo\_face.Right\_Eyebrow(pwm\_channel\_1,degree\_135)

Robo\_face.Left\_Eye\_Lid(pwm\_channel\_2,degree\_90)

Robo\_face.Right\_Eye\_Lid(pwm\_channel\_3,degree\_120)

os.system('flite -voice rms -t "Hey, I am Excited"')

time.sleep(0.5)

def Very\_happy(self):

Robo\_face.Mouth(pwm\_channel\_6,degree\_60)

Robo\_face.Eye\_Vertical(pwm\_channel\_5,degree\_100)

Robo\_face.Left\_Eyebrow(pwm\_channel\_0,degree\_135)

Robo\_face.Right\_Eyebrow(pwm\_channel\_1,degree\_135)

Robo\_face.Left\_Eye\_Lid(pwm\_channel\_2,degree\_140)

Robo\_face.Right\_Eye\_Lid(pwm\_channel\_3,degree\_70)

Robo\_face.Eye\_Vertical(pwm\_channel\_5,degree\_120)

os.system('flite -voice rms -t "I am Very Happy Right now"')

time.sleep(0.5)

def Sleepy(self):

Robo\_face.Mouth(pwm\_channel\_6,degree\_0)

Robo\_face.Left\_Eyebrow(pwm\_channel\_0,degree\_135)

Robo\_face.Right\_Eyebrow(pwm\_channel\_1,degree\_135)

Robo\_face.Left\_Eye\_Lid(pwm\_channel\_2,degree\_140)

Robo\_face.Right\_Eye\_Lid(pwm\_channel\_3,degree\_70)

Robo\_face.Eye\_Vertical(pwm\_channel\_5,degree\_80)

os.system('flite -voice rms -t "I am feeling sleepy"')

time.sleep(0.5)

def Sleep(self):

Robo\_face.Mouth(pwm\_channel\_6,degree\_0)

Robo\_face.Left\_Eyebrow(pwm\_channel\_0,degree\_120)

Robo\_face.Right\_Eyebrow(pwm\_channel\_1,degree\_120)

Robo\_face.Left\_Eye\_Lid(pwm\_channel\_2,degree\_150)

Robo\_face.Right\_Eye\_Lid(pwm\_channel\_3,degree\_60)

Robo\_face.Eye\_Vertical(pwm\_channel\_5,degree\_60)

os.system('aplay snore.wav')

time.sleep(0.5)

def Sad(self):

Robo\_face.Mouth(pwm\_channel\_6,degree\_45)

Robo\_face.Left\_Eyebrow(pwm\_channel\_0,degree\_140)

Robo\_face.Right\_Eyebrow(pwm\_channel\_1,degree\_140)

Robo\_face.Left\_Eye\_Lid(pwm\_channel\_2,degree\_130)

Robo\_face.Right\_Eye\_Lid(pwm\_channel\_3,degree\_85)

Robo\_face.Eye\_Vertical(pwm\_channel\_5,degree\_100)

os.system('flite -voice rms -t "I am sad man and really really very upset"')

time.sleep(0.5)

def Suspicious(self):

Robo\_face.Mouth(pwm\_channel\_6,degree\_0)

Robo\_face.Left\_Eyebrow(pwm\_channel\_0,degree\_100)

Robo\_face.Right\_Eyebrow(pwm\_channel\_1,degree\_100)

Robo\_face.Left\_Eye\_Lid(pwm\_channel\_2,degree\_135)

Robo\_face.Right\_Eye\_Lid(pwm\_channel\_3,degree\_80)

Robo\_face.Eye\_Vertical(pwm\_channel\_5,degree\_100)

os.system('flite -voice rms -t "Whose down there, I am suspicious"')

time.sleep(0.5)

def Angry(self):

Robo\_face.Mouth(pwm\_channel\_6,degree\_45)

Robo\_face.Left\_Eyebrow(pwm\_channel\_0,degree\_90)

Robo\_face.Right\_Eyebrow(pwm\_channel\_1,degree\_90)

Robo\_face.Left\_Eye\_Lid(pwm\_channel\_2,degree\_100)

Robo\_face.Right\_Eye\_Lid(pwm\_channel\_3,degree\_110)

Robo\_face.Eye\_Vertical(pwm\_channel\_5,degree\_100)

os.system('flite -voice rms -t "Angry about What u did to me"')

time.sleep(0.5)

def Winky(self):

i = 0

Robo\_face.Mouth(pwm\_channel\_6,degree\_60)

Robo\_face.Left\_Eyebrow(pwm\_channel\_0,degree\_135)

Robo\_face.Right\_Eyebrow(pwm\_channel\_1,degree\_135)

Robo\_face.Left\_Eye\_Lid(pwm\_channel\_2,degree\_90)

Robo\_face.Right\_Eye\_Lid(pwm\_channel\_3,degree\_120)

os.system('flite -voice rms -t "how are u doing"')

while(i < 3):

Robo\_face.Right\_Eye\_Lid(pwm\_channel\_3,degree\_60)

Robo\_face.Right\_Eyebrow(pwm\_channel\_1,degree\_90)

time.sleep(0.5)

Robo\_face.Right\_Eye\_Lid(pwm\_channel\_3,degree\_120)

Robo\_face.Right\_Eyebrow(pwm\_channel\_1,degree\_135)

time.sleep(0.5)

i = i + 1

time.sleep(0.5)

class Arm:

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

print ("Arm init")

def right\_shoulder(self,channel,degree):

pwm.set\_pwm(channel,0,degree)

def left\_shoulder(self,channel,degree):

pwm.set\_pwm(channel,0,degree)

def right\_biceps(self,channel,degree):

pwm.set\_pwm(channel,0,degree)

def left\_biceps(self,channel,degree):

pwm.set\_pwm(channel,0,degree)

def right\_hand(self,channel,degree):

pwm.set\_pwm(channel,0,degree)

def left\_hand(self,channel,degree):

pwm.set\_pwm(channel,0,degree)

def normal(self,channel,degree):

pwm.set\_pwm(channel,0,degree)

def Arm\_Reset(self):

Robo\_arm.right\_hand(pwm\_channel\_12,degree\_150)

Robo\_arm.left\_hand(pwm\_channel\_11,degree\_60)

time.sleep(1)

Robo\_arm.left\_biceps(pwm\_channel\_9,degree\_90)

Robo\_arm.right\_biceps(pwm\_channel\_10,degree\_180)

time.sleep(1)

Robo\_arm.left\_biceps(pwm\_channel\_9,degree\_90)

Robo\_arm.right\_shoulder(pwm\_channel\_8,degree\_0)

Robo\_arm.left\_shoulder(pwm\_channel\_7,degree\_90)

Robo\_arm.right\_hand(pwm\_channel\_12,degree\_30)

Robo\_arm.left\_hand(pwm\_channel\_11,degree\_180)

Robo\_arm.right\_biceps(pwm\_channel\_10,degree\_180)

def Arm\_Initial(self):

Robo\_arm.right\_hand(pwm\_channel\_12,degree\_150)

Robo\_arm.left\_hand(pwm\_channel\_11,degree\_60)

time.sleep(1)

Robo\_arm.right\_shoulder(pwm\_channel\_8,degree\_0)

Robo\_arm.left\_shoulder(pwm\_channel\_7,degree\_90)

time.sleep(1)

Robo\_arm.left\_biceps(pwm\_channel\_9,degree\_120)

Robo\_arm.right\_biceps(pwm\_channel\_10,degree\_130)

time.sleep(1)

def Drum(self):

i = 0

os.system('flite -voice rms -t "Hey, this is first pattern i am composing"')

while i < 15:

Robo\_arm.right\_hand(pwm\_channel\_12,degree\_30)

Robo\_arm.left\_hand(pwm\_channel\_11,degree\_150)

Robo\_arm.left\_biceps(pwm\_channel\_9,degree\_120)

time.sleep(0.5)

Robo\_arm.right\_hand(pwm\_channel\_12,degree\_60)

time.sleep(0.2)

Robo\_arm.right\_hand(pwm\_channel\_12,degree\_30)

Robo\_arm.left\_hand(pwm\_channel\_11,degree\_180)

time.sleep(0.2)

Robo\_arm.right\_hand(pwm\_channel\_12,degree\_60)

Robo\_arm.left\_hand(pwm\_channel\_11,degree\_150)

Robo\_arm.left\_biceps(pwm\_channel\_9,degree\_120)

time.sleep(0.2)

i = i + 1

def Drum1(self):

i = 0

j = 0

k = 0

while k < 3:

Robo\_arm.left\_hand(pwm\_channel\_11,degree\_150)

while i < 10:

Robo\_arm.right\_hand(pwm\_channel\_12,degree\_30)

Robo\_arm.left\_hand(pwm\_channel\_11,degree\_150)

time.sleep(0.2)

Robo\_arm.right\_hand(pwm\_channel\_12,degree\_60)

time.sleep(0.3)

i = i + 1

Robo\_arm.right\_hand(pwm\_channel\_12,degree\_60)

while j < 5:

Robo\_arm.left\_hand(pwm\_channel\_11,degree\_180)

time.sleep(0.5)

Robo\_arm.left\_hand(pwm\_channel\_11,degree\_130)

time.sleep(1)

j = j + 1

k = k + 1

def Drum2(self):

os.system('flite -voice rms -t "Just look at this one"')

i = 0

j = 0

k = 0

while k < 3:

Robo\_arm.left\_hand(pwm\_channel\_11,degree\_150)

while i < 10:

Robo\_arm.right\_hand(pwm\_channel\_12,degree\_30)

Robo\_arm.left\_hand(pwm\_channel\_11,degree\_150)

time.sleep(0.2)

Robo\_arm.right\_hand(pwm\_channel\_12,degree\_60)

time.sleep(0.1)

i = i + 1

Robo\_arm.right\_hand(pwm\_channel\_12,degree\_60)

while j < 10:

Robo\_arm.left\_hand(pwm\_channel\_11,degree\_180)

time.sleep(0.7)

Robo\_arm.left\_hand(pwm\_channel\_11,degree\_150)

time.sleep(0.7)

j = j + 1

k = k + 1

def Drum3(self):

i = 0

while i < 10:

Robo\_arm.left\_hand(pwm\_channel\_11,degree\_180)

time.sleep(0.1)

Robo\_arm.right\_hand(pwm\_channel\_12,degree\_60)

time.sleep(0.2)

Robo\_arm.right\_hand(pwm\_channel\_12,degree\_30)

time.sleep(0.1)

Robo\_arm.left\_hand(pwm\_channel\_11,degree\_150)

time.sleep(0.5)

i = i + 1

def Drum4(self):

os.system('flite -voice rms -t "I bet this is Awesome"')

i = 0

while i < 15:

Robo\_arm.left\_hand(pwm\_channel\_11,degree\_150)

#Robo\_arm.left\_biceps(pwm\_channel\_9,degree\_150)

Robo\_arm.right\_hand(pwm\_channel\_12,degree\_60)

time.sleep(0.2)

Robo\_arm.right\_hand(pwm\_channel\_12,degree\_30)

Robo\_arm.left\_biceps(pwm\_channel\_9,degree\_120)

time.sleep(0.2)

Robo\_arm.left\_hand(pwm\_channel\_11,degree\_180)

time.sleep(0.2)

i = i + 1

# Helper function to make setting a servo pulse width simpler.

def set\_servo\_pulse(channel, pulse):

pulse\_length = 1000000 # 1,000,000 us per second

pulse\_length //= 60 # 60 Hz

print('{0}us per period'.format(pulse\_length))

pulse\_length //= 4096 # 12 bits of resolution

print('{0}us per bit'.format(pulse\_length))

pulse \*= 1000

pulse //= pulse\_length

pwm.set\_pwm(channel, 0, pulse)

# Set frequency to 50hz, good for servos.

pwm.set\_pwm\_freq(50)

print('Moving servo on channel 0, press Ctrl-C to quit...')

Robo\_arm = Arm()

Robo\_face = Face()

def music\_callback(data):

select = data.data

if select == "drum 1":

Robo\_arm.Drum1()

elif select == "drum 2":

Robo\_arm.Drum2()

elif select == "drum 3":

Robo\_arm.Drum3()

elif select == "drum 4":

Robo\_arm.Drum4()

print("arm working")

def face\_callback(data):

select = data.data

if select == "excited":

Robo\_face.Excited()

elif select == "winky":

Robo\_face.Winky()

elif select == "sleepy":

Robo\_face.Sleepy()

elif select == "sad":

Robo\_face.Sad()

elif select == "sleep":

Robo\_face.Sleep()

elif select == "suspicious":

Robo\_face.Suspicious()

elif select == "angry":

Robo\_face.Angry()

elif select == "veryhappy":

Robo\_face.Very\_happy()

print("face working")

def test():

# Create Vision node

rospy.init\_node("body", anonymous = True)

print("Body node init")

rospy.Subscriber('drum', String, music\_callback)

rospy.Subscriber('face', String, face\_callback)

rospy.spin()

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

try:

Robo\_arm.Arm\_Reset()

Robo\_face.Face\_Reset()

time.sleep(3)

Robo\_arm.Arm\_Initial()

time.sleep(5)

test()

except rospy.ROSInterruptException:

pass