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.

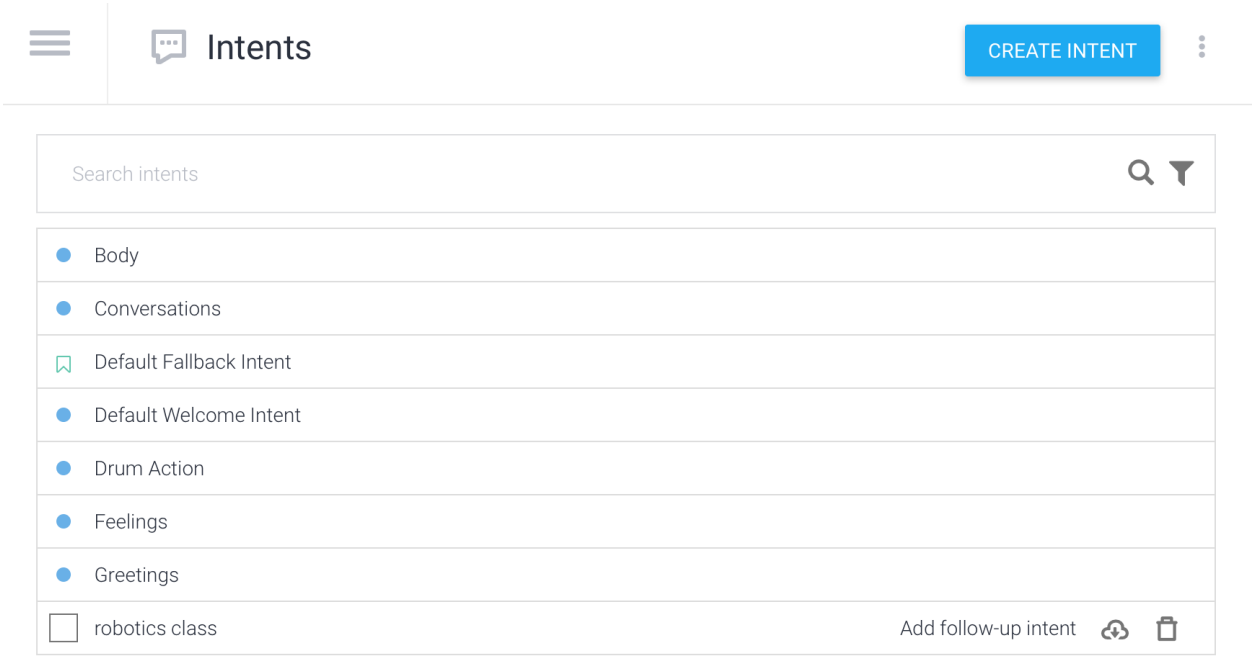
Google Dialogflow

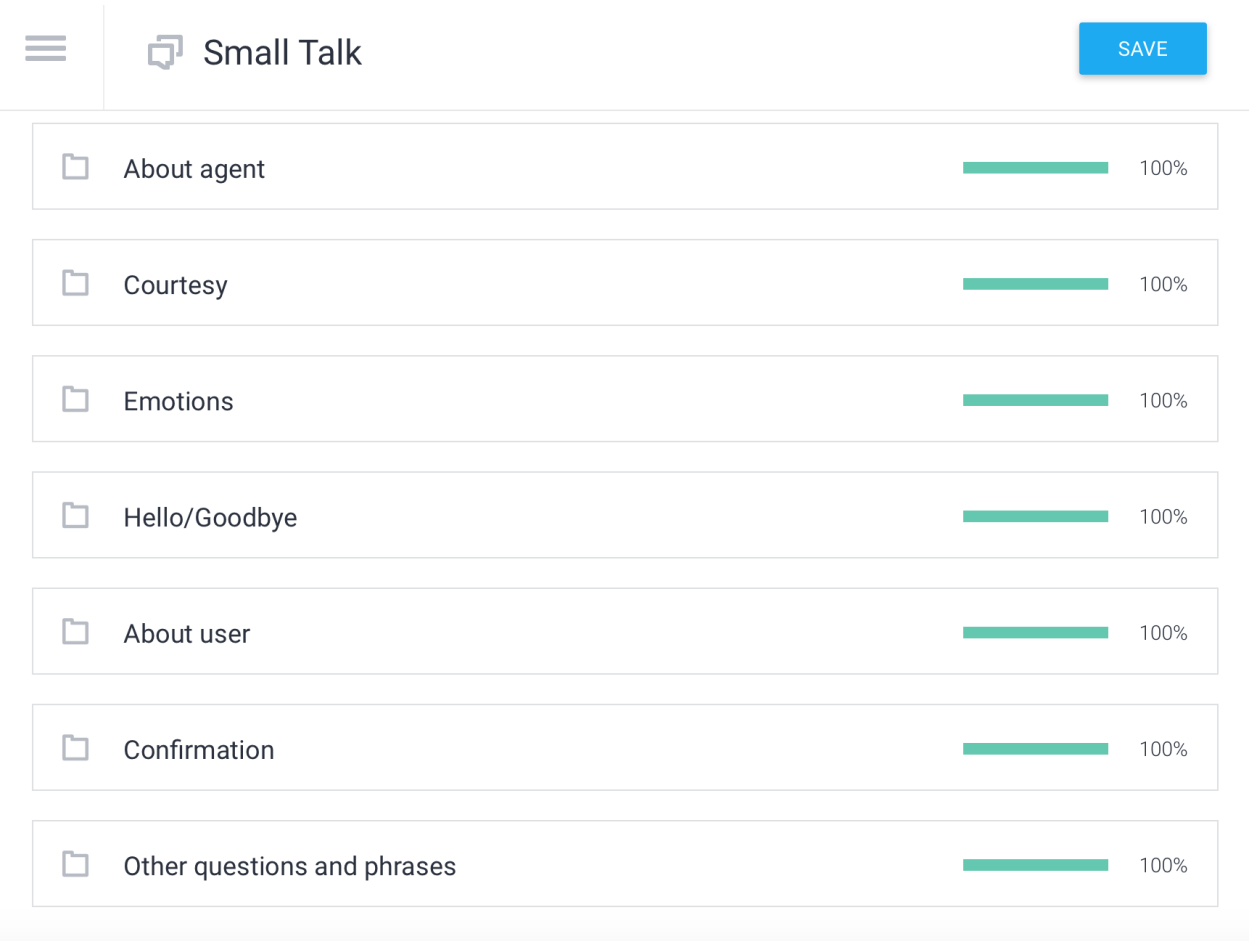
In Google Dialogflow, the first step is to create an Agent (in our case our agent named Dimbot). In the Intents section where it is used for defining a specific action that the user can invoke by using one of the defined terms in the Dialogflow console. Where in our Dimbot has the following intents:

- 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

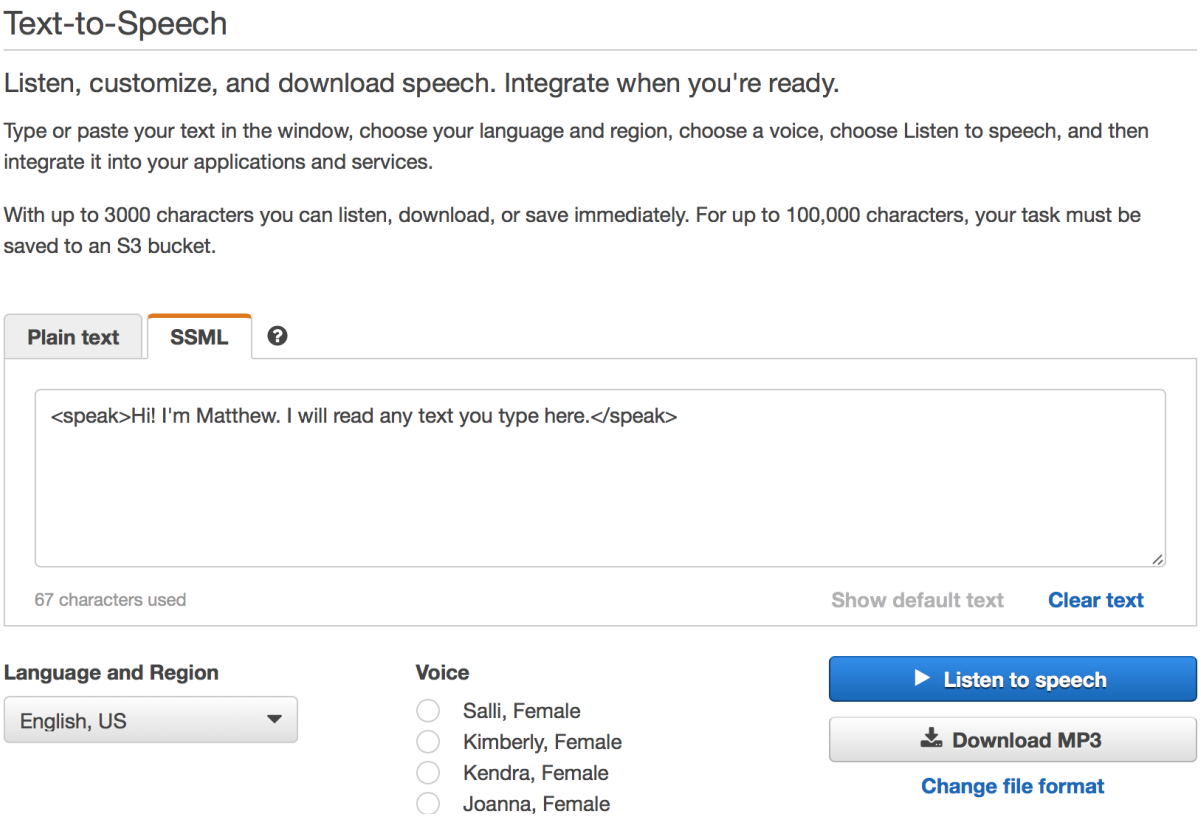
Create an amazon poly accounts

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

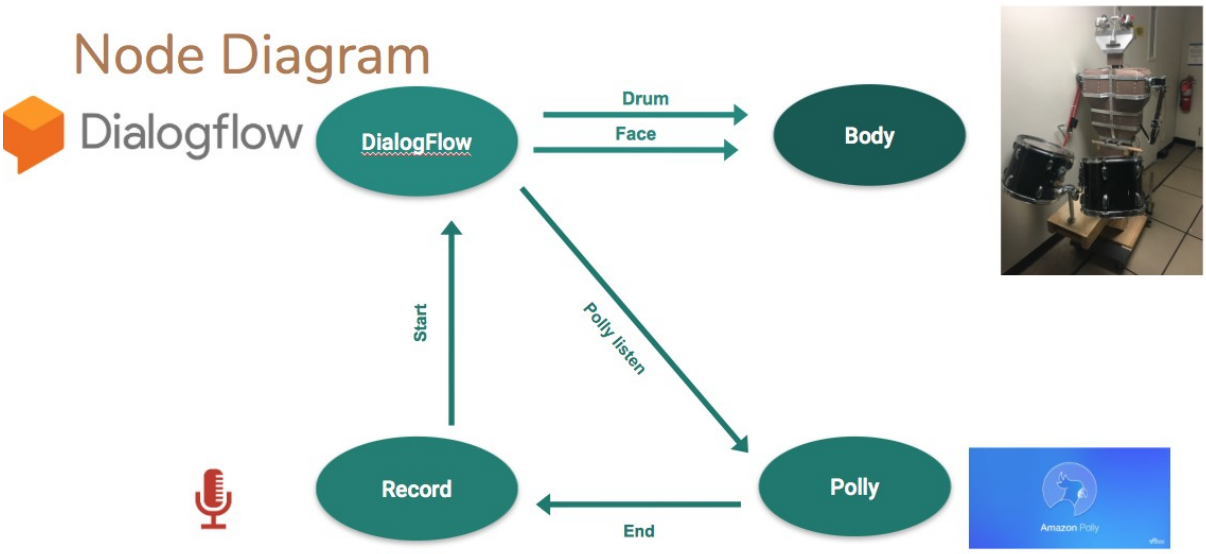
Amazon Polly Theater Recordings:

/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
```