

Turtlebot2 Einstein

Project 2

ECE 478

**Phong Nguyen,**

**Ali Boshehri**



# 

[**Objective**](#_j5l9hfyw72n9) **3**

[**Platform**](#_7njjd83myyo1) **3**

[**Before we start**](#_iyl10qgwrpip) **3**

[**Packages**](#_8bpugfyvuhgq) **4**

[**Part 1: 3 robots talking to each others**](#_6zde4wkd26o8) **4**

[Suggest Schematic and information](#_uuq12kx4suf1) 4

[Code for Part 1](#_vmhql6u3i6o6) 5

[Optional Schematic and information](#_8iy1wc7ner14) 7

[Code for Optional part 1](#_me8uyz8bxie7) 7

[**Part 2: Robot respond to user speak**](#_azy0r6ccmwxe) **10**

[Schematic and information](#_7i1xr63sdqdo) 10

[Code for part 2](#_4wigij4j4eib) 11

[**Challenges**](#_5z73l1wc1prf) **18**

[**Future ideas and improvements**](#_yem1asom0diw) **18**

# 

# Objective

* Implement speech recognition into the robot by using Google Dialogflow.
  + Be able to have a small talk with the robot.
* Implement speech synthesis into your robot by using Amazon Polly
  + We chose German accent to fit Einstein’s character.

# 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

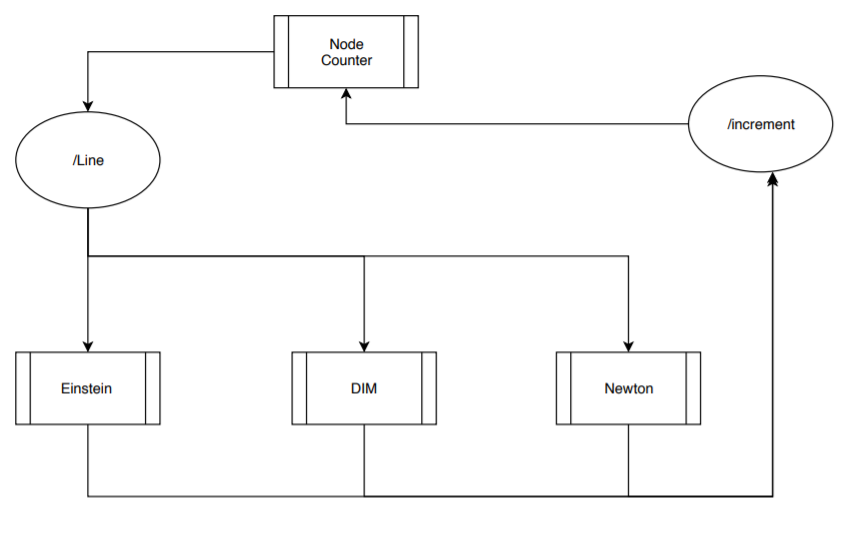
The contents in this document is not original. We documented steps and information from various sources (e.g., tutorials and setup) along the way as we work for the demos. This is mainly for educational purposes not beneficial. Throughout the project, we have learned and worked with the ROS programming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

# Packages

1. Einstein
2. my\_dynamixel \_Tutorial
3. Resbot
4. Speakbot

# Part 1: 3 robots talking to each others

## Suggest Schematic and information



Three robots are talking to each others. In this schematic, we can see there are 3 nodes name Einstein, DIM, and Newton. The counter publish topic /line once. Then if one of the node has for a script audio file for that particular line then start playing the file otherwise it won’t do anything. When the robot play the audio file it will automatically publish to topic /increment. That is when the counter node increment and update new value and publish to topic /line.

This code is really efficient when trying to synchronous timing between the 3 robots and it is short and it is also a neat code.

## Code for Part 1

The code nodes Einstein, DIM, and Newton have similar format.

**Einstein.py**

**#!/usr/bin/env python**

**'''**

**says a line if the line exists for the robot.**

**Then publishes increment to indicate its done.**

**'''**

**import rospy**

**import wave**

**import os**

**import sys**

**from subprocess import Popen**

**from std\_msgs.msg import Int32**

**from std\_msgs.msg import String**

**from playsound import playsound**

**arm = rospy.Publisher('/arm\_act', Int32, queue\_size=1)**

**#lines shared with dim**

**def lineCallback(data):**

**#sharedLines = [0,0]**

**line = data.data**

**if(os.path.isfile("/home/myturtlebot/catkin\_ws/src/einstein/play1/"+ str(line) +".wav")):**

**playsound("/home/myturtlebot/catkin\_ws/src/einstein/play1/"+ str(line) +".wav")**

**return**

**rospy.init\_node("Einstein")**

**increment = rospy.Publisher('/increment', Int32, queue\_size=1)**

**rospy.Subscriber("/lines",Int32,lineCallback)**

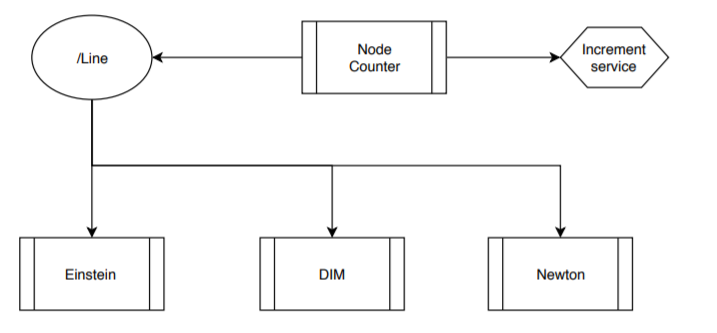
**rospy.spin()**

**Counter.py**

**#!/usr/bin/env python  
  
'''  
increments lines for play  
'''  
  
  
import rospy  
  
from std\_msgs.msg import Int32  
  
  
def incrementCallback(data):  
 increment = data.data  
 increment = increment + 1  
 line\_num.publish(increment)  
 return  
  
rospy.init\_node("Counter")  
line\_num = rospy.Publisher('/lines', Int32, queue\_size=1)  
rospy.Subscriber("/increment",Int32,incrementCallback)  
  
rospy.spin()**

## 

## Optional Schematic and information

This schematic is our group suggestion. But our demo was based on Suggest Schematic. Node counter works differently this time. It will constantly publish the /line numbers for every node Einstein, DIM, and Newton to subscribe to. Every Times, when the node its existing play audio file. It will call increment service, then the node counter will stop publish and servicing that request service from the node. After it servicing it will increment to line and publish a new line value.

***Challenge when coding:***

Timing and synchronization between the bots.

## Code for Optional part 1

**#Line\_counter.py**

**#!/usr/bin/env python**

**import rospy**

**from std\_msgs.msg import Int32**

**from std\_msgs.msg import String**

**from line\_counter.srv import myservice**

**num = 0**

**pub = rospy.Publisher("line", Int32, queue\_size=10)**

**def handle\_service(data):**

**global num**

**num = num + data.req**

**return num**

**def counter():**

**global num**

**#initilize the node**

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

**#set the publishing rate**

**r = rospy.Rate(0.2) # 10Hz**

**#counter loop**

**s=rospy.Service('increment',myservice,handle\_service)**

**#publish values**

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

**pub.publish(num)**

**rospy.loginfo(num)**

**r.sleep()**

**rospy.spin()**

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

**try:**

**counter()**

**except rospy.ROSInterruptException():**

**pass**

=======================================================================

Robot1.py

**#!/usr/bin/env python**

**# Import Libraries**

**import rospy**

**from std\_msgs.msg import Int32**

**from std\_msgs.msg import String**

**from line\_counter.srv import \***

**import time**

**handle=rospy.ServiceProxy('increment',myservice)**

**x=0**

**def callback(data):**

**global x**

**#get data**

**x = data.data**

**rospy.loginfo('data from sub %d',x)**

**r = rospy.Rate(0.2)**

**#process the line**

**if(os.path.isfile("/home/myturtlebot/catkin\_ws/src/einstein/play1/"+str(line) +".wav")):**

**playsound("/home/myturtlebot/catkin\_ws/src/einstein/play1/"+ str(line) +".wav")**

**x=x+1**

**x = handle(x).res**

**r.sleep()**

**#after process service increment**

**def robot():**

**#initilize the node**

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

**#subscribe**

**rospy.Subscriber('/line', Int32, callback)**

**#keep spinning until done**

**rospy.spin()**

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

**try:**

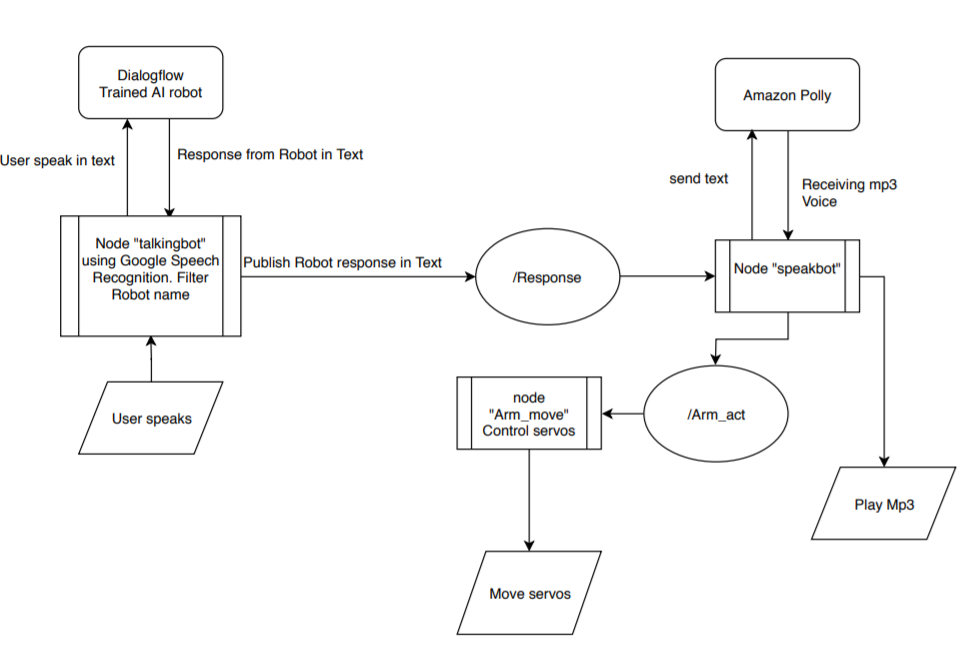
**robot()**

**except rospy.ROSInterruptException():**

**pass**

# Part 2: Robot respond to user speak

## Schematic and information



This schematic, we have used 3 clouds services: Google speech recognition, Dialogflow, and Amazon Polly. Consists of 2 main nodes : talkingbot and speakbot.

Talkingbot node: getting user speaking input convert speech into text by Google speech recognition. It will constant listening to its name. You can name it any thing that you want like Andrew, Albert, Einstein , etc,.. . when it hears its name being, the node will process sending the user text into Dialogflow to getting response from the AI. This is efficient because it can filter out the name when it take the text from user speak, and only upload the truncate text without its name. Dialogflow does not get confusing by name. Also, when we getting to speakbot node it does not listen to itself.

Speakbot node: just simply stay there and listen to anything on the /response topic. When it gets the text from /Response topic it will do several things. Uploading the text to Amazon polly and getting mp3 file from Amazon polly. Then play that audio file. At the same time, it will publish to topic /Arm\_act a value for Arm\_move node to move servos, some gestures.

***Challenge when coding:***

Tricky part is try to get talkingbot node to work with dialogflow. Dialogflow has some strictly authentication. You may find it hard to set it up.

## Code for part 2

Dialogflow.py

**#!/usr/bin/env python**

**import pdb**

**#debug**

**#pdb.set\_trace()**

**import rospy**

**from std\_msgs.msg import String**

**import speech\_recognition as sr**

**import argparse**

**import uuid**

**import os**

**import dialogflow\_v2 as dialogflow**

**os.environ["GOOGLE\_APPLICATION\_CREDENTIALS"] = '/home/myturtlebot/Downloads/Robotic-5595bb2b581d.json'**

**project\_id = 'robotic-223705'**

**session\_id = str(uuid.uuid4())**

**language\_code='en-US'**

**BOTNAME="John"**

**r = sr.Recognizer()**

**m = sr.Microphone()**

**pub = rospy.Publisher('/response', String, queue\_size=10)**

**def callback(res):**

**#Start to publish the topic /response**

**res = str(res)**

**rospy.loginfo(res)**

**pub.publish(res)**

**def detect\_intent\_texts(texts):**

**global project\_id,session\_id,language\_code**

**#Dialogflow Google process**

**session\_client = dialogflow.SessionsClient()**

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

**for text in texts:**

**text\_input = dialogflow.types.TextInput(text=text, language\_code=language\_code)**

**query\_input = dialogflow.types.QueryInput(text=text\_input)**

**response = session\_client.detect\_intent(session=session, query\_input=query\_input)**

**print('{}\n'.format(response.query\_result.fulfillment\_text))**

**callback(response.query\_result.fulfillment\_text)**

**rospy.init\_node("talkingbot")**

**#prepare the microphone**

**try:**

**print("A moment of silence, please...")**

**while True:**

**with m as source: r.adjust\_for\_ambient\_noise(source)**

**print("Set minimum energy threshold to {}".format(r.energy\_threshold))**

**print("Say something!")**

**with m as source: audio = r.listen(source)**

**print("Got it! Now to recognize it...")**

**value="r"**

**try:**

**# recognize speech using Google Speech Recognition**

**value = r.recognize\_google(audio)**

**print(value)**

**except sr.UnknownValueError:**

**pass**

**except sr.RequestError as e:**

**print("Uh oh! Couldn't request results from Google Speech Recognition service; {0}".format(e))**

**pos=0**

**pos=value.find(BOTNAME)**

**if pos != -1:**

**#filter the bot name**

**value=value.replace(BOTNAME,'')**

**#remove the white space**

**value = value[:pos] + value[(pos+1):]**

**if len(value)>3:**

**#unicode to string**

**texts= str(value)**

**#string to list**

**texts=[texts]**

**print(texts)**

**detect\_intent\_texts(texts)**

**except rospy.ROSInterruptException:**

**pass**

**except KeyboardInterrupt:**

**pass**

**rospy.spin()**

**==========================================================================**

**#Speakbot.py**

**#!/usr/bin/env python**

**import pdb**

**import rospy**

**import boto3**

**import os**

**from playsound import playsound**

**from subprocess import Popen**

**from std\_msgs.msg import String,Int32**

**arm = rospy.Publisher('/arm\_act', Int32, queue\_size=1)**

**polly\_client = boto3.Session(aws\_access\_key\_id='###########',aws\_secret\_access\_key='#################',region\_name='us-west-2').client('polly')**

**def callback(data):**

**response = polly\_client.synthesize\_speech(VoiceId='Matthew',OutputFormat='mp3',Text = data.data)**

**file = open('/home/myturtlebot/catkin\_ws/speech.mp3', 'w')**

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

**file.close()**

**playsound("/home/myturtlebot/catkin\_ws/speech.mp3")**

**r = rospy.Rate(0.2);**

**value=1**

**arm.publish(value)**

**r.sleep()**

**value=0**

**arm.publish(value)**

**r.sleep()**

**rospy.init\_node("speakingbot")**

**rospy.Subscriber("/response",String,callback)**

**rospy.spin()**

B. my\_dynamixel \_Tutorial

This package includes one python script and it’s responsible for moving the upper body of the turtlebot (servos).

1. Amr\_move (code)

**#!/usr/bin/env python**

**import rospy**

**from std\_msgs.msg import String, Int32, Float64**

**from geometry\_msgs.msg import Twist**

**from math import radians**

**from time import sleep**

**arm1 = rospy.Publisher('/tilt\_controller1/command', Float64, queue\_size=1)**

**arm2 = rospy.Publisher('/tilt\_controller2/command', Float64, queue\_size=1)**

**arm3 = rospy.Publisher('/tilt\_controller3/command', Float64, queue\_size=1)**

**arm4 = rospy.Publisher('/tilt\_controller4/command', Float64, queue\_size=1)**

**arm5 = rospy.Publisher('/tilt\_controller5/command', Float64, queue\_size=1)**

**arm6 = rospy.Publisher('/tilt\_controller6/command', Float64, queue\_size=1)**

**arm7 = rospy.Publisher('/tilt\_controller7/command', Float64, queue\_size=1)**

**arm8 = rospy.Publisher('/tilt\_controller8/command', Float64, queue\_size=1)**

**arm9 = rospy.Publisher('/tilt\_controller9/command', Float64, queue\_size=1)**

**arm10 = rospy.Publisher('/tilt\_controller10/command', Float64, queue\_size=1)**

**def callback(data):**

**arm\_val=data.data**

**rospy.loginfo("activate is %d",arm\_val)**

**#movement here**

**r = rospy.Rate(0.8);**

**#initialize arm\_value 1 activate, 0 deactivate**

**if arm\_val==0 :**

**rospy.loginfo("Normal mode")**

**#Normal mode**

**arm1\_pos = -1**

**arm2\_pos = -0.5**

**arm3\_pos = 0**

**arm4\_pos = 0**

**arm5\_pos = 0.5**

**arm6\_pos = 0.5**

**arm7\_pos = 1.5**

**arm8\_pos = -1**

**arm9\_pos = 0**

**arm10\_pos = 0**

**arm1.publish(arm1\_pos)**

**arm2.publish(arm2\_pos)**

**arm3.publish(arm3\_pos)**

**arm4.publish(arm4\_pos)**

**arm5.publish(arm5\_pos)**

**arm6.publish(arm6\_pos)**

**arm7.publish(arm7\_pos)**

**arm8.publish(arm8\_pos)**

**arm9.publish(arm9\_pos)**

**arm10.publish(arm10\_pos)**

**elif arm\_val==1 :**

**rospy.loginfo("waving")**

**arm1\_pos = 1.5**

**arm5\_pos = 0**

**arm6\_pos = 1**

**arm7\_pos = -1**

**arm10\_pos = 1**

**arm1.publish(arm1\_pos)**

**arm5.publish(arm5\_pos)**

**arm6.publish(arm6\_pos)**

**arm7.publish(arm7\_pos)**

**rospy.loginfo("activate is %d",arm\_val)**

**for i in range (1,3):**

**#waving here**

**rospy.loginfo("waving 1")**

**arm2\_pos = 0.5 #motion 0.5 -> -0.5**

**arm2.publish(arm2\_pos)**

**arm8\_pos = -0.5 #motion -0.5 -> -1**

**arm8.publish(arm8\_pos)**

**r.sleep()**

**rospy.loginfo("waving 2")**

**arm2\_pos = -0.5**

**arm2.publish(arm2\_pos)**

**arm8\_pos = -1**

**arm8.publish(arm8\_pos)**

**arm10.publish(arm10\_pos)**

**arm\_val=0**

**rospy.loginfo("arm\_val: %d",arm\_val)**

**r.sleep()**

**rospy.init\_node('arm\_move',anonymous=True)**

**rospy.Subscriber("/arm\_act",Int32, callback)**

**rospy.spin()**

# Challenges

We have already worked with ROS, so that wasn’t an issue, but having the robot to be synchronous with two other robots was a challenge. We used Intel Joule which was a better than raspberry pi at performance. The play could have been improved a lot better if all teams used Intel Joule. Unfortunately, The model has stopped production.

# Future ideas and improvements

There are many different ideas that can be implemented into training the robot to respond different sentences. DialogFlow speech can be expanded even more and the robot can have many different subject to talk about and respond to the user. Amazon Polly is the same, it has a great potential to add more speech to the robot. And we can add more upper body gestures and movement of base listen to user command.