Gesture Controlled Spotify Player

By Dillon MacDonald, for EEC 172

October 29, 2021

***Abstract*—This report is intended to document the creation of a gesture-controlled Spotify player. The project involves the use of a APDS-9960 gesture sensor from Sparkfun, a CC3200 TI microcontroller, the Spotify API, and Amazon Web Services.**

***Index Terms*—IEEE, IEEEtran, journal, LaTeX, paper, template.**

I. Introduction

The idea for this project came after completing lab 4. After seeing that the CC3200 could connect to the internet and send POST and GET http requests, I had the idea that this could possibly work for the Spotify API. On the project idea handout provided by the instructors there was a project that requested weather data from an external API, and this supported my idea that interacting with the Spotify API from the CC3200 board was very much possible. Implementing gestures came from the exercise machine learning project where certain actions are recognized. Putting these two ideas together, I found the APDS-9960 gesture sensor could read hand motions using I2C. We had used I2C protocol in lab 3 to read data from the on-board accelerometer,

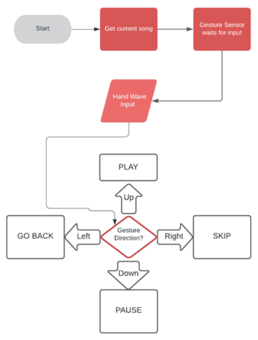


Fig. 1. Diagram of Project

II. Methodology

A. Connecting the APDS-9960 Gesture Sensor to the CC3200

The APDS-9960 has six pins. These pins and their connections to the CC3200 are:

- VL : None
- GND : GND
- VCC : 3.3V
- SDA : P02
- SCL : P01
- INT : P58

VL is an optional pin to power the small LED that indicates if the sensor is receiving power. This pin is connected to VCC by default and does not need it's own signal unless a built in pull up resistor is desoldered. VCC powers the gesture sensor, and connects to the 3.3V pin on the CC3200. The serial data (SDA) pin is what transfers data from the sensor to the microcontroller to be read. This pin connects to P02 on the board. The serial clock (SCL) pin is what links the sensor to the CC3200's internal clock. This makes sure that instructions are being executed in sync between the board and the sensor. This pin connects to P01 on the board. The interrupt (INT) pin is active low when an external interrupt event occurs. This pin connects to P58 on the board.
The APDS-9960 gesture sensor comes with two source files. These files are titled "Sparkfun-APDS9960.cpp" and "Sparkfun-APDS9960.h". They can be downloaded from the sensor's webpage on Sparkfun. [2] Unfortunately these files are written in C++ using Arduino Libraries. This was an issue initially, and I will go over the solution in the next section.

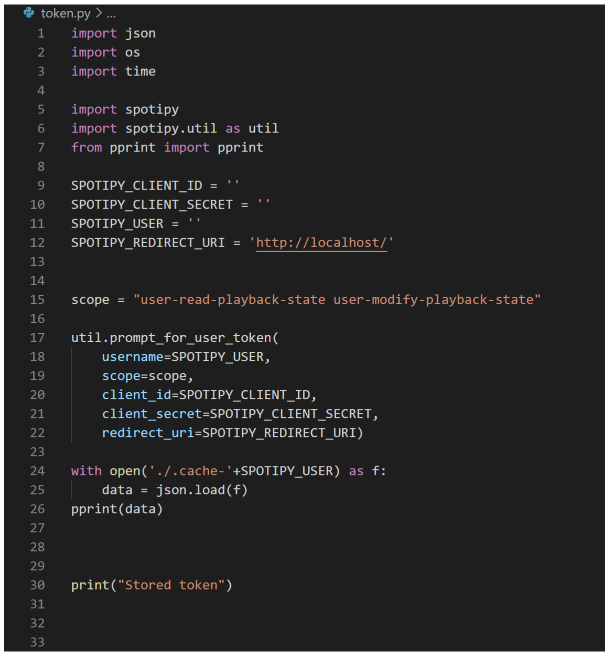
B. Energia

Energia is an Arduino IDE fork that enables users to run Arduino programs, also known as sketches, on TI microcontrollers even if they use Arduino libraries. Because the source files used Arduino libraries, Code Composer Studio was no longer viable for this project. Sparkfun also provided example code in their library download for the gesture sensor. This code can be imported into Energia and compiled to the CC3200 board. The provided code, however, needs to be changed in 3 places. In GestureTest.ino, the sensors example sketch, all calls of the functions detachInterrupt() and attachInterrupt() will have the integer 0 as their first argument. This argument must be changed from integer 0 to integer 2. After uploading the sketch to the CC3200, gestures read by the sensor will be printed to Energia's serial monitor. [1]

C. Connecting the CC3200 to AWS using Energia

Because of the new IDE and language change, a new way to connect the CC3200 to Amazon Web Services must be found. For this, altered versions of Arduino's "WifiClient" library were used from a similar project's GitHub repository, found in the references section. [3] This altered library allows for checking of the CC3200's flashed certificates downloaded from AWS IoT. Once the board is connected to AWS, we publish gesture data from the CC3200 to AWS using the MQTT topic "sensors/cc3200/cmd". MQTT is similar to HTTP, but in my opinion much similar. The advantages over HTTP are include directly sending data to a user defined endpoint called a topic, and built in functions to handle both subscribing and publishing.

D. Communicating with Spotify



```
token.py > ...
import json
import os
import time

import spotipy
import spotipy.util as util
from pprint import pprint

SPOTIFY_CLIENT_ID = ''
SPOTIFY_CLIENT_SECRET = ''
SPOTIFY_USER = ''
SPOTIFY_REDIRECT_URI = 'http://localhost/'


scope = "user-read-playback-state user-modify-playback-state"

util.prompt_for_user_token(
    username=SPOTIFY_USER,
    scope=scope,
    client_id=SPOTIFY_CLIENT_ID,
    client_secret=SPOTIFY_CLIENT_SECRET,
    redirect_uri=SPOTIFY_REDIRECT_URI)

with open('./.cache-'+SPOTIFY_USER) as f:
    data = json.load(f)
pprint(data)



print("Stored token")
```

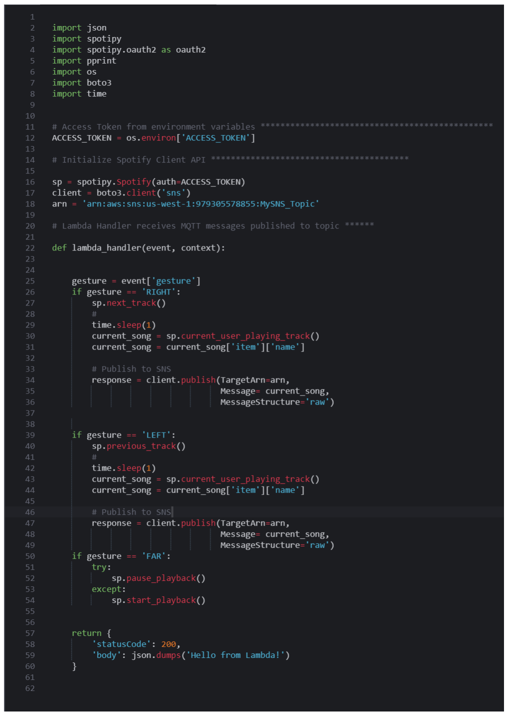
Fig. 2. token.py

To access user data using Spotify's API, one must first retrieve authorization. Spotify uses a type of authorization named OAuth2.0. [5] What this means is that a user must request an access token from Spotify using HTTP GET requests. When a request is sent to one of Spotify's API endpoints, Spotify replies with a dictionary containing an access token. This token is good for one hour upon receipt. This process usually takes many lines of code but is made extremely simple with the open source python library named "Spotipy." [4] Spotipy allows for authorization to be completed in just a few lines of code. I've created a program called "token.py" that requests an access token from the Spotify API using the "Spotipy" library. Before running the program, a user must create an application at Spotify's developer webpage and retrieve both their client ID and client secret ID. The two IDs must then be written into the client id and client secret variables in the program as strings. The user's Spotify username must also be specified as a string under the user variable. When the program is run, a new page is opened in the users browser to complete authorization. An access token is returned and printed in the terminal afterward.

E. AWS Lambda

In AWS Lambda, we create a function and set its trigger to be data published to our MQTT topic. We also set the functions environment variable to be our access token we received from Spotify. Alternatively, this access token can be pasted into the access token variable in our lambda function. MQTT Data is sent as a key value pair, example as follows:

- mqttClient.publish("sensor/cc3200/cmd", DATA-RIGHT)
- DATA-RIGHT[] = "gesture": 'RIGHT';



```

import json
import spotipy
import spotipy.oauth2 as oauth2
import pprint
import os
import boto3
import time


# Access Token from environment variables ***********************************************
ACCESS_TOKEN = os.environ['ACCESS_TOKEN']

# Initialize Spotify Client API ****************************************

sp = spotipy.Spotify(auth=ACCESS_TOKEN)
client = boto3.client('sns')
arn = 'arn:aws:sns:us-west-1:979305578855:MySNS_Topic'

# Lambda Handler receives MQTT messages published to topic ******

def lambda_handler(event, context):


    gesture = event['gesture']
    if gesture == 'RIGHT':
        sp.next_track()
        #
        time.sleep(1)
        current_song = sp.current_user_playing_track()
        current_song = current_song['item']['name']

        # Publish to SNS
        response = client.publish(TargetArn=arn,
                                  Message= current_song,
                                  MessageStructure='raw')


    if gesture == 'LEFT':
        sp.previous_track()
        #
        time.sleep(1)
        current_song = sp.current_user_playing_track()
        current_song = current_song['item']['name']

        # Publish to SNS
        response = client.publish(TargetArn=arn,
                                  Message= current_song,
                                  MessageStructure='raw')
    if gesture == 'FAR':
        try:
            sp.pause_playback()
        except:
            sp.start_playback()


    return {
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda!')
    }


```

When data is sent, our AWS Lambda function is triggered. Our lambda function, "lambda-function.py" processes this data as an event in the lambda handler and decides what to do with it. If the gesture data reads "RIGHT", a function is called that skips to the next song. If the gesture data reads "LEFT" the player goes back to the previous song. If the data reads "FAR", the player pauses or plays the current song.

III. Results

The whole project came together nicely after many hours of reading and research into Energia, MQTT, AWS IoT and Lambda, and the Spotify API. A demo can be found at this link:

- https://youtu.be/zylCQy4vR20

After some non-technical user testing, the pause function was not reading as consistently as I would have liked. To remedy this, I added pause functionality to gestures ”NEAR”, ”DOWN”, and ”NONE”. None is the default value when a gesture cannot be read. However, gestures ”RIGHT” and ”LEFT” are incredibly consistent and are always read, so ”NONE” and therefore pause is never accidentally triggered. The lambda function is purposefully kept short to reduce latency. As it stands, from gesture action to Spotify reaction, the latency is extremely small and can even beat manually changing songs in some cases. The functionality of the gesture sensor could only really be beat if the user was already hovering their mouse cursor on the Spotify player buttons.

IV. Conclusion

Ultimately, I am extremely happy with how the project turned out. For me, it felt great to put so many hours into a project I was overwhelmingly interested in. Making small progress or clearing an obstacle in the development process felt really good and reminded me why I chose an engineering major.

Acknowledgment

I would like to thank the creators of the Spotipy library as they fast tracked gaining authorization and interacting with Spotify’s API. I would also like to thank the creator of the CC3200-AWS email project, as it helped greatly in connecting the board to AWS using Energia and Arduino libraries rather than in C using Code Composer Studio.

References

[1] Energia. https://energia.nu/.
[2] Sparkfun apds-9960. https://www.sparkfun.com/products/12787.
[3] McFizh. hackster-cc3200-email. https://github.com/McFizh/hackster-cc3200-email.
[4] Paul Plamere. Spotipy. https://spotipy.readthedocs.io/en/2.17.1/.
[5] Spotify. Spotify api. https://developer.spotify.com/documentation/web-api/.