



# Tecnológico de Monterrey

## Reporte Técnico de Situación Problema

**Diseño de Chips**

16 de Junio 2022

### **Integrantes**

A01275161 | José Alfredo García Alvarado  
A01236090 | Arturo José Murra López  
A01284083 | Oscar Cárdenas Gómez  
A01284147 | Ricardo Gonzalo Cruz Castillo

## **Introducción**

Esta situación problema consiste en desarrollar un reproductor de música implementando un sistema computacional basado en microcontroladores. Los microcontroladores son un sistema integrado que contiene todos los componentes necesarios para construir una computadora. Estos se pueden usar para desarrollar proyectos biomédicos, de IoT, de robótica y automotrices.

## **Justificación del problema**

Aproximadamente, alrededor del 40% de la producción de un automóvil es de los sistemas electrónicos incluidos. Estos sistemas tienen un microprocesador o microcontrolador que les permite cumplir con tareas específicas, en este caso la tarea es realizar un sistema de infoentretenimiento. Estos sistemas forman gran parte de automóviles modernos, pero también forman parte de muchos sistemas en diferentes áreas de la vida cotidiana. El realizar un sistema de infoentretenimiento nosotros mismos, pone en práctica nuestros conocimientos de la raspberry pi, microcontroladores, sistema operativo de linux y stm32BluePill.

## **Metodología**

### **Herramientas de Desarrollo**

- STM32CUBE IDE
  - Es una plataforma de desarrollo multi-OS en C/C++ para el ecosistema de microcontroladores y microprocesadores STM32.
- VSCODE
  - VSCode es un editor de código de fuente desarrollado por Microsoft.
- Spotify API
  - Application Programming Interface. Conjunto de subrutinas y funciones que ofrece cierta biblioteca para ser utilizada por otro software como capa de abstracción.

### **Lenguajes de programación**

- C
  - Lenguaje de programación de propósito general.
  - Se utiliza para el desarrollo de software como OS, compiladores, bases de datos.
- Python
  - Lenguaje de programación de propósito general de alto nivel. Hace hincapié en la legibilidad del código.

### **Periféricos**

- Keypad
  - Keypad Matricial 4x4 para las diferentes funcionalidades de un reproductor de audio
- LCD
  - Display para mostrar los detalles del audio que se está reproduciendo

- FTDI
  - Establece la comunicación serial del controlador y raspberry pi
- USB
  - Para Albergar los programas que dan utilidad al proyecto
- Bocina

### **Manual de instrucciones del funcionamiento del sistema**

- C “Play”
  - Botón utilizado para continuar la reproducción de una canción.
- D “Pause”
- Botón utilizado para dar pausa a la reproducción de una canción.
- “Rewind”
  - Botón que se regresa al inicio de una canción.
- “Forward”
  - Botón que reproduce la siguiente canción

### **Marco teórico**

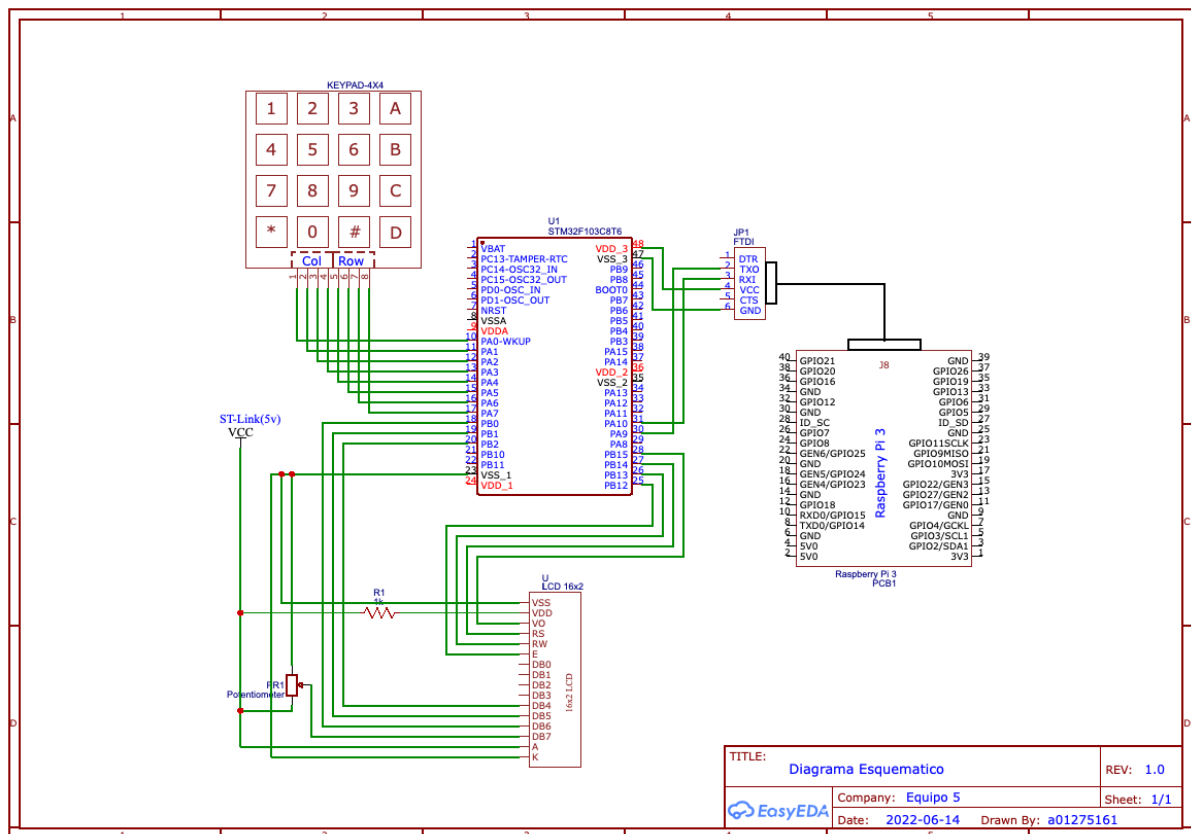
La placa stm32 es una placa de 32 bits y de bajo costo que sirve para realizar proyectos sobre un microcontrolador. La Raspberry Pi también es una computadora compacta y de bajo costo que sirve para computación de uso general. Estos dispositivos pueden usar el sistema embebido de Linux y sirven para usar bibliotecas y aplicaciones directamente en ellas. El problema es que no podemos utilizar la raspberry pi en implementaciones de tiempo real, entonces, cómo se utilizó en el proyecto, es mejor que sea usada como centro de inteligencia. Durante este proyecto se utilizaron diferentes tareas y se tuvo que utilizar la calendarización de tareas. Esto nos ayudó a darle diferentes prioridades a las tareas y saber que tareas se tienen que realizar y cuando se tienen que realizar. Además utilizamos threads, que son procesos que comparten memoria con otros procesos. Se utilizaron estos threads para poder organizar los recursos del proyecto.

#### **Links de funcionamiento:**

<https://drive.google.com/file/d/1t6P-tfrCuwtLUW-rq6iRmhu6hpXme8Z5/view?usp=sharing>

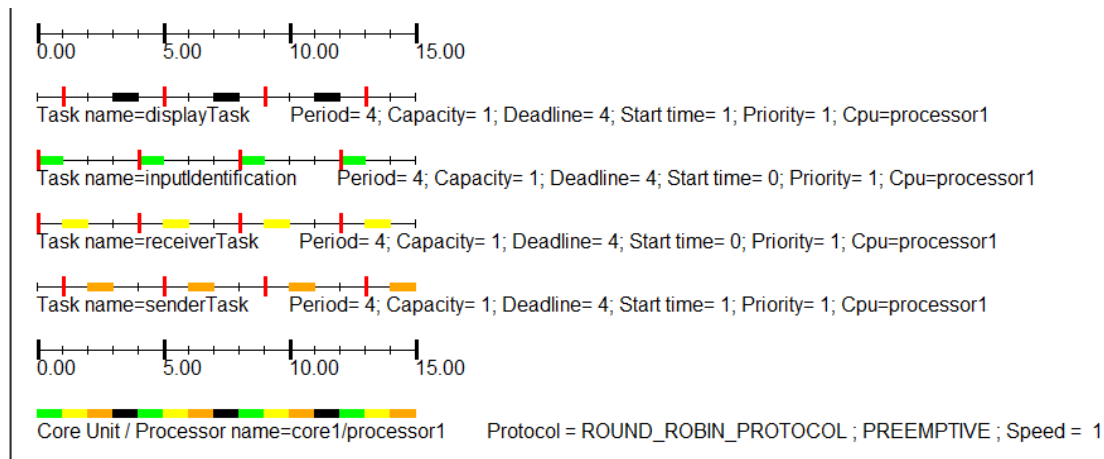
[https://drive.google.com/file/d/16C\\_iorUK5yc-0B56\\_qvG0hic5-SvOgKH/view?usp=sharing](https://drive.google.com/file/d/16C_iorUK5yc-0B56_qvG0hic5-SvOgKH/view?usp=sharing)

### **Diagrama esquemático general del sistema**



## Linea de tiempos de calendarización de tareas del CMSIS RTOS

### Tiempos de ejecución aproximados



Calendarización basada en algoritmo Round Robin con un quantum de 1 segundo, considerando 3 tareas apenas pasando un segundo y la cuarta cercana a 0.

## Explicación de partes fundamentales del código en el stm32

```

osMessageQDef(MsgQueueHandle, 1, int);
MsgQueueHandle = osMessageCreate(osMessageQ(MsgQueueHandle), NULL);
osMessageQDef(MsgQueueHandle2, 1, uint8_t);
MsgQueueHandle2 = osMessageCreate(osMessageQ(MsgQueueHandle2), NULL);

/* USER CODE END RTOS_QUEUES */

/* Create the thread(s) */
/* definition and creation of defaultTask */

/* USER CODE BEGIN RTOS_THREADS */
/* add threads, ... */
/* Create the task, storing the handle. */
osThreadDef(iTaskHandle, inputIdentification, osPriorityNormal, 0, 128);
iTaskHandle = osThreadCreate(osThread(iTaskHandle), NULL);

osThreadDef(sTaskHandle, senderTask, osPriorityNormal, 0, 128);
sTaskHandle = osThreadCreate(osThread(sTaskHandle), NULL);

osThreadDef(rTaskHandle, receiverTask, osPriorityNormal, 0, 128);
rTaskHandle = osThreadCreate(osThread(rTaskHandle), NULL);

osThreadDef(dTaskHandle, displayTask, osPriorityNormal, 0, 128);
dTaskHandle = osThreadCreate(osThread(dTaskHandle), NULL);

```

Creación de las tareas con un nivel de prioridad equivalente para todas. Así como la creación de los queues que se encargaran del manejo de la transmisión de datos entre tareas.

```

void inputIdentification(void const * argument)
{
    /* USER CODE BEGIN 5 */
    osStatus_t i_event;
    int i = 0;
    /* Infinite loop */
    for(;;)
    {
        i = read_keypad();
        i_event = osMessagePut(MsgQueueHandle, i, 1000);
        osDelay(1000);
    }
    /* USER CODE END 5 */
}

void senderTask(void const * argument)
{
    /* USER CODE BEGIN 5 */
    osEvent s_event;
    osDelay(1000);
    /* Infinite loop */
    for(;;)
    {
        s_event = osMessageGet(MsgQueueHandle, 1000);
        if( s_event.status == osEventMessage ){
            if(s_event.value.v == 3)
                myprintf("R"); //B
            else if(s_event.value.v == 4)
                myprintf("P"); //C
            else if(s_event.value.v == 2)
                myprintf("F"); //A
            else if(s_event.value.v == 5)
                myprintf("C"); //D
        }

        osDelay(1000);
    }
    /* USER CODE END 5 */
}

```

Primera tarea “inputIdentification” se encarga de llamar a la función de interpretación del keypad y de enviar al primer queue el identificador del botón que haya sido presionado. Mientras que la segunda tarea, “senderTask”, recibe el dato del primer queue. Si la tarea detecta que fue exitosa la recepción, enviará por medio del protocolo serial la instrucción que deberá de ejecutar la raspberry pi.

```

void receiverTask(void const * argument){
    osStatus s_event;
    uint8_t data;
    uint8_t i = ' ';

    for(;;){
        data = USER_USART1_Receive();
        if(data){
            i = data;
            s_event = osMessagePut(MsgQueueHandle2, i, 1000);
        }

        osDelay(1000);
    }
}

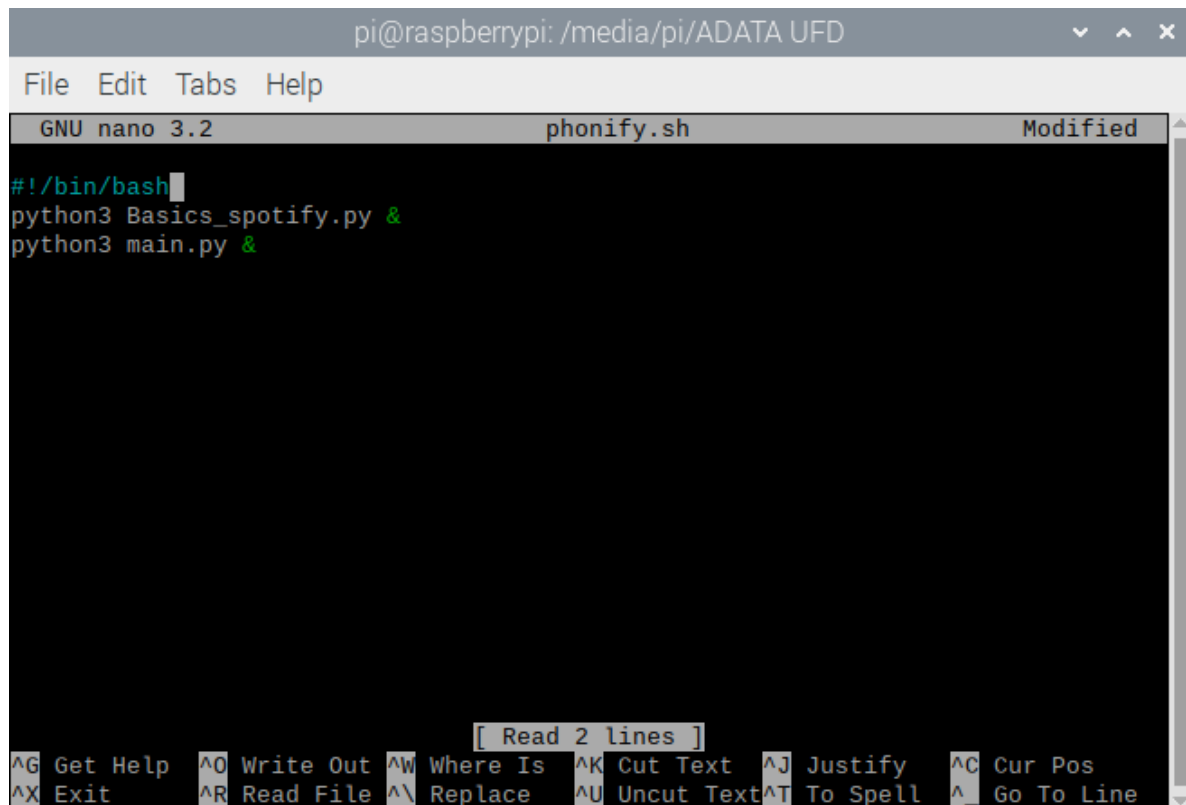
void displayTask(void const * argument){
    osEvent d_event;
    LCD_Init( );
    LCD_Cursor_OFF( );
    LCD_Set_Cursor(1, 0);
    LCD_Put_Str("Esta es la Prueba");
    uint8_t mystring[16] = "Esta es la prueb";
    osDelay(1000);

    for(;;){
        d_event = osMessageGet(MsgQueueHandle2, 1000);
        if( d_event.status == osEventMessage ){
            uint8_t tem = mystring[0];
            for(int i = 0; i < 16 - 1; i++) mystring[i] = mystring[i + 1];
            mystring[15] = d_event.value.v;
        }
        LCD_Clear();
        LCD_Put_Str(mystring);
    }
}

```

La tercera tarea llamada “receiverTask” está encargada de recibir por medio del serial la información con respecto a la canción que se esté reproduciendo en el momento, enviando esta información al segundo queue. La cuarta tarea, “displayTask” se encarga de recibir el nombre y autor de cada canción para posteriormente desplegarlas en una tira que se desplaza en el display LCD.

### **Explicación de partes fundamentales del código en la Raspberry Pi Bash(Inicialización en paralelo)**



```
pi@raspberrypi: /media/pi/ADATA UFD
File Edit Tabs Help
GNU nano 3.2 phonify.sh Modified
#!/bin/bash
python3 Basics_spotify.py &
python3 main.py &
[ Read 2 lines ]
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Spell ^_ Go To Line
```

Creamos un bash script en el cual le indicamos a la raspberry pi qué archivos debe de abrir simultáneamente, esto con el propósito de correr ambas funcionalidades al mismo tiempo para ahorrar tiempo.

## Raspberry Pi +Controlador

```
1 import time
2 import serial
3 import spotipy
4 from spotipy.oauth2 import SpotifyOAuth
5 import os
6
7 client_ID= os.environ.get('client_id')
8 client_SECRET= os.environ.get('client_secret')
9 redirect_url='http://localhost:9000'
10
11 scope= 'user-modify-playback-state','user-read-playback-state','user-read-currently-playing'
12
13 sp = spotipy.Spotify(auth_manager=SpotifyOAuth(client_id=client_ID, client_secret= client_SECR
14
15
16 ser = serial.Serial("/dev/ttyUSB0" , baudrate = 9600, parity=serial.PARITY_NONE, stopbits=seri
17 while 1:
18     rast = sp.current_user_playing_track() #Recupera el nombre de la canción
19     sart_name = rast['item']['name']
20     sart_artist = [artist for artist in rast['item']['artists']]
21     artist_names = ', '.join([artist['name'] for artist in sart_artist])
22     word = f'{sart_name} ({artist_names})' # formatea el nombre a cancion (artista[s])
23     i=0
24     while i< len(word):
25         time.sleep(1)
26         rx_data=ser.write(word[i].encode()) #Enviar nombre de la canción
27         print(rx_data) #Display in console
28         ser.write(rx_data) #LCD echo
29         print("Printed") #Flag de que jalo
30         rx_data = ser.readline() #Lee el input del controlador
31         rx_data=rx_data.decode("utf-8") #Re-interpreta
```

Importa las librerías necesarias para el funcionamiento del programa (time, serial, os y Spotify [la cual nos permite enlazar la API con nuestro lenguaje de programación python] ) Recolectamos las credenciales de la API de spotify (las cuales se llevaron

como variables de entorno para proteger su uso) y recabamos los scopes (o funcionalidades que queremos acceso del usuario de spotify). Inicializamos el puerto serial para establecer la comunicación del mismo nombre. Dentro de un loop infinito usamos la API de Spotify para capturar el nombre actual de la canción que está escuchando el usuario al momento. Al ser una api nos regresara por naturaleza información estructurada en formato JSON para tener una impresión de solo los datos que realmente ocupamos discriminamos la información que no ocupamos y configuramos un string con el formato de canción (artista[s]). Ejecutamos un segundo loop el cual se usará para mandar esta información al lcd, el cual dura solo la longitud del string formateado. Damos un sleep para darle tiempo de enviar los paquetes de datos los cuales se llevan por medio de una encriptación a bytes (debido a la naturaleza del protocolo) Así mismo en el loop podemos recibir los datos de la comunicación serial por parte del controlador la cual esta se decodifica a UTF-8 para poder interpretar con facilidad lo que estamos recibiendo.

```
rx_data=rx_data.decode ("utf-8") #Re-interpreta
print(rx_data) #Display en consola, de acuerdo a su input ejecuta los comandos
if ('C' in rx_data):
    state=sp.pause_playback()
elif ('P' in rx_data):
    stat=sp.start_playback()
elif('R' in rx_data):
    st=sp.previous_track()
elif ('F' in rx_data):
    s=sp.next_track()
time.sleep(1)
i += 1
```

En esta parte lo que estamos haciendo es leer el dato recibido y verificar que letra contiene al inicio (el cual nos será útil para poder controlar nuestro reproductor) La C será para darle pausa, P para darle reproducción, R ir a la canción anterior y F a la siguiente. Finalmente damos otro sleep para asegurarnos que no hayan datos faltantes y sumamos uno más al contador que nos ayuda a interpretar en qué carácter debemos de estar imprimiendo.



## Interfaz gráfica

```
def DOWN():
    d=sp.volume(50,"542f3eb9fff01f9a4f0895adb06b5b1d890be8e")
def rep():
    r=sp.repeat("track","542f3eb9fff01f9a4f0895adb06b5b1d890be8e")
gui = Tk()
gui.geometry("500x500")
gui.resizable(0,0)
gui.title("Phonify")
photo = PhotoImage(file = "p-logo.png")
gui.iconphoto(False, photo)
lbl =PhotoImage(file = "Fondo.png")
label1 = Label(gui,image= lbl)
label1.place(x=0,y=0)
button = Button(gui,text = "Request Song",command= clicked)
button.place(x=230,y=240)
up = Button(text = "Volume UP",command= UP )
up.place(x = 400 , y = 400)
down = Button(text = "Volume DOWN",command=DOWN)
down.place(x=0, y= 400)
rept = Button(text="Loop",command=rep)
rept.place(x=230,y=400)
gui.mainloop()
```

```
def clicked():
    record()
    if (usage() == "angry"):
        angry = Toplevel(gui)
        angry.configure(bg = 'red')
        angry.title("Phonify Emotion Result ")
        angry.geometry("200x200")
        Label(angry,text="Voice Emotion is Angry",bg='red').pack()
        angry.after(5000,angry.destroy)
        gui.mainloop()
    elif(usage() == "sad"):
        Sad = Toplevel(gui)
        Sad.configure(bg = 'blue')
        Sad.title("Phonify Emotion Result")
        Sad.geometry("200x200")
        Label(Sad,text="Voice Emotion is Sad",bg='blue' ).pack()
        Sad.after(5000,Sad.destroy)
    elif (usage() == "happy"):
        Happy = Toplevel(gui)
        Happy.configure(bg = 'yellow')
        Happy.title("Phonify Emotion Result")
        Happy.geometry("200x200")
        Label(Happy,ext="Voice Emotion is Happy",bg='yellow' ).pack()
        Happy.after(5000,Happy.destroy)
    elif (usage() == "calm"):
        Calm = Toplevel(gui)
        Calm.configure(bg = 'green')
        Calm.geometry("100x100")
        Calm.title("Phonify Emotion Result")
        Label(Calm,text="Voice Emotion is calm",bg='green' ).pack()
        Calm.after(5000,Calm.destroy)
def UP():
    u=sp.volume(100,"542f3eb9fff01f9a4f0895adb06b5b1d890be8e")
```

Utilizando la librería de Tkinter (la cual está especializada en la creación de interfaces gráficas en python) creamos una interfaz root (o principal) le damos tamaño default,le damos funcionalidad de poder manipular su tamaño, establecemos título que tendrá en display una vez inicializado, fondo de pantalla, los botones con su nombre y comandos a ejecutar si se presionan así como donde se ubican en la interfaz. Los comandos están en diferentes métodos algunos son los que usamos para poder manipular spotify con el cual usamos la api nuevamente y realizamos el mismo proceso de llamar credenciales y scopes para su uso en los casos de down, up y rep nos referimos para el uso de volumen (incrementar o subirle) y en repetir la misma canción reproduciendo. El método clicked es para poder acceder a otro script de python que se verá más adelante por medio del método record() y el if elif statement nos brinda la posibilidad de usar la red neuronal tal cual la cual detecta las emociones del usuario y nos ayuda a darnos las canciones apropiadas a las emociones que experimentamos. Cada uno de los stamentos creará una segunda pantalla la cual acorde a la emoción nos da diferentes ventanas de colores amarillo (feliz), azul (triste), rojo (Enojado) y verde (calmado) con un titulo en comun el cual es “Phonify Emotion result “ con la misma geometria y con un texto el cual nos da que emoción estamos experimentando.

## Captura de Audio

```
# import required libraries
from asyncio import subprocess
import sounddevice as sd
from scipy.io.wavfile import write
import wavio as wv
from pydub import AudioSegment

def record () :
    # Sampling frequency
    freq = 44100

    # Recording duration
    duration = 5

    # Start recorder with the given values
    # of duration and sample frequency
    recording = sd.rec(int(duration * freq),
        |         |         |         | samplerate=freq, channels=2)

    # Record audio for the given number of seconds
    sd.wait()

    # This will convert the NumPy array to an audio
    # file with the given sampling frequency
    write("recording0.wav", freq, recording)

    # Convert the NumPy array to audio file
    wv.write("output.wav", recording, freq, sampwidth=2)
    print("Done rec")
    sound = AudioSegment.from_wav("output.wav")
    sound = sound.set_channels(1)
    sound.export("output.wav" format="wav")
```

En este script de python lo que estamos haciendo es modular una frecuencia, darnos una duración de la grabación, ejecutamos la grabación con estos parámetros y lo guardamos en un archivo de tipo .wav. [Nota: la razón por la que hacemos dos archivos es debido a que uno lo usamos como base de la conversión de audio y el otro es el archivo de audio que se usa y se puede reproducir de manera común] Finalmente para poder ser utilizado en la red neuronal debe de estar en mono y en

este proceso lo grabamos como stereo para darle mejor calidad. Por lo tanto realizamos una conversi3n a mono y exportamos el archivo.

## Red Neuronal (Analizador de Librería Spotify)

```
happy=[]
sad = []
angry = []
calm = []

def Happy():
    #Grabs ID for Liked Songs
    print ('\n', "Grabbing the liked songs id")
    results = sp.current_user_saved_tracks(limit=50)
    for idx, item in enumerate(results['items']):
        track = item['track']
        metrics=sp.audio_features(track['id'])
        if(metrics[0]['valence']>0.5 and metrics[0]['energy']>0.5 ):
            #print(idx," ",track['name'],", " ",track['id'],", " ", "Valence: ",metrics[0]['valence'], "Energy: ",metrics[0]['energy'])
            happy.append(track['id'])if track['id'] not in happy else happy
            song = track['id']
            #print(song)

    ('\n', "Grabbing the playlist id")
    #Grabs ID for Playlists
    further_results = sp.current_user_playlists(limit=50)
    for idx, item in enumerate(further_results['items']):
        playlist = item['id']
```

```
print(idx,item['name'], playlist)
map_playlist = sp.user_playlist_tracks(None,playlist)
try:
    for idy, item in enumerate(map_playlist['items']):
        track = item['track']
        metrics=sp.audio_features(track['id'])
        if(metrics[0]['valence']>0.5 and metrics[0]['energy']>0.5 ):
            #print('\t',idy," ",track['name'],", " ",track['id'],", " ", "Valence: ",metrics[0]['valence'], "Energy: ",metrics[0]['energy'])
            happy.append(track['id'])if track['id'] not in happy else happy
            song = track['id']
except:
    pass

print ('\n', "Grabbing Albums id")
#Grabs ID for Albums
beyond_results = sp.current_user_saved_albums(limit=20)
for idx,item in enumerate(beyond_results['items']):
    albums = item['album']['id']
    print(idx,item['album']['name'],albums)
    map_album = sp.album_tracks(albums,limit=50)
    for idy, item in enumerate(map_album['items']):
        track = item['id']
        metrics = sp.audio_features(track)
        if(metrics[0]['valence']>0.5 and metrics[0]['energy']>0.5 ):
```

```
print ('\n', "Grabbing Albums id")
#Grabs ID for Albums
beyond_results = sp.current_user_saved_albums(limit=20)
for idx,item in enumerate(beyond_results['items']):
    albums = item['album']['id']
    print(idx,item['album']['name'],albums)
    map_album = sp.album_tracks(albums,limit=50)
    for idy, item in enumerate(map_album['items']):
        track = item['id']
        metrics = sp.audio_features(track)
        if(metrics[0]['valence']>0.5 and metrics[0]['energy']>0.5 ):
            #print('\t',idy," ",item['name'],", " ",track," ", "Valence: ",metrics[0]['valence'], "Energy: ",metrics[0]['energy'])
            happy.append(track)if track not in happy else happy
            song = track
```

```

Happy()
Sad()
Angry()
Calm()
sizehappy=len(happy)
print(sizehappy)
sizesad=len(sad)
print(sizesad)
sizeangry=len(angry)
print(sizeangry)
sizecalm=len(calm)
print(sizecalm)

SpotifyLibrary = {"HappySongs": happy, "SadSongs": sad, "AngrySongs": angry, "CalmSongs": calm}

import json
with open('data.json', 'w', encoding='utf-8') as f:
    json.dump(SpotifyLibrary, f, ensure_ascii=False, indent=4)

```

Inicializamos 4 arrays que son acorde al número de emociones que estaremos capturando. Así mismo generamos 4 métodos para las emociones. En estos métodos lo que hacemos es llamar a un método de la api de spotify para empezar a recolectar los datos de cada canción iniciando con las canciones gustadas por el usuario, después las que se encuentren en las playlists y finalmente en los álbumes salvados por el usuario (Nota: debido a las limitaciones de la api solo podemos obtener hasta 50 por eso el limit = 50 en el caso de las canciones gustadas y en el caso de los álbumes limit = 20 por la misma razón). Spotify ya nos facilita esta sección ya que ofrece un análisis de la canción. La cual nos da dos elementos clave para poder identificar qué tipo de canción es (si es feliz, triste, etc). Estos elementos son balance y energía. Al igual que con el script de conexión con el microcontrolador discriminamos toda la información exceptuando esas dos. Una vez recolectados las canciones que cumplan con los parámetros de balance y energía para poder ser clasificados acorde a las emociones serán adjuntados al array correspondiente. Así mismo para evitar repeticiones en el array checamos que la canción no se encuentre ya en el array. En dado caso que si, solamente no lo agregamos. Una vez que tenemos el array recolectamos la longitud de los arrays y a los arrays lo transformamos en un diccionario de python el cual más adelante lo transformamos en un archivo JSON. Para su fácil acceso y optimización de nuestro código.

## Red Neuronal (Entrenamiento de Modelo)

```
global X_test,y_test

#extracting features mfcc,chroma,mel from sound file
def feature_extraction(fileName,mfcc,chroma,mel):
    with sf.SoundFile(fileName) as file:
        sound = file.read(dtype='float32')#reading the sound file
        sample_rate = file.samplerate      #finding sample rate of sound
        if chroma:          #if chroma is true then finding stft
            stft = np.abs(librosa.stft(sound))
        feature = np.array([])              #initializing feature array
        if mfcc:
            mfcc = np.mean(librosa.feature.mfcc(y=sound,sr=sample_rate,n_mfcc=40).T,axis=0)
            feature =np.hstack((feature,mfcc))
        if chroma:
            chroma = np.mean(librosa.feature.chroma_stft(S=stft,sr=sample_rate).T,axis=0)
            feature = np.hstack((feature,chroma))
        if mel:
            mel = np.mean(librosa.feature.melspectrogram(y=sound,sr=sample_rate).T,axis=0)
            feature =np.hstack((feature,mel))
        return feature #return feature extracted from audio
```

```
#All available emotion in dataset
int_emotion = {
    "01": "neutral",
    "02": "calm",
    "03": "happy",
    "04": "sad",
    "05": "angry",
    "06": "fearful",
    "07": "disgust",
    "08": "surprised"
}

#Emotions we want to observe
EMOTIONS = {"happy","sad","calm","angry"}

#making and spliting train and test data
def train_test_data(test_size=0.05):
    features, emotions = [],[] #initializing features and emotions
    for file in glob.glob("data/Actor_*/*.wav"):
        fileName = os.path.basename(file) #obtaining the file name
        emotion = int_emotion[fileName.split("-")[2]] #getting the emotion
        if emotion not in EMOTIONS:
            continue
        feature=feature_extraction(file,mfcc=True,chroma=True,mel=True,) #extracting feature from audio
        features.append(feature)
```

```
        emotions.append(emotion)
    return train_test_split(np.array(features),emotions, test_size=test_size, random_state=0) #returning the data splited into train and test

#dataset
X_train,X_test,y_train,y_test=train_test_data(test_size=0.05)
print("Total number of training sample: ",X_train.shape[0])
print("Total number of testing example: ",X_test.shape[0])
print("Feature extracted",X_train.shape[1])

#initializing the multi layer perceptron model
model=MLPClassifier(alpha=0.01, batch_size=256, epsilon=1e-08, hidden_layer_sizes=(400,), learning_rate='adaptive', max_iter=1000)

#fitting the training data into model
print("_____Training the model_____")
model.fit(X_train,y_train)

#Predicting the output value for testing data
y_pred = model.predict(X_test)

#calculating accuracy
accuracy = accuracy_score(y_true=y_test,y_pred=y_pred)
accuracy*=100
print("accuracy: {:.4f}%".format(accuracy))
```

```
#saving the model
if not os.path.isdir("model"):
    os.mkdir("model")
pickle.dump(model, open("model/mlp_classifier.model", "wb"))
```

```
cf = confusion_matrix(y_test,y_pred,labels= ["happy","sad","angry","calm"])
print(cf)
cfr = metrics.classification_report(y_test,y_pred,labels= ["happy","sad","angry","calm"])
print(cfr)
```

Esta es en sí la red neuronal correspondiente. Iniciamos mediante un diccionario de python el cual le decimos que emociones vamos a poder encontrar en la base de datos que le proveeremos como entrenamiento y le indicamos que emociones son las que estaremos trabajando. En el método de características utilizaremos la librería de librosa como apoyo para realizar un análisis de los audios, agarramos los archivos enviados y realizamos su análisis correspondiente a gran profundidad. Una vez capturado ese análisis regresamos los resultados. Continuamos con un entrenamiento en el cual fragmentamos el uso de datos dividimos del 100% solamente un 5% de los datos para probar el modelo y 95% serán para entrenar el modelo. Recolectamos la base de datos que le proporcionamos y llamamos al método de análisis de características y le enviamos cada archivo que se encuentre en ella. Cómo le proporcionamos al modelo una guía de cada una de las emociones que usaremos adjuntamos sus resultados, de otra manera se ignora. Finalmente después de sus resultados lo regresamos. Ahora Construimos el modelo acorde a los resultados obtenidos anteriormente, y empezamos a predecir cuáles serían los resultados de la voz de nuestros usuarios. Imprimimos su resultado de precisión y salvamos el modelo usando pickle (para usar el modelo más adelante) y en una dirección en caso de no haberlo salvado antes. Por último para darle mejor certeza a nuestro modelo imprimimos la matriz de confusión (que tanto nuestro modelo puede fallar o malinterpretar datos) y un reporte desglosando el análisis de este mismo.

## Resultados de Entrenamiento

```
Total number of training sample: 729
Total number of testing example: 39
Feature extracted 180
_____Training the model_____
accuracy: 87.1795%
[[10  0  1  1]
 [ 2  9  0  0]
 [ 0  0  7  0]
 [ 0  1  0  8]]
```

	precision	recall	f1-score	support
happy	0.83	0.83	0.83	12
sad	0.90	0.82	0.86	11
angry	0.88	1.00	0.93	7
calm	0.89	0.89	0.89	9
accuracy			0.87	39
macro avg	0.87	0.89	0.88	39
weighted avg	0.87	0.87	0.87	39

## Aplicación

```
loaded_model = pickle.load(open("model/mlp_classifier.model", 'rb'))
print("Preparing to analyze Voice")
emotion = feature_extraction("t2.wav", mfcc=True, chroma=True, mel=True).reshape(1, -1)
# predict
result = loaded_model.predict(emotion)[0]
# show the result !
print("result:", result)

if (result == "angry"):
    f = open("data.json")
    k=json.load(f)
    song=k["AngrySongs"]
    n=randint(0,334)
    url="https://open.spotify.com"
    url = url + "/track/"+song[n]
    webbrowser.open(url)
    time.sleep(5)
elif (result == "sad"):
    f = open("data.json")
    k=json.load(f)
    song=k["SadSongs"]
    n=randint(0,472)
    url= "https://open.spotify.com"
    url= url + "/track/"+song[n]
```

```

        time.sleep(5)
        webbrowser.open(url)
    elif (result == "happy"):
        f = open("data.json")
        k=json.load(f)
        song=k["HappySongs"]
        n=randint(0,404)
        url = "https://open.spotify.com"
        url = url + "/track/" + song[n]
        time.sleep(5)
        webbrowser.open(url)
    elif (result == "calm"):
        f = open("data.json")
        k=json.load(f)
        song=k["CalmSongs"]
        n=randint(0,44)
        url = "https://open.spotify.com"
        url = url + "/track/" + song[n]
        time.sleep(5)
        webbrowser.open(url)

```

Para finalizar con el uso de la red neuronal, su aplicación consiste en cargar el modelo ya salvado, realizar una extracción de características de acuerdo al audio que salvamos en la grabación por medio del micrófono. Y predecimos el resultado de la emoción. Finalmente de acuerdo a cada emoción cargamos el archivo json con todas las canciones clasificadas y discriminamos qué secciones de este archivo no usaremos. Usamos un randomizer para poder obtener una canción única cada vez que lo corremos entre 0 y la longitud de canciones de acuerdo a la emoción y usando la librería de webbrowser abrimos la url de spotify de la propia canción.



## **Conclusión**

Un sistema embebido, usualmente realizado en tiempo real, se encarga de realizar funciones dedicadas. El procesamiento del sistema usualmente recae en un centro de inteligencia donde los datos son tratados, estos usualmente son microprocesadores o microcontroladores. La manera en la que se afrontó esta situación problema es digna de llamarse un sistema embebido puesto que se realizó una composición de tarjetas de desarrollo, periféricos y lenguajes de programación para poder llevar a cabo la función dedicada de un sistema reproductor de audio en un automóvil. Usando la Raspberry Pi como centro de inteligencia se recibieron inputs de entrada a través de un teclado matricial 4x4 en la STM32 Blue Pill. Se entabló una comunicación con la RPI para que ésta recibiera y procese los datos y diera instrucciones a la Blue Pill. Consecuentemente, el output enviado por la Blue Pill era desplegado en una pantalla LCD. Sin embargo, esta situación problema fue un proyecto retador puesto que implicó pasos como la definición de pines (para el keypad) o la comunicación sería entre placas de desarrollo. Afortunadamente, estos temas fueron vistos en clase lo cual resultó en un reto desafiante pero llevadero. Sin más que agregar, se puede decir que esta actividad fue un gran paso dentro del diseño de sistemas en chip.

## **Reflexiones individuales sobre aprendizaje**

Arturo: Hasta el momento, siento que ha sido la situación problema que más me ha dejado aprendizaje y fue una de las más interesantes también. Nunca había utilizado una raspberry pi y el hecho de que la utilizamos y también utilizamos la Stm32 todo al mismo tiempo fue como una nueva experiencia y algo que me va a servir mucho en un futuro. Es muy útil aprender lo que aprendimos sobre las conexiones de diferentes dispositivos y también el software y hardware por sí mismos. Además este tipo de sistemas que creamos son muy comunes y necesarios en la vida real, entonces siento que eso ayuda mucho a sacarle provecho al proyecto.

Alfredo: Me pareció un proyecto bastante retador en el planteamiento sin tener el conocimiento del funcionamiento de los diferentes elementos a integrar. Pero fue una experiencia interesante, el pensar en como un elemento que se estaba viendo en clase podría ser implementado para la resolución del reto. Pienso que tuve mucha experiencia con el manejo de todos los aspectos del proyecto y que me llevo las habilidades necesarias para montar un proyecto con esta cantidad de factores a considerar.

Ricardo: La elaboración de la situación problema me resultó muy amena. Creo que fue una gran manera de integrar los conocimientos que desarrollamos a lo largo de estas 10 semanas. Además, a lo largo de este periodo se fueron desarrollando actividades las cuales nos dieron las bases para poder realizar el proyecto. No lo

sentí apresurado con respecto al proyecto del primer bloque. De la misma manera, algo que me pareció muy útil que se implementó en este proyecto fue la comunicación entre la RPI y la Blue Pill. Esto es debido a que en 3r semestre, en la elaboración del reto de IoT, la placa con la que yo desarrolle el proyecto presentaba ciertas limitaciones y quería hacer que se comunicara un arduino uno con mi ESP32.

Oscar: El proyecto en sí fue algo muy familiar para mí, a diferencia de mis compañeros ya había trabajado con python y linux. Sin embargo debo de decir que la parte del microcontrolador sí fue algo retador y nuevo para mí. Puesto que nunca había trabajado con ellos. Para darle mayor profundidad y aprovechar mis conocimientos en esta área, quise realizar y tomar uno de los proyectos que había estado trabajando con anterioridad y reestructurarlo con base al contexto de este proyecto. Por ello la red neuronal, además de identificar con varios amigos y familiares e inclusive con mis compañeros de equipo que la música que escuchamos muchas veces lo hacemos acorde a nuestras emociones, una razón más para intentar mejorar la experiencia del usuario para mejor comodidad. En el lapso posterior a este semestre planeo retomar el proyecto y realizar múltiples mejoras.

## **Referencias**

STM32F103C8. (2021). STMicroelectronics.

<https://www.st.com/en/microcontrollers-microprocessors/stm32f103c8.html>

Le, H. (2021). *ARM Microcontrollers: Theory and Practical Applications*.

Cognella Academic Publishing.

Molloy, D. (2016). *Exploring Raspberry Pi: Interfacing to the Real World with Embedded Linux* (1. ed.). Wiley.

*Welcome to Spotipy! — spotipy 2.0 documentation*. (n.d.). Spotipy. Retrieved

May 25, 2022, from <https://spotipy.readthedocs.io/en/2.19.0/>

*RAVDESS Emotional speech audio*. (2019, January 19). Kaggle. Retrieved

June 8, 2022, from

<https://www.kaggle.com/datasets/uwrfkaggler/ravdess-emotional-speech-audio>

[o](#)

ProjectPro. (2022, June 14). *Speech Emotion Recognition Project using*

*Machine Learning*. Retrieved June 9, 2022, from

<https://www.projectpro.io/article/speech-emotion-recognition-project-using-machine-learning/573>

