UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

Reproductor de música con reconocimiento de voz

Autor: Fernando Navarro Sánchez

Tutor: Francisco José Martínez Zaldívar

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2019-2020

Valencia, 9 de septiembre de 2020

Escuela Técnica Superior de Ingeniería de Telecomunicación

Universitat Politècnica de València

Edificio 4D. Camino de Vera, s/n, 46022 Valencia

Tel. +34 96 387 71 90, ext. 77190

www.etsit.upv.es

VLC/ CAMPUS VALENCIA, INTERNATIONAL CAMPUS OF EXCELLENCE

EMAS Gestión ambiental verificada REG.NO. ES-CV- 000030

Resumen

El objetivo es crear un reproductor físico de música, al cual, mediante comandos de voz, se le pueda decir la canción o la *PlayList* que se desea escuchar y este la buscará en YouTube procediendo a su reproducción. También mediante pulsadores será posible controlar la reproducción, pudiendo pausar, continuar, parar, pedir siguiente canción y canción anterior. El reproductor estará conectado a una red Wi-Fi para poder reconocer los comandos que se le digan por voz y buscar la música. En caso de no tener configurada una conexión Wi-Fi en el dispositivo se podrá configurar una mediante una aplicación Android, empleando Bluetooth para la comunicación entre el reproductor y el terminal móvil. Gracias a una batería tiene la posibilidad de transportarse a cualquier lugar sin necesidad de conectarlo a la red eléctrica, salvo para su recarga.

Palabras Clave: Reproductor, Música, Pulsadores, Wi-Fi, Canción, YouTube.

Resum

L'objectiu és crear un reproductor físic de música, al qual, mitjançant ordres de veu, se li pugui dir la cançó o la *PlayList* que es desitja escoltar i aquest la buscarà a YouTube procedint a la seva reproducció. També mitjançant polsadors serà possible controlar la reproducció, podent pausar, continuar, parar, demanar següent cançó i cançó anterior. El reproductor estarà connectat a una xarxa Wi-Fi per poder reconèixer les ordres que se li diguin per veu i buscar la música. En cas de no tenir configurada una connexió Wi-Fi al dispositiu es podrà configurar una mitjançant una aplicació Android, emprant Bluetooth per a la comunicació entre el reproductor i el terminal mòbil. Gràcies a una bateria té la possibilitat de transportar-se a qualsevol lloc sense necessitat de connectar-lo a la xarxa elèctrica, excepte per a la seva recàrrega.

Paraules Clau: Reproductor, Música, Polsadors, Wi-Fi, Cançó, YouTube.

Abstract

The objective is to create a physical music player, which, through voice commands, can be told the song or the *PlayList* that you want to listen to and it will search for it on YouTube, proceeding to play it. Also, by means of buttons it will be possible to control the reproduction, being able to pause, continue, stop, next song and previous song. The player will be connected to a Wi-Fi network, to be able to recognize the commands that are given to it by voice and search for music. If you do not have a Wi-Fi connection configured in the device, one can be configured through an Android application, using Bluetooth for communication between the player and the mobile terminal. Thanks to a battery, it has the possibility of being transported anywhere without having to connect it to the electrical network, except for recharging.

Keywords: Player, Music, Push buttons, Wi-Fi, Song, YouTube.

Índice

Tabla de Ilustraciones

Capítulo 1. Objetivo de este documento

En este capítulo se pretende exponer la motivación que llevó a realizar este trabajo, los objetivos que se persiguen e introducir los conceptos teóricos necesarios para una comprensión de la memoria.

1.1 Motivación

La motivación de la elaboración del proyecto fue la posibilidad de tener un dispositivo que reproduzca música, sin la necesidad de estar registrado en los servidores de grandes compañías. Es un dispositivo totalmente anónimo que no recopila datos de carácter personal. Permite disfrutar de la música que se desee escuchar en cualquier momento, sin que eso afecte en las sugerencias de YouTube para futuras reproducciones.

1.2 Objetivos

Los objetivos del proyecto son la creación de un dispositivo, capaz de reproducir música de forma autónoma y totalmente anónimo, en todos los aspectos. Desde el reconocimiento de voz, de forma *offline*, es decir, que el mismo dispositivo integre un reconocedor de voz en el sistema operativo, hasta la reproducción de la música, que no sea necesario utilizar un usuario. De tal forma que no se recopilen datos de carácter privado, como puede ser, el estilo de música que escucha una persona en concreto o un grupo de personas. O en caso del reconocimiento de voz que no se registre un perfil de voz. Ya que, aun empleando los sistemas de reconocimiento de voz de las grandes compañías, estos sistemas pueden crear una sensación de “Gran Hermano”, es decir, falta de intimidad.

Estos objetivos se analizarán en las conclusiones para ver hasta qué nivel es factible mediante el uso de los dispositivos empleados, llevar a cabo el proyecto.

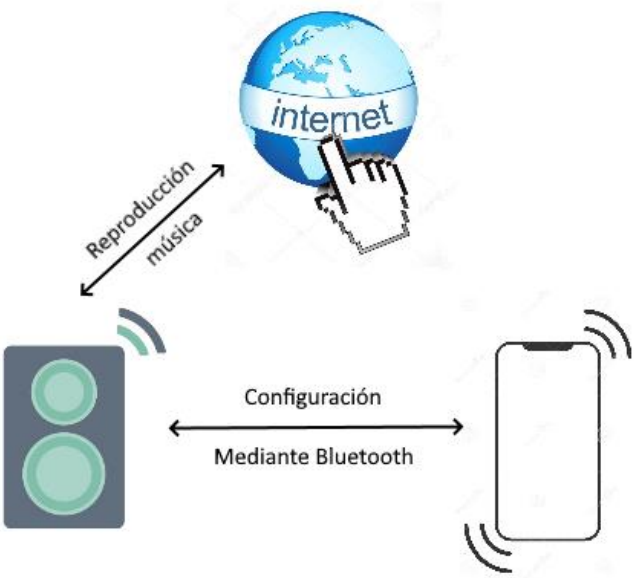

Ilustración 1: Gráfico conceptual

1.3 Marco Teórico

En el marco teórico se explican los conceptos del hardware y software empleados para el desarrollo del proyecto. Se emplea una Raspberry Pi 4 que es una placa de prototipo con un procesador de cuatro núcleos a 1,5GHz, muy común para el desarrollo de proyectos, por tanto, es el pilar clave para la ejecución del programa y la conectividad Wi-Fi y Bluetooth. Mediante una aplicación para Android y la comunicación Bluetooth se puede configurar la red Wi-fi a la que se conectará.

En el reconocimiento de voz se emplea una librería existente en Python que permite conectarse a diferentes servicios de reconocimiento de voz, ya sean *online* como *offline*. Una vez se han reconocido las órdenes que se tienen que llevar a cabo, tienen que ser tratadas para poder reproducir la música. Las librerías encargas de procesar la información son las librerías Pafy y VLC. Para que el usuario tenga una interacción con el sistema y pueda saber si todo está funcionando y la música que se reproducirá es la que ha solicitado se ha introducido una librería que convierte el texto a voz, Pyttsx3.

Los siguientes subpuntos del marco teórico, se describe la Raspberry Pi y las librerías empleadas en el proyecto.

1.3.1 Raspberry Pi

Se considera un ordenador de placa reducida y bajo coste. Desarrollada en Reino Unido por la fundación Raspberry Pi. El objetivo de la fundación fue la de facilitar la enseñanza de informática en los colegios de bajos recursos, pudiendo llegar a todos los niños.

El software es de código abierto, teniendo como sistema operativo oficial una versión adaptada de Debian, que tiene como nombre Raspbian, aunque permite el uso de otros sistemas operativos, incluido una versión de Windows 10.

Como Hardware tiene un procesador Broadcom, que cambia según la versión de placa. En cuanto a la memoria RAM, en la actualidad hay placas del último modelo con capacidades de almacenamiento diferente que van de un gigabyte hasta ocho gigabytes. Estos modelos incluyen grandes cambios respecto al resto de hardware, como el uso de USB tipo C para la alimentación de la placa, doble HDMI para conectar dos monitores y dos puertos USB3.0.

Gracias a esta placa se ha fomentado el uso y aprendizaje del lenguaje de programación Python. Aunque también existen otros lenguajes soportados como Tiny BASIC, C, Perl o Ruby.

Ilustración 2. Raspberry Pi 4

Una de las características más llamativas de la Raspberry Pi son los pines GPIO de entrada / salida de propósito general. Se encuentran en el borde superior de la placa un total de 40 pines en todas las placas actuales.

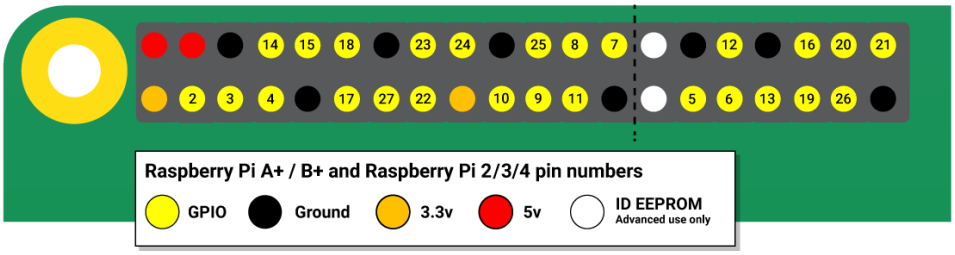

Ilustración 3. Pines GPIO Raspberry Pi

Para más información en las siguientes webs se encuentra la información completa sobre la Raspberry Pi.

https://es.wikipedia.org/wiki/Raspberry_Pi

https://www.raspberrypi.org/documentation/usage/gpio/

1.3.2 *Speech Recognition*

Es un conjunto de librerías, con soporte para muchos motores de API, tanto *online* como *offline*.

Los motores / API's soportadas por Speech Recognition son las siguientes:

- CMU Sphinx (offline)
- Google Speech Recognition
- Google Cloud Speech API
- Wit.ai
- Microsoft Bing Voice Recognition
- Houndify API
- IBM Speech to Text
- Snowboy Hotword Detection (offline)

La librería de referencia de *Speech Recognition* es pública y se puede encontrar en el siguiente enlace: https://github.com/Uberi/speech_recognition/blob/master/reference/library-reference.rst.

En este documento se realizará un pequeño resumen, centrándose en las partes relevantes para la realización del proyecto.

El uso de *Speech Recognition* necesita otro módulo de Python3 que es PyAudio, el cual proporciona una comunicación de los dispositivos de entrada / salida de audio, pudiendo con facilidad reproducir y grabar audio en diferentes plataformas. Como solo es necesario para la comunicación con el micrófono y los altavoces del dispositivo no se dará más información, no obstante, en el siguiente enlace se puede encontrar un documento más extenso donde se explica en detalle su funcionamiento. https://people.csail.mit.edu/hubert/pyaudio/docs/ .

La grabación del micrófono emplea fragmentos definidos en **"chunk_size"**, a una velocidad definida en **"sample_rate"**, es decir, muestras por segundos (hercios).

La utilización de valores altos en **"sample_rate"** aporta mejor calidad de audio, pero también necesita mayor ancho de banda (por tanto, un reconocimiento más lento). Algunas máquinas como los modelos obsoletos de Raspberry Pi, no pueden mantener el ritmo de procesado de datos si este valor es demasiado alto.

Los **"chunk_size"** con valores altos evitan la activación de la grabación del micrófono por ruido ambiental, aunque, por el contrario, la detección es menos sensible, pudiendo no detectar a la persona. Así que es conveniente no modificar el valor predeterminado salvo que sea necesario.

En caso de necesitar definir una duración para la grabación de audio, es posible emplear una vez definida la instancia de grabación, el comando **"duration: float = valor"**, siendo el valor definido en segundos y el menor de ellos tiene que ser 0.5.

Hay una forma de medir el nivel de ruido mediante la siguiente línea. **"recognizer_instance.adjust_for_ambient_noise(source, duration = valor)"**. Permite el ajuste del ruido ambiental definiendo la duración del tiempo de medición. La información será empleada para controlar la energía del ruido, que en caso de ser constante es posible definirlo, sin embargo, para este proyecto la opción de ajustar el ruido ambiental mediante la línea anterior es mejor opción.

Existen varias formas de definir cómo se quiere grabar el audio del micrófono, para un sistema de reconocimiento de voz se emplea el expuesto en las siguientes líneas. **"recognizer_instance.recognize_sphinx(audio_data: AudioData, language: str = "en-US", keyword_entries: Union[Iterable[Tuple[str, float]], None] = None, grammar: Union[str, None] = None, show_all: bool = False) -> Union[str, pocketsphinx.pocketsphinx.Decoder]"**. Esta línea ha sido extraída de la documentación. No será esta misma línea la empleada, porque no es necesario definir tanta información.

Realiza reconocimiento de voz en **"audio_data"**, en este caso utilizando CMU Sphinx, siendo uno de los reconocedores de voz *offline*. El idioma de reconocimiento se determina por **"language"** definido por una variable de tipo string con el valor **"en-US"**. Este sistema de reconocimiento de voz funciona con los siguientes idiomas: inglés, francés, chino e italiano.

La devolución de los datos está contenida en el objeto **"pocketsphinx.pocketsphinx.Decoder"**. En caso de **"show_all"** devolverá la transcripción más probable.

Sphinx funciona en distribuciones de Linux derivadas Debian. Para poder trabajar con idiomas completos es necesario bastante RAM (16GB recomendado) y bastante espacio de almacenamiento (20GB recomendado). Es posible encontrar más información en la documentación oficial en el siguiente enlace: https://github.com/Uberi/speech_recognition/blob/master/reference/pocketsphinx.rst

Uno de los reconocedores *online* de la lista es Google *Speech Recognition*, el cual es uno de los sistemas de reconocimiento de voz más potente. La siguiente línea es la utilizada para definirlo. **"recognizer_instance.recognize_google(audio_data: AudioData, key: Union[str, None] = None, language: str = "en-US", , pfilter: Union[0, 1], show_all: bool = False) -> Union[str, Dict[str, Any]]"**.

Es necesario el uso de una clave API que se especifica en **"key"**. Si no se especifica, utilizará una clave genérica, la cual tendrá limitaciones de cantidad de uso.

Al tratarse de Google los idiomas definidos en **"language"** son más extensos que en CMU Sphinx, definiendo los valores de igual forma. También es posible definir un filtro para no mostrar insultos o palabras malsonantes. El valor **"0"** es para no usar filtro, el valor **"1"** únicamente muestra el primer carácter y sustituye el resto con asteriscos, si no se define el valor se aplica **"0"** como valor predeterminado.

La resolución de los datos de audio es devuelta como un JSON (*JavaScript Object Notation* - Notación de Objetos de JavaScript), en caso de que **"show_all"** tenga un valor de falso, devolverá la transcripción más probable. Si el audio enviado es ininteligible provocará una excepción **"speech_recognition.UnknownValueError"**. Si no se dispone de conexión a Internet, fallo en la clave API o fallo en el reconocimiento de voz generará la siguiente excepción **"speech_recognition.RequestError"**.

1.3.3 API YouTube

Una API es una interfaz de programación de aplicaciones con un conjunto de subrutinas, funciones y procedimientos integrados en una biblioteca específica para ser usados por otro software.

Utilizando la API de YouTube es posible comunicarse con el sistema de vídeos de éste. La búsqueda de contenido y obtención de los resultados para ser reproducidos ya sea por vídeo más audio, audio únicamente o la miniatura del vídeo, según la necesidad que se tenga a la hora de realizar la aplicación. Para tener acceso al sistema de YouTube es necesario tener una clave

asociada a nuestro programa, de tal forma que no requerirá autorización de usuario. En la web de desarrolladores de Google, se encuentra la información al completo, que no es necesaria para el desarrollo del proyecto. https://developers.google.com/youtube/v3/quickstart/python

El módulo empleado para el uso de la API es **"youtube_dl".** Es un módulo que permite descargar vídeos de la plataforma, mediante línea de comandos. Como la mayoría de los módulos, funcionan en Unix, Windows y MacOS. Dicho módulo es de dominio público, pudiéndose modificar, redistribuir o usarse como se desee. En este proyecto se empleará dentro de la aplicación junto con Pafy y VLC. Otros módulos de los que se puede hablar, para reproducir mediante streaming el audio de los vídeos musicales.

El comando que se emplea para la ejecución del módulo sobre terminal del sistema es el siguiente:

```
youtube-dl [OPTIONS] URL [URL...]
```

En las opciones existe un amplio abanico de posibilidades, que proporcionará según el valor de OPTIONS una información y otra. A continuación, se expone una lista de algunas de las opciones y su utilidad.

```
-h, --help                       Print this help text and exit
--version                        Print program version and exit
-U, --update                     Update this program to latest version. Make
                                 sure that you have sufficient permissions
                                 (run with sudo if needed)
-i, --ignore-errors              Continue on download errors, for example to
                                 skip unavailable videos in a playlist
-4, --force-ipv4                 Make all connections via IPv4
-6, --force-ipv6                 Make all connections via IPv6
--write-thumbnail                Write thumbnail image to disk
--write-all-thumbnails           Write all thumbnail image formats to disk
--list-thumbnails                Simulate and list all available thumbnail
                                 formats
-q, --quiet                      Activate quiet mode
--no-warnings                    Ignore warnings
-s, --simulate                   Do not download the video and do not write
                                 anything to disk
--skip-download                  Do not download the video
-g, --get-url                    Simulate, quiet but print URL
-e, --get-title                  Simulate, quiet but print title
--get-id                         Simulate, quiet but print id
--get-thumbnail                  Simulate, quiet but print thumbnail URL
--get-description                Simulate, quiet but print video description
--get-duration                   Simulate, quiet but print video length
--get-filename                   Simulate, quiet but print output filename
--get-format                     Simulate, quiet but print output format
-j, --dump-json                  Simulate, quiet but print JSON information.
                                 See the "OUTPUT TEMPLATE" for a description
```

```
                                of available keys.
-J, --dump-single-json          Simulate, quiet but print JSON information
                                for each command-line argument. If the URL
                                refers to a playlist, dump the whole
                                playlist information in a single line.
--print-json                    Be quiet and print the video information as
                                JSON (video is still being downloaded).
--encoding ENCODING             Force the specified encoding (experimental)
--no-check-certificate          Suppress HTTPS certificate validation
--prefer-insecure               Use an unencrypted connection to retrieve
                                information about the video. (Currently
                                supported only for YouTube)
--user-agent UA                 Specify a custom user agent
--referer URL                   Specify a custom referer, use if the video
                                access is restricted to one domain
```

La lista completa de opciones se encuentra en el siguiente enlace, también está toda la información referente al módulo.

https://github.com/ytdl-org/youtube-dl

1.3.4 Pafy

Es un módulo que permite acceder a diferentes contenidos de YouTube y el procesado de este. Pudiendo reproducirlo en las diferentes opciones que permite, como cambiar la resolución. A continuación se adjunta una lista con las características del módulo.

Características

- Recupera metadatos como conteo de vistas, duración, calificación, autor, miniatura, palabras clave
- Descarga vídeo o audio a la resolución / velocidad de bits / formato / tamaño de archivo solicitados
- Herramienta de línea de comando (ytdl) para descargar directamente desde la línea de comando
- Recupera la URL para transmitir el vídeo en un reproductor como vlc o *mplayer*
- Funciona con vídeos restringidos por edad y vídeos no integrables
- Archivo de módulo pequeño, independiente y único importable (pafy.py)
- Selecciona la transmisión de mayor calidad para descargar o transmitir
- Descargar sólo vídeo (sin audio) en formato m4v o webm
- Descargar sólo audio (sin vídeo) en formato ogg o m4a
- Recuperar listas de reproducción y metadatos de listas de reproducción
- Funciona con Python 2.6+ y 3.3+
- Opcionalmente depende de **“youtube-dl”** (recomendado; más estable)

Ejemplo de uso

Se explica cómo usar el módulo con su propio código:

```
>>> import pafy

>>> url = "URL"

>>> video = pafy. new (url)

>>> streams = video. streams

>>> for s in streams:

... print (s)

...

normal: mp4 @ 1280x720

normal: webm @ 640x360

normal: mp4 @ 640x360

normal: flv @ 320x240

normal: 3gp @ 320x240

normal: 3gp @ 176x144
```

```
>>> for s in sequences:

... print (s. resolution, s. extension, s. get_filesize (), s. url)
```

```
...

1280x720 mp4 2421958510 https: //r1---sn-aiglln7e.googlevideo .com /
videoplayba [...]

640x360 webm 547015732 https://r1---sn-
aiglln7e.googlevideo.com/videoplaybac [...]

640x360 mp4 470655850 https://r1---sn-
aiglln7e.googlevideo.com/videoplayback [...]

320x240 flv 345455674 https://r1---sn-
aiglln7e.googlevideo.com/videoplayback [...]

320x240 3gp 208603447 https://r1---sn-
aiglln7e.googlevideo.com/videoplayback [...]

176x144 3gp 60905732 https://r1---sn-
aiglln7e.googlevideo.com/videoplayback? [...]
```

Los métodos de Pafy son una forma rápida para acceder a las transmisiones de mayor calidad, para un vídeo en particular sin necesidad de consultar las listas de transmisiones. **"Pafy.getbest()"**, **"Pafy.getbestaudio()"** y **"Pafy.getbestvideo()"**. Dentro de cada método es posible definir una serie de parámetros como el tipo de preferencia de codec de vídeo o audio. Estos métodos tienen como retorno **"pafy.Stream"**.

El objeto Stream puede ser usado para acceder a varios atributos. **"Stream.url"**, puede ser usado para retransmitir mediante *mplayer* o *vlc* o descargar con wget o curl. Es posible descargar directamente usando el método **"Stream.download()"**. **"Stream.url_https"**, es utilizado para acceder a la URL HTTPS.

Ejemplo de código.

```
>>> import pafy

>>> v = pafy.new("cyMHZVT91Dw")

>>> v.audiostreams
```

```
[audio:m4a@48k, audio:m4a@128k, audio:m4a@256k]

>>> mystream = v.audiostreams[2]

>>> mystream.rawbitrate

255940

>>> mystream.bitrate

'256k'

>>> mystream.url

'http://r20---sn-
aigllnes.c.youtube.com/videoplayback?ipbits=8&clen=1130...
```

En el enlace que se adjunta a continuación está toda la documentación.

https://pythonhosted.org/Pafy/

1.3.5 VLC (Python)

El módulo VLC para Python emplea libVLC (VLC SDK), puede ser integrado en una aplicación para obtener capacidades multimedia.

La instalación del módulo se realiza mediante la siguiente línea introducida en el terminal.

```
pip3 install python-vlc
```

La creación de un archivo es simple. Solo es necesario importar el módulo VLC.

```
import vlc
```

```
player = vlc.MediaPlayer (" ruta archivo a reproducir ")

player.play()
```

Con estas simples líneas son suficientes para reproducir un archivo de audio almacenado en nuestro disco.

Para pausar o parar el archivo de audio se emplearán los siguientes comandos.

```
player.pause()

player.stop()
```

En el desarrollo del proyecto se extenderá la información haciendo referencia al código de éste.

Para la información completa se puede acceder desde el siguiente enlace, donde está toda la documentación sobre el módulo VLC.

https://wiki.videolan.org/Python_bindings/

1.3.6 Pyttsx3 (TTS)

Es una librería de Text-to-Speech para Python. Funciona sin necesidad de conectarse a Internet, por tanto, es totalmente *offline*, siendo compatible con Python 2 y 3.

La instalación se realiza con el siguiente comando:

```
pip3 install pyttsx3
```

Para el uso de la librería únicamente será necesario importar la librería e inicializarla, con dos simples líneas es posible comprobar que la librería funciona correctamente.

```
import pyttsx3

engine = pyttsx3.init()

engine.say("I will speak this text")

engine.runAndWait()
```

Es posible cambiar de voz entre hombre y mujer, el volumen e incluso hay posibilidad si se desea guardar el contenido emitido en un fichero de audio. Para crear el fichero de audio, es necesario tener instalado **“espeak”** y **“ffmpeg”**.

```
"""VOZ"""

voices = engine.getProperty('voices')       #getting details of
current voice

#engine.setProperty('voice', voices[0].id)  #changing index, changes
voices. o for male

engine.setProperty('voice', voices[1].id)   #changing index, changes
voices. 1 for female

"""VOLUMEN"""

volume = engine.getProperty('volume')   #getting to know current
volume level (min=0 and max=1)

print (volume)                          #printing current volume level

engine.setProperty('volume',1.0)    # setting up volume level  between
0 and 1

"""Guardar la voz en un fichero"""

# On linux make sure that 'espeak' and 'ffmpeg' are installed
```

```
engine.save_to_file('Hello World', 'test.mp3')

engine.runAndWait()
```

Para más información sobre la documentación se puede consultar en el siguiente enlace.

https://pyttsx3.readthedocs.io/en/latest/engine.html

1.3.7 Android

Es un sistema operativo, principalmente para móvil desarrollado por Google, basado en el Kernel de Linux y otro software de código abierto.

Ilustración 4. Logo Android

La programación para Android se puede realizar en diferentes entornos de programación. Como por ejemplo Android Studio, totalmente gratuito. Será el empleado para el desarrollo de la aplicación de este Proyecto.

Android Studio es un entorno oficial para desarrollar aplicaciones para Android, basado en IntelliJ IDEA. Ofrece multitud de funciones para aumentar la productividad cuando se desarrollan aplicaciones.

- Un sistema de compilación flexible basado en Gradle
- Un emulador rápido y cargado de funciones
- Un entorno unificado donde puedes desarrollar para todos los dispositivos Android
- Aplicación de cambios para insertar cambios de códigos y recursos a la aplicación en ejecución sin reiniciar la aplicación
- Integración con GitHub y plantillas de código para ayudarte a compilar funciones de aplicaciones comunes y también importar código de muestra

- Variedad de marcos de trabajo y herramientas de prueba
- Herramientas de Lint para identificar problemas de rendimiento, usabilidad y compatibilidad de la versión, entre otros
- Compatibilidad con C++ y NDK
- Compatibilidad integrada con Google Cloud Platform, que facilita la integración con Google Cloud Messaging y App Engine

Para más información del funcionamiento de Android Studio, consultar el siguiente enlace.

https://developer.android.com/studio/intro

1.3.8 Scripts Linux (#!/bin/bash)

Es un lenguaje de programación usado por Bash para ser ejecutado por la Shell de Linux (Intérprete de comandos), ya que es nativo del sistema. Normalmente se emplea para la manipulación de archivos, ejecución de programas e impresión de texto. De tal forma que permite la automatización de tareas. Permitiendo la ejecución de comandos de Linux dentro del propio Script. Esto hace que sea un lenguaje de programación para Linux muy potente, pudiendo realizar operaciones complejas utilizando otros lenguajes de manera más sencilla y eficiente.

A continuación se muestra un ejemplo de un programa realizado con este lenguaje.

```
#!/bin/bash
echo Hola Mundo!
```

Para ejecutar un programa realizado con este lenguaje se hace de la siguiente manera. Primero es necesario darle permisos de ejecución mediante el siguiente comando. **“chmod +x script.sh”**. Una vez se ha dado permiso de ejecución ya es posible ejecutar el programa de la siguiente forma. **“./script.sh”**.

Para el desarrollo de este proyecto solo se necesita saber qué es este lenguaje de programación, ya que únicamente se utiliza para poder ejecutar la aplicación en la Raspberry Pi en el momento del arranque del sistema. Más adelante se detallará el proceso.

Capítulo 2. Código del proyecto

El código del proyecto está escrito en Python 3 dentro de la Raspberry Pi, salvo una línea de código que descubre el Bluetooth. En el teléfono móvil el código está escrito en Android Java.

Las condiciones previas para la realización del proyecto son las siguientes.

En la Raspberry Pi, hay instalado Raspbian OS, el cual tiene instalado previamente Python 3, sin embargo, los módulos mencionados en la parte teórica de la memoria, sí que son necesarios instalarlos de la siguiente manera.

Speech Recognition: `pip3 install SpeechRecognition`

API Youtube: `sudo -H pip3 install --upgrade youtube-dl`

Pafy: `pip3 install pafy`

VLC (Python): `pip3 install python-vlc`

Pyttsx3 (TTS): `pip3 install pyttsx3`

Una vez instalados todos los módulos ya es posible emplear el código del proyecto.

A continuación se muestra una gráfica de Gantt y un diagrama de flujo, del funcionamiento íntegro del programa.

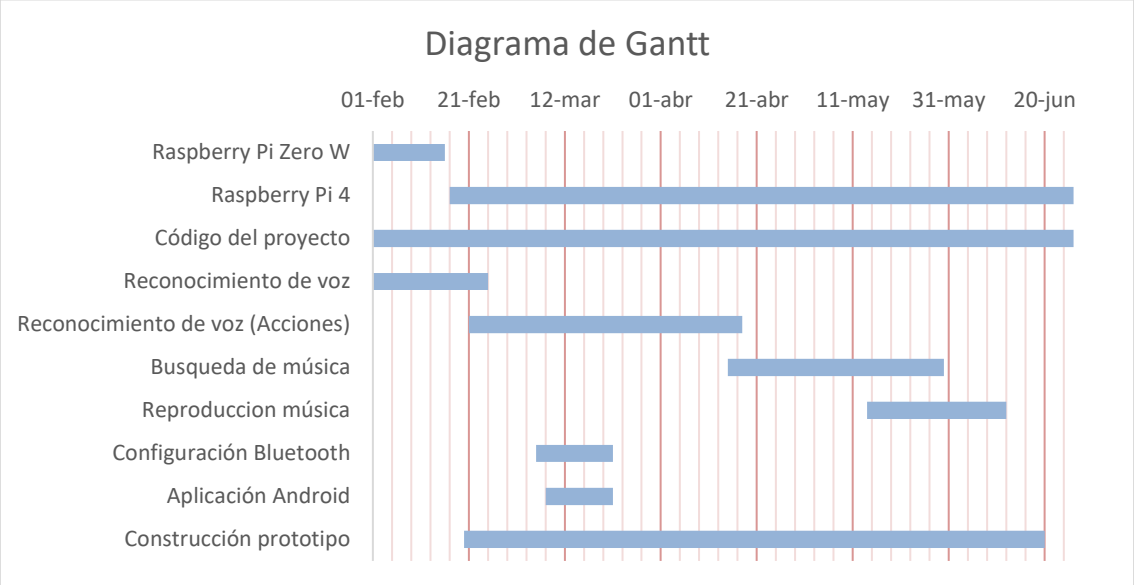

Ilustración 5: Diagrama de Gantt

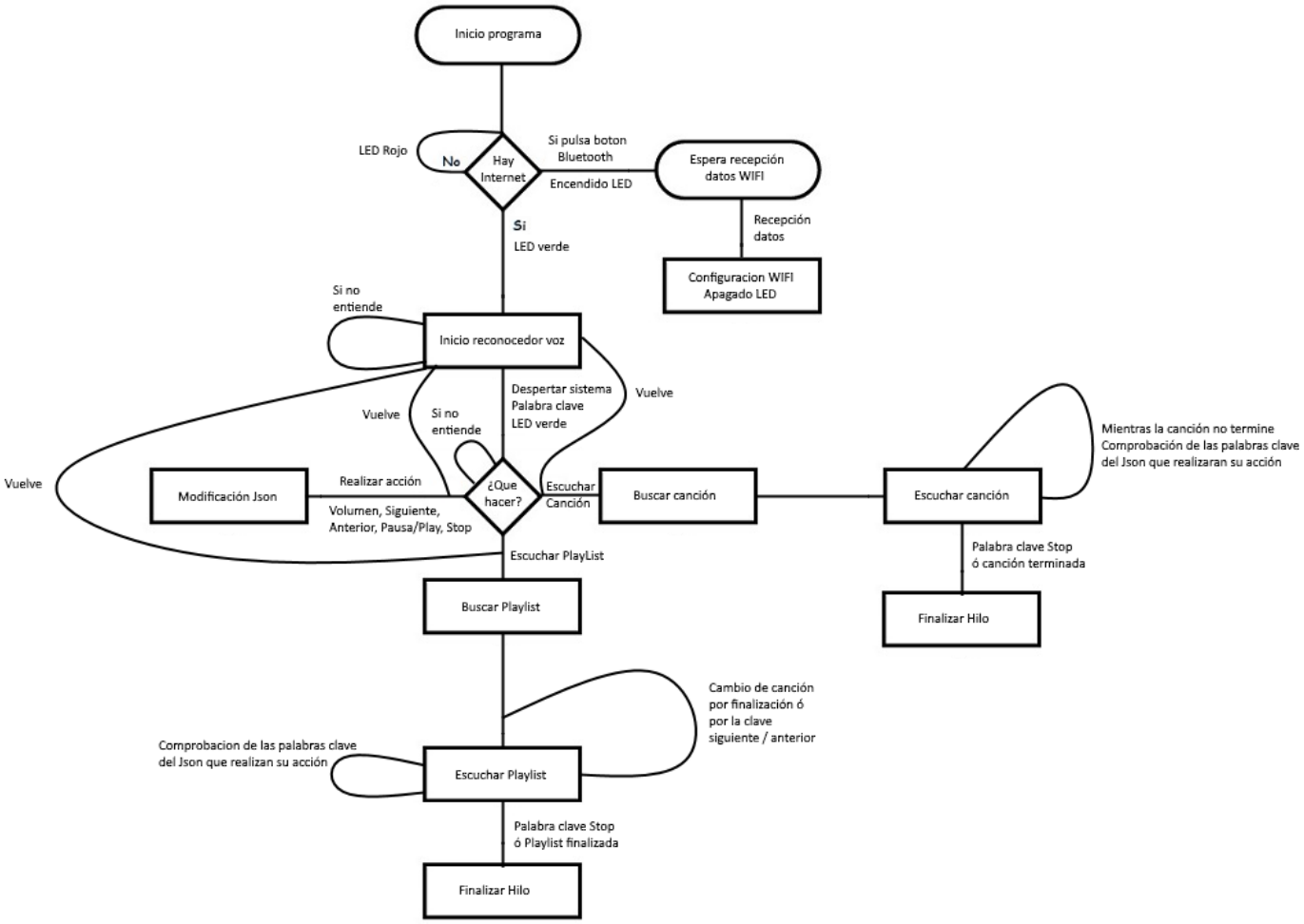

Ilustración 6. Diagrama flujo

2.1 Obtención de la clave de API

Para poder usar la API de YouTube, es necesario obtener la clave de desarrollo, sin ella no será posible realizar consultas a los servicios de YouTube y por tanto no será posible escuchar música. A continuación, se expone una guía de cómo conseguir dicha clave de desarrollador, además de otros datos que también serán necesarios.

Es necesario acceder al siguiente enlace. https://console.developers.google.com/ e iniciar sesión con una cuenta de Google, en caso de no tener, es necesario crear una. Una vez iniciada la sesión, se debe ir a la parte de credenciales.

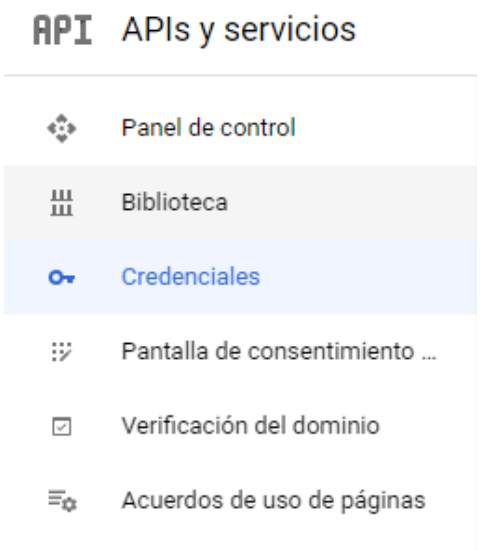

Ilustración 7: Credenciales

En el apartado de credenciales se pulsará sobre crear credenciales y clave de API. Aparece la siguiente ventana con la clave API que será asignada por Google. Si solo va a ser usada para acceso a YouTube, es conveniente restringir el acceso de la clave, de tal forma que nada externo a lo que esté destinado pueda utilizarla, porque una clave API puede tener acceso a todos los servicios de Google, pudiendo ser usado de forma malintencionada y los accesos que realiza una clave API también pueden conllevar un coste económico, dependiendo de la cantidad de accesos que realiza, como sucede en Google Maps.

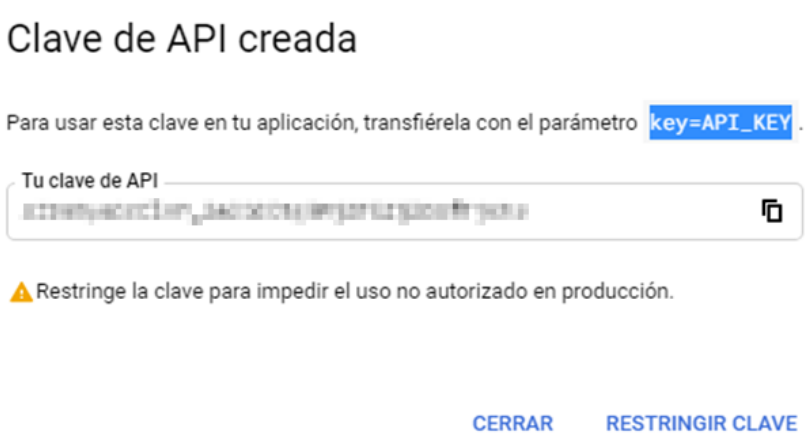

Ilustración 8: API Google

Para restringir el acceso de la clave API, se puede realizar de dos formas, pulsando sobre restringir clave como aparece en la ilustración 8 o, si ya está creada, se pulsará sobre el símbolo de editar que aparece a la derecha de la clave en la parte de credenciales. Aparecerá la siguiente información. Donde se deberá seleccionar lo marcado en la ilustración 9, en el desplegable, de las restricciones de API. En caso de no aparecer YouTube Data API v3, es posible filtrar la información que aparece en el desplegable. Una vez realizado todo, la información que necesitamos es la siguiente. API Key, versión de API en este caso v3 y API Service Name, que es youtube en minúsculas. Con esta información es posible realizar la búsqueda y reproducción de música desde YouTube.

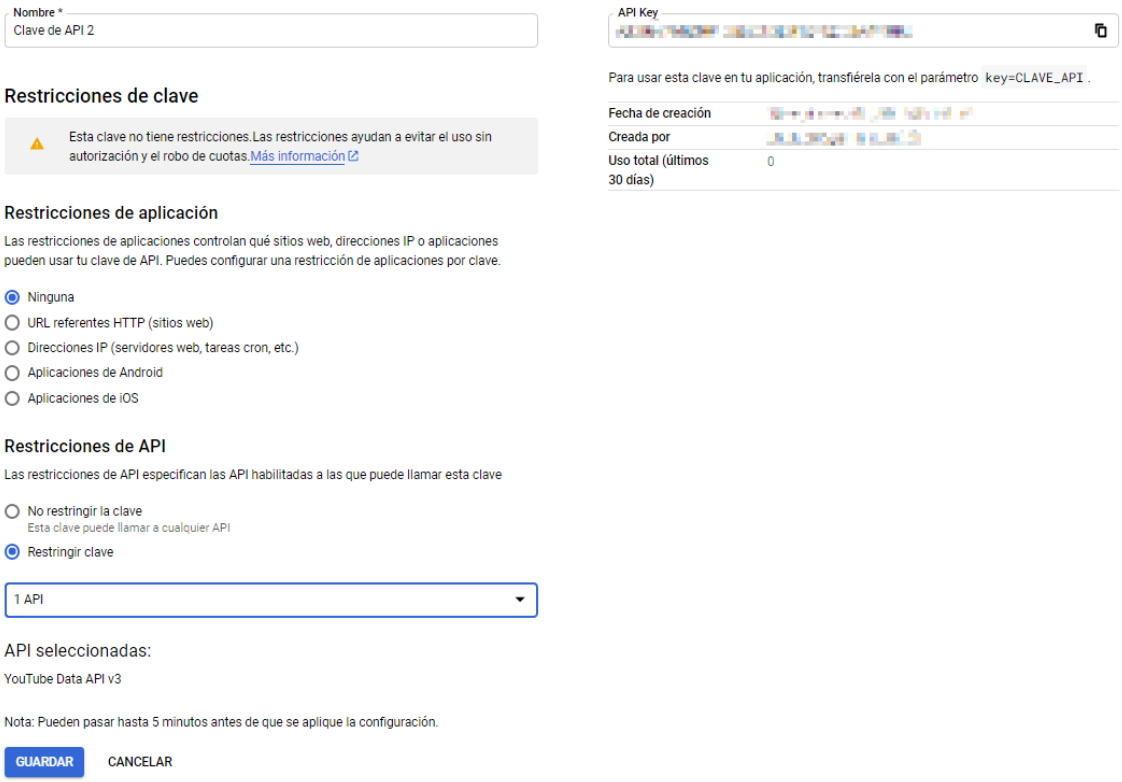

Ilustración 9: Restricciones API

2.2 Código en Raspberry Pi (Python)

El lenguaje de programación empleado para el desarrollo del programa en la Raspberry Pi es Python3. Se decidió emplear este, porque es el más versátil a la hora de emplear los GPIO. También al tratarse de un lenguaje actualmente en alza, existe mucha documentación y librerías actualizadas, como en el caso de la API de YouTube, que se actualiza constantemente.

2.2.1 *Comprobación de Internet y botón Bluetooth*

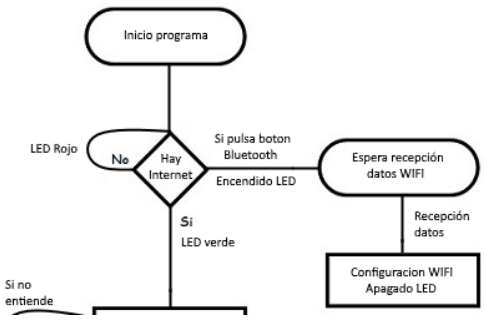

Ilustración 10: Parte del diagrama de flujo (Bluetooth)

El código del programa empieza comprobando si existe conexión a Internet. Si tiene acceso continuará con la ejecución del programa, pero en caso de no ser así se mantendrá dentro de un bucle, que seguirá comprobando si existe acceso. Al mismo tiempo comprobará si se ha pulsado el botón que descubre el sistema Bluetooth de la Raspberry Pi. Así podrá ser detectado por la aplicación móvil de la cual se hablará más adelante.

En el momento que el botón de activación de Bluetooth es pulsado se crea un hilo manteniéndose a la espera. Hasta recibir la configuración Wi-fi de la Raspberry Pi y así tener acceso a Internet.

A nivel de Hardware para identificar si hay o no acceso a Internet se emplea un LED RGB. En caso de no tener acceso, el LED será de color rojo y en caso contrario de color verde. En el apartado Bluetooth se emplea un segundo LED, que en el momento de la activación mediante la pulsación del botón se encenderá de color azul.

Para ello, se define la configuración de los LED y el botón de Bluetooth. Una vez configurado y definido, se apagan los LED para garantizar que todo esté iniciado desde cero y nada se encuentra en un estado que no se desea.

```
GPIO.setup(26, GPIO.OUT) # Rojo
GPIO.setup(19, GPIO.OUT) # Azul
GPIO.setup(13, GPIO.OUT) # Verde

GPIO.setup(12, GPIO.IN,pull_up_down=GPIO.PUD_UP) # Botón Bluetooth

GPIO.output(26, False) # Apaga LED Rojo
GPIO.output(19, False) # Apaga LED Rojo
GPIO.output(13, False) # Apaga LED Rojo
```

Para la comprobación de conectividad se ejecuta el siguiente código, que devolverá una variable de tipo booleano junto con un condicional para comprobar su estado. En caso de ser 'True', el programa continuará y llamará a la función encargada del reconocimiento de voz.

```
net = requests.get('https://google.com').ok # Comprobación
conectividad

if 'True' in net:

   GPIO.output(26, False) # Apaga LED Rojo
   GPIO.output(13, True)  # Enciende LED Verde

   voz() # Llamada al reconocimiento de voz
```

Si no existiera conectividad a Internet, el programa entrará en un bucle. Donde se comprueba si es pulsado el botón que provocará la entrada de otro condicional. El cual descubrirá el Bluetooth y crea un hilo que llamará a la función para configurar la conectividad Wifi.

```
GPIO.output(26, True) # Enciende LED Rojo

    while inet == "True":

      time.sleep(0.5)

      b = GPIO.input(12) # Botón Bluetooth
      if b == False:
        GPIO.output(19, True) #Enciende LED Azul

        os.system("echo " "'discoverable on \nquit' " "|
bluetoothctl") # Descubre el Bluetooth

        #blueconfig()
        t = threading.Thread(target=blueconfig, name="bluetooth")
        t.daemon = True
        t.start()
```

2.2.2 *Configuración Wifi mediante Bluetooth*

El método configura el socket del Bluetooth, para comunicarse con la aplicación móvil. La codificación del sistema de comunicación 'utf-8' y el UID, tienen como objetivo que ambos sistemas se entiendan entre sí y se realice la conexión.

```
server_sock=BluetoothSocket( RFCOMM )
server_sock.bind(("",PORT_ANY))
server_sock.listen(1)

port = server_sock.getsockname()[1]

encoding = 'utf-8';
uuid = "94f39d29-7d6d-437d-973b-fba39e49d4ee"
#Configuración de la conexión Bluetooth mediante puerto serie
advertise_service( server_sock, "Pi4-Bluetooth",
                   service_id = uuid,
                   service_classes = [ uuid, SERIAL_PORT_CLASS ],
                   profiles = [ SERIAL_PORT_PROFILE ]
                    )
```

Al no conectarse instantáneamente es necesario mantenerse dentro de un bucle, que espera la conectividad con la aplicación móvil. Dentro del mismo también se realizará el procesado de los datos recibidos en caso de realizar la conexión y el envió de datos. Se envía una respuesta a la aplicación del dispositivo móvil para confirmar la recepción.

```
Client_sock, client_info = server_sock.accept()

data = client_sock.recv(1024) #Recibimos los datos enviados desde el
móvil

if len(data) == 0:
   break
```

```
    info = str(data, encoding);

    data = 'Datos Recibidos'
    client_sock.send(data)
```

Una vez recibidos los datos que son el nombre de la red Wifi y su contraseña. Los cuales son recibidos separados por coma. De esta forma, utilizando la separación se procesan los datos para poder configurar la red Wifi. Una vez configurada, el Bluetooth se ocultará y se reinicia el sistema Wifi de la Raspberry Pi, para poder conectarse. También se apaga el LED y se cierran las conexiones de los sockets que se han creado.

```
#Dividimos los datos, para obtener el nombre de la red y la contraseña

m = re.search('(.*),(.*)', info) #El carácter "," es el separador de
los datos
SSID = m.group(1)
Password = m.group(2)

#Directamente escribimos sobre el fichero de configuración
wpa_supplicant

config = (
                '\nnetwork={{\n' +
                '\tssid="{}"\n' +
                '\tpsk="{}"\n' +
                '\tkey_mgmt=WPA-PSK\n' +'}}').format(SSID, Password)

with open("/etc/wpa_supplicant/wpa_supplicant.conf", "a+") as wifi:
            wifi.write(config)
wifi.close()
os.system("wpa_cli -i wlan0 reconfigure")  # Código esencial, para el
funcionamiento de la red.

#Cerramos la conexión Bluetooth
os.system("echo " "'discoverable off \nquit' " "| bluetoothctl") #
Descubre el Bluetooth

GPIO.output(19, False) #Apaga LED Azul
client_sock.close()
server_sock.close()
```

2.2.3 *Reconocimiento de voz (Despertar)*

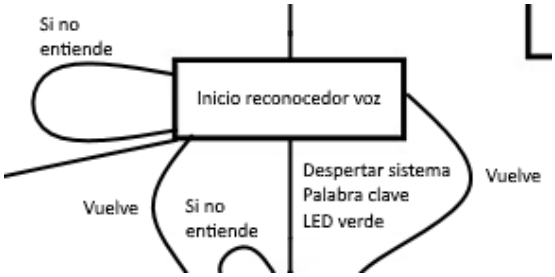

Ilustración 11: Parte diagrama de flujo (Despertar)

Dentro de la función del reconocimiento de voz se inicializa el sistema de voz. Una vez realizado se ajustará el ruido ambiental durante cinco segundos, para que no se envíe información no relevante y provoque una activación innecesaria. Una vez que el sistema tiene definida la energía del ruido ambiental, se mantendrá a la espera hasta que el micrófono reciba una respuesta sonora, como puede ser una persona hablando y se enviará a los servidores de Google, que devolverá la transcripción del audio en texto.

Para hacer saber que el sistema de reconocimiento de voz está activo se ha introducido unas líneas de código que pasan un texto a voz, así el usuario tiene una interacción con el sistema más satisfactoria.

```
r = sr.Recognizer()

""" Velocidad audio """
engine.setProperty('rate', 140) # configuración de la velocidad a la
que la frase es reproducido

with sr.Microphone() as source:
     #Escucha durante 5 segundos y creara un nivel de ruido ambiental
     r.adjust_for_ambient_noise(source, duration=5)

     # Sistema de texto a voz
     voices = engine.getProperty('voices')
     engine.setProperty('voice', voices[1].id)    # Selección idioma,
castellano
     engine.say("Sistema iniciado") # Texto que se utiliza

     engine.runAndWait()

     audio = r.listen(source)

     text = r.recognize_google(audio, language = 'es_ES')
```

Una vez se tiene la transcripción del audio enviado a Google se comprueba la existencia de la palabra clave, provocando la continuación del programa, encendiendo el LED2 en color verde y realizando la llamada a la siguiente función ‘peticion()’. En caso de no contener la palabra clave seguirá a la escucha hasta que la palabra clave aparezca.

```
if 'friday' in text.lower(): #Comprueba la palabra clave

   GPIO.output(7, False)  #Enciende LED Verde
   peticion() #Llamada a la siguiente función
```

2.2.4 *Reconocimiento de voz (Acciones)*

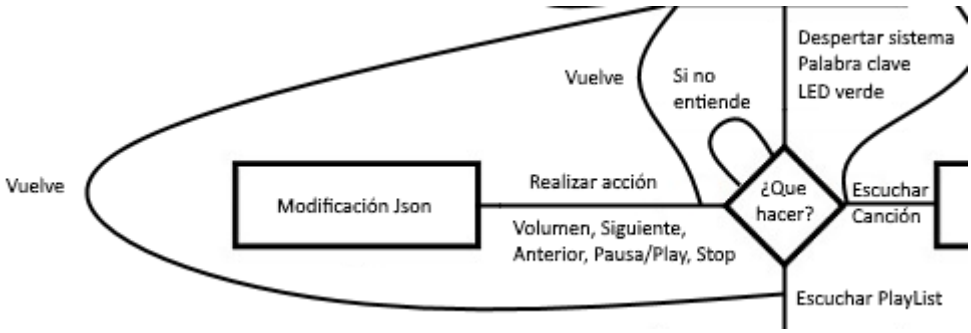

Ilustración 12: Parte del diagrama de flujo (Acciones)

Las acciones que se pueden realizar en esta función son las siguientes:

- Petición de canción o *PlayList*
- Cambios en el volumen en la reproducción
- Pausa o Continuar con la reproducción
- Pasar a la siguiente o a la anterior canción en caso de ser una *PlayList*

La metodología sobre el uso del sistema de voz es similar al punto anterior, a excepción del cálculo del ruido ambiental. En esta función se emplea otra alternativa que consiste en darle un valor de energía para que todo sonido inferior a este sea descartado.

Para gestionar las acciones, se emplea un fichero JSON, permitiendo cambios en tiempo real, ya que durante la reproducción será necesario la realización de cambios en el contenido del fichero. Por tanto, al inicio de la función se leerá el fichero para almacenar su contenido en una variable, que será modificada según las necesidades en función de las órdenes recibidas en la transcripción del audio a texto y se escribirán los cambios en el fichero JSON.

Al inicio se realiza la lectura del JSON. De esta forma al tener los datos actualizados del contenido del fichero, es posible pausar la reproducción en curso y por tanto facilitar el reconocimiento de voz. Así se garantiza que el nivel de ruido ambiental no sea superior al umbral fijado y el sistema no se mantenga a la escucha, provocando retrasos en el tratamiento de información y una espera innecesaria en la orden a ejecutar. En caso de no recibir ninguna orden tras un tiempo definido de diez segundos, la reproducción continuará.

```
with open("acciones.json", "r") as jsonFile:
      data = json.load(jsonFile)

      data["pausar"] = "3" # Si se reproduce música la para


with open("acciones.json", "w") as jsonFile:
      json.dump(data, jsonFile)
```

```
with sr.Microphone() as source:

    r = sr.Recognizer()
    r.energy_threshold = 400 # Valor de umbral de ruido
    escucha = 'True'

    audio = r.listen(source, timeout = 10) # Tiempo definido, para la
pausa
```

Las palabras clave o frases que activan cada acción son las siguientes.

- Peticiones: Quiero escuchar "Nombre de la canción"
- Peticiones: Quiero escuchar una *PlayList* de "Artista o Grupo"
- Volumen "Número de 0 a 100"
- Siguiente
- Anterior
- Pausar
- Continuar
- Stop

Según la palabra o frase utilizada se cumplirá una condición que realizará una acción determinada. Las acciones simples son similares, a excepción del volumen, que procesa el texto que devuelve el reconocedor de voz, para extraer el número dicho por el usuario y modificar el volumen de la reproducción. Una vez cumplida la condición se modifica el valor almacenado en la variable y se escribe en el fichero JSON, siendo usado por el método de reproducción de música para modificar el volumen, método del cual se hablará más adelante.

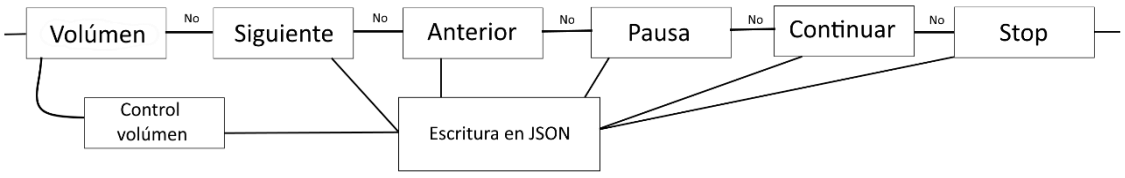

Ilustración 13: Comprobaciones de los condicionales y su orden

```
peticiones = r.recognize_google(audio, language = 'es_ES')
if 'volumen' in peticiones: # Si se cumple, entra

   raux = re.search(r'\d+', peticiones) # Busca en la variable
peticiones los números
   aux = raux.group()

   if aux >= 100: # Si el número es superior, se iguala a 100
      aux = 100

   data["volumen"] = aux # Se almacena el valor
   with open("acciones.json", "w") as jsonFile:
        json.dump(data, jsonFile) # Se escribe el valor del volúmen en
el fichero Json
```

```
    voz()

if 'siguiente' in peticiones: # Si se cumple, entra

    data["siguiente"] = 1 # Se almacena el valor
    with open("acciones.json", "w") as jsonFile:
        json.dump(data, jsonFile) # Se escribe el valor del volúmen en
el fichero Json
    voz()

if 'anterior' in peticiones: # Si se cumple, entra

    data["anterior"] = 1 # Se almacena el valor
    with open("acciones.json", "w") as jsonFile:
        json.dump(data, jsonFile) # Se escribe el valor del volúmen en
el fichero Json
    voz()

if 'pausa' in peticiones: # Si se cumple, entra

    data["pausa"] = 3 # Se almacena el valor
    with open("acciones.json", "w") as jsonFile:
        json.dump(data, jsonFile) # Se escribe el valor del volúmen
en el fichero Json
    voz()

if 'continuar' in peticiones: # Si se cumple, entra

    data["pausa"] = 4 # Se almacena el valor
    with open("acciones.json", "w") as jsonFile:
        json.dump(data, jsonFile) # Se escribe el valor del volúmen
en el fichero Json
    voz()

if 'stop' in peticiones: # Si se cumple, entra

    data["stop"] = 1 # Se almacena el valor
    with open("acciones.json", "w") as jsonFile:
        json.dump(data, jsonFile) # Se escribe el valor del volúmen
en el fichero Json
    voz()
```

En el caso de las peticiones de música, el condicional revisa primero si se hace una petición de *PlayList*, en caso de ser así, se procesa el contenido que va después de la palabra *PlayList* ya que, según el criterio definido, lo siguiente a dicha palabra ha de ser el artista o grupo. El procesado de los datos de la variable es almacenado en otra variable que se adjuntará como argumento en la creación del hilo, que llamará al método buscar. También se escuchará por los altavoces la petición, para hacer saber al usuario que todo funciona correctamente.

```
if 'playlist' in peticiones:

    r1 = re.search(r"playlist(?:[^a-zA-Z'-]+[a-zA-Z'-]+){0,5}",
peticiones)
    aux = r1.group()
```

```
    voices = engine.getProperty('voices')
    engine.setProperty('voice', voices[1].id) # Selección idioma,
castellano
    engine.say("Te pongo la Playlist de : "+aux[12:]) # Texto que se
utiliza
    engine.runAndWait()
    time.sleep(0.5)

     #Llamada mediante hilo demonio a la PlayList
     t = threading.Thread(target=youtube_search_playlist, name="buscar"
, args=(aux[12:],))
     t.daemon
     t.start()
     t.stop()
```

Si solo se quiere escuchar una canción, la condición de *PlayList* no se cumple y pasará a la siguiente condición, que buscará si se encuentra la palabra escuchar en el texto. En caso de ser así, después de dicha palabra se procesará el contenido de la variable, para extraer el título y artista de la canción que se quiere escuchar. De igual modo se escuchará por los altavoces la petición realizada para hacer saber que todo funciona correctamente.

```
r1 = re.search(r"escuchar(?:[^a-zA-Z'-]+[a-zA-Z'-]+){0,5}",
peticiones)
aux = r1.group()
voices = engine.getProperty('voices')
engine.setProperty('voice', voices[1].id)    # Selección idioma,
castellano
engine.say("Te pongo la cancion: "+aux[9:]) # Texto que se utiliza
engine.runAndWait()
time.sleep(0.5)

t = threading.Thread(target=youtube_search, name="buscar" ,
args=(aux[9:],))
t.daemon
t.start()
t.stop()
```

En caso de no reconocer ninguna de las palabras o frases anteriores, se modificará el valor del contenido de la variable que almacena la pausa de la música y se modificará en el fichero JSON. Así podrá ser leído por el método de reproducción de música que continuará con la misma y se llamará al método 'voz()' que espera la palabra clave, que activará el sistema.

```
data["pausar"] = "4"

with open("acciones.json", "w") as jsonFile:
     json.dump(data, jsonFile)

voz()
```

2.2.5 *Búsqueda música*

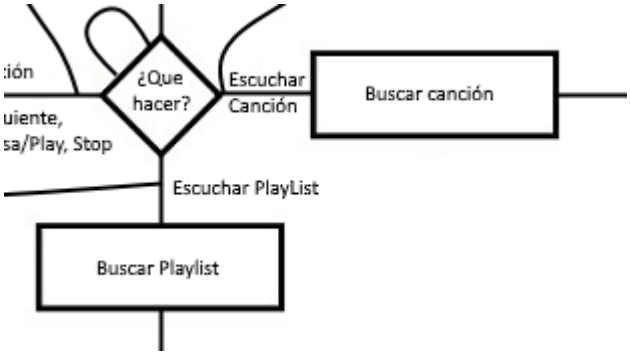

Ilustración 14: Parte del diagrama flujo (Búsqueda música)

La búsqueda de música emplea dos métodos diferentes. Uno para la búsqueda de una sola canción y otro para la búsqueda de una *PlayList*. Esta decisión se tomó por su practicidad, ya que en un primer momento pueden parecer similares, pero cuando se analiza la forma que tiene YouTube de tratar la información son diferentes.

Cuando se trata de una sola canción, YouTube proporciona como resultado de la búsqueda un enlace directamente del vídeo que se desea escuchar. Para poder realizar la búsqueda es necesario el uso del módulo de YouTube para Python3 descrito en el apartado teórico de este mismo documento. También es necesario el uso de una clave de desarrollador, un nombre de servicio de API y tipo de versión de API a usar. La obtención de todos estos datos se encuentra en el punto 2.1 Obtención de la clave de API.

Para la búsqueda de una canción, es necesario definir una serie de opciones: nombre de la canción, máximos resultados en este caso uno, el tipo que será vídeo y el orden que será relevancia.

```
youtube = build(YOUTUBE_API_SERVICE_NAME, YOUTUBE_API_VERSION,
developerKey=DEVELOPER_KEY)
search_response = youtube.search().list(
      q = options,
      part='id,snippet',
      maxResults=1,
      type='video',
      order='relevance'
).execute()
```

El resultado se filtrará para obtener el identificador de vídeo. Que es el único cambio, porque el resto de la URL es idéntico para cualquier vídeo. También se revisará si existe una reproducción en curso, sin importar que sea una canción o una *PlayList*. En caso de ser así, se modificará la variable de control, y se accederá al hilo, para hacerse efectivo el cambio. Al detenerse la reproducción, el hilo se cierra automáticamente y retornará al instante del programa

en el que estaba antes de entrar al hilo. A continuación, se volverá a crear un hilo de reproducción adjuntado el identificador del vídeo buscado.

```
videos

for search_result in search_response.get('items', []):

    #Se filtra la variable para sacar el ID
    if search_result['id']['kind'] == 'youtube#video':
       videos.append('%s' % (search_result['id']['videoId']))

if videos:

   global stop
   for t in threading.enumerate(): # Se revisa si existe hilo de
reproducción

       if t.getName() == "reproductor": # En caso de ser así se cierra
          stop = "True"
          t.join()

   # Se crea un hilo nuevo de reproducción
   t = threading.Thread(target=pafy_video, args=(videos[0],),
name="reproductor")
   t.daemon = True
   t.start()
```

Para la búsqueda de una *PlayList*, en las opciones de búsqueda hay que definir el tipo, para que busque una *PlayList*, todo en minúsculas, siendo el resto de las opciones iguales. Para filtrar el resultado, es necesario filtrar por el **"playlistId"** y se comprobará si existe una reproducción en curso y de ser así se detendrá, tal y como se definió anteriormente. En una variable se almacena la URL completa de la *PlayList* y se crea un hilo de reproducción adjuntando la variable.

```
playlists
for search_result in search_response.get('items', []):

    # Se filtra para obtener la playlistId
    if search_result['id']['kind'] == 'youtube#playlist':
      playlists.append('%s' % (search_result['id']['playlistId']))

    global stop
    for t in threading.enumerate():# Se revisa si existe hilo de
reproducción

        if t.getName() == "reproductor": # En caso de ser así se
cierra
           stop = "True"
           t.join()
    # Variable con el enlace completo
    url =
"https://www.youtube.com/playlist?list={0}".format(playlists[0])
```

```
      # Se crea un hilo nuevo de reproducción
      t = threading.Thread(target=pafy_playlist, args=(url,),
name="reproductor")
      t.daemon = True
      t.start()
```

2.2.6 *Reproducción música*

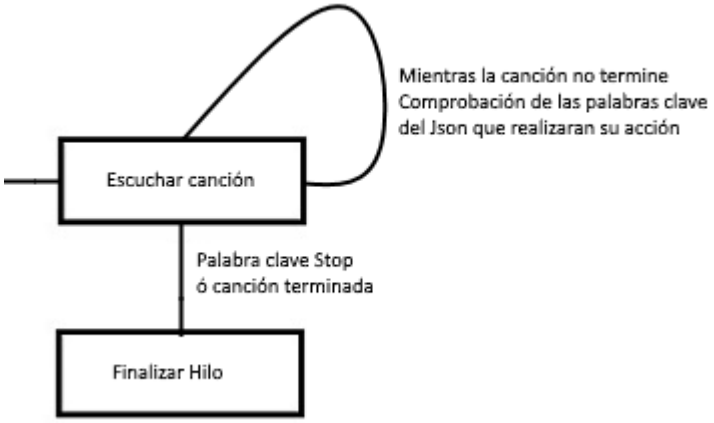

Ilustración 15: Parte diagrama de flujo (Reproducción canción)

Para la reproducción de una canción se emplearán los módulos, Pafy y VLC. Aunque todo se puede controlar por comandos de voz, mediante las acciones descritas anteriormente. También se han programado pulsadores para poder realizar las mismas acciones mediante ellos. También es necesario la lectura y escritura del fichero JSON, para las modificaciones que se realizan mediante las acciones descritas en el apartado 2.2.4 Reconocimiento de voz (Acciones). La duración de la canción es almacenada, ya que es necesario saber cuánto dura para terminar el bucle de reproducción de la canción y así reinicializar el sistema.

```
with open("replayScript.json", "r") as jsonFile: # Lectura del Json
        data = json.load(jsonFile)
volumen = int(data["volumen"]) # Se almacena el valor de volumen

video = pafy.new(url)
duration = video.length # Se obtiene la duración del video
best = video.getbestaudio()
playurl = best.url
ins = vlc.Instance()
player = ins.media_player_new() # Instancia para reproducir
player.audio_set_volume(volumen)
code = urllib.request.urlopen(url).getcode()
Media = ins.media_new(playurl)
Media.get_mrl()
player.set_media(Media)
player.play() # comienza la reproducción
```

La reproducción será controlada mediante un bucle que revisará la duración de la canción o si se va a reproducir de una nueva canción o una *PlayList*, para terminar el bucle. Dentro del bucle, se controlará mediante condicionales, si existe alguna modificación en el fichero JSON y si es accionado algún pulsador.

Mediante la pulsación de volumen, los valores se modifican sumando diez o restando diez, al valor que tenga en ese momento la variable que controla el volumen siendo el valor máximo cien y mínimo cero. Se tomó esa decisión para ver un cambio significativo y rápido, porque desplazar el volumen de uno en uno, puede ser cansado para el usuario.

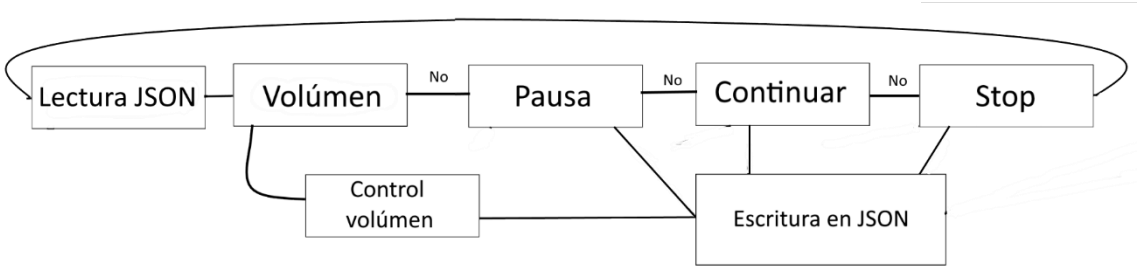

Ilustración 16: Funcionamiento acciones durante la reproducción de música

```
while i != duration*2 and stop == "False":

      with open("replayScript.json", "r") as jsonFile: # Lectura del
fichero Json
            data = json.load(jsonFile)

      pausa = int(data["pausar"]) # Se almacenan los datos del fichero
      vol = int(data["volumen"]) # Se almacenan los datos del fichero
      s = int(data["stop"]) # Se almacenan los datos del fichero

        # Control de volúmen
        StatusVolMas = GPIO.input(16)
        StatusVolMenos = GPIO.input(20)

        if volumen is not vol:
           print("volumen_musica:", vol)
           volumen = vol
           player.audio_set_volume(volumen)

        if StatusVolMas == False and volumen < 100:

           volumen = volumen + 10
           vol = volumen
           data["volumen"] = volumen

           # Al modificase el valor, hay que escribir en el fichero
Json
           with open("replayScript.json", "w") as jsonFile:
               json.dump(data, jsonFile)
           player.audio_set_volume(volumen)
```

```
        elif StatusVolMenos == False and volumen > 0:

            volumen = volumen - 10
            vol = volumen
            data["volumen"] = volumen

            # Al modificase el valor, hay que escribir en el fichero
Json
            with open("replayScript.json", "w") as jsonFile:
                json.dump(data, jsonFile)
            player.audio_set_volume(volumen)


        # Se crean variables para controlar las pulsaciones
        StatusPausa  = GPIO.input(23)
        StatusStop = GPIO.input(24)

        if StatusPausa  == False and pausa == 1:

            pausa = 2
            player.pause()

        elif StatusPausa == False and pausa == 2:

            pausa = 1
            player.play()

        elif StatusStop == False or s == 1:

            data["stop"] = "0"

            # Al modificase el valor, hay que escribir en el fichero
Json
            with open("replayScript.json", "w") as jsonFile:
                json.dump(data, jsonFile)

            stop == "True"
            player.stop()

        if pausa == 1:
            i = i+1
            time.sleep(0.5)
        else:
            time.sleep(0.5)

stop = "False"
player.stop()
```

Dentro del bucle anterior se encuentra una variable de control por condicional que tiene como utilidad, la pausa de la canción que se esté reproduciendo. Cuando se dice la palabra clave la variable cambia de valor y pausa la reproducción de música., para poder realizar los descrito en el punto 2.2.4 Reconocimiento de voz (Acciones).

```
if pausa == 3: # Según el valor de pausa, se realiza una acción u otra

    player.pause()
    data["pausar"] = "2"

    with open("replayScript.json", "w") as jsonFile:
        json.dump(data, jsonFile)

if pausa == 4:

    player.play()
    data["pausar"] = "1"

    # Al modificase el valor, hay que escribir en el fichero Json
    with open("replayScript.json", "w") as jsonFile:
        json.dump(data, jsonFile)
```

Para la reproducción de una *PlayList* será necesario el uso de la clave de desarrollador, porque se realizará una consulta utilizando el identificador de la *PlayList*, para obtener los identificadores de los vídeos.

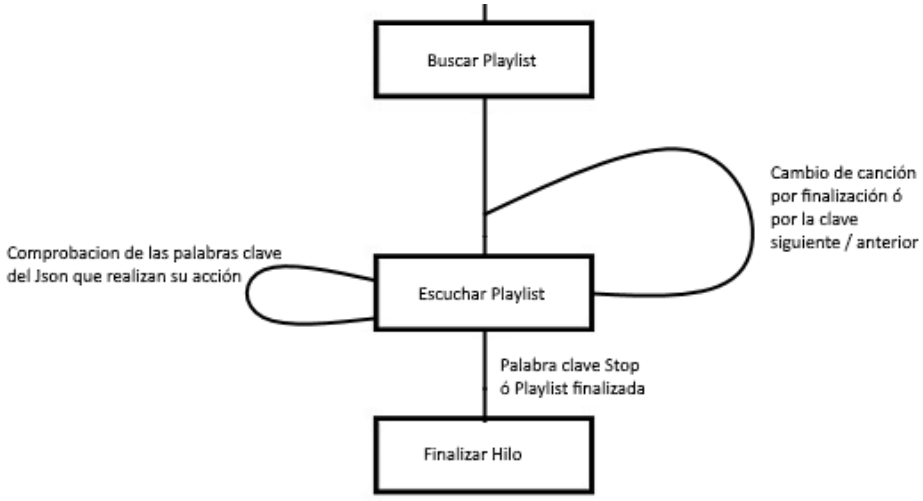

Ilustración 17: Parte diagrama de flujo (Reproducción *PlayList*)

```
url = [] # Variable para almacenar los identificadores de los videos
query = parse_qs(urlparse(playListUrl).query, keep_blank_values=True)
playlist_id = query["list"][0]

youtube = googleapiclient.discovery.build("youtube", "v3",
developerKey = DEVELOPER_KEY)
# Petición de la lista de videos de la PlayList
request = youtube.playlistItems().list(
            part = "snippet",
            playlistId = playlist_id,
            maxResults = 50
)
```

```
response = request.execute() # Resultado de la petición

playlist_items = []
while request is not None: # Procesado de los resultados
      response = request.execute()
      playlist_items += response["items"]
      request = youtube.playlistItems().list_next(request, response)

for link in playlist_items: # Almacenamos los enlaces completos de
cada canción

    url.append('%s' %
(f'https://www.youtube.com/watch?v={link["snippet"]["resourceId"]["vid
eoId"]}'))
```

Una vez se tienen todos los enlaces, el funcionamiento de la reproducción es similar al de la reproducción de una canción, sin embargo, al tratarse de más de una canción se añaden las condiciones que controlan los pulsadores y acciones para pasar a la siguiente canción o la canción anterior. De igual manera se controlará la duración de cada canción mediante un bucle, con la salvedad de que, cuando la canción termine, pasa a la siguiente canción, hasta que se termine la lista de canciones.

```
# Lectura del fichero Json
with open("replayScript.json", "r") as jsonFile:
     data = json.load(jsonFile)

# Se almacena en diferentes variables la información del Json
pausa = int(data["pausar"])
vol = int(data["volumen"])
next = int(data["siguiente"])
prev = int(data["anterior"])
s = int(data["stop"])

# Se crean variables para controlar la pulsación.
StatusPausa = GPIO.input(23)
StatusNext = GPIO.input(21)
StatusPrev = GPIO.input(25)
StatusStop = GPIO.input(24)
StatusVolMas = GPIO.input(16)
StatusVolMenos = GPIO.input(20)
```

Los condicionales que controlan las acciones y las pulsaciones son similares a la reproducción de una canción, con el añadido de los pulsadores de las acciones para pasar a la siguiente canción o a la anterior canción.

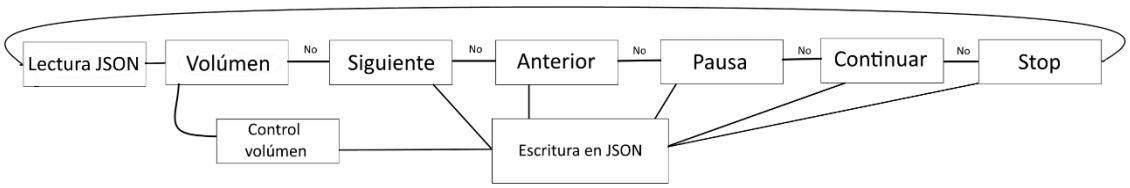

Ilustración 18: Funcionamiento acciones durante la reproducción de música

```
# Control sobre la pulsación para el volumen, y que no exceda de 100
el valor de la variable
if StatusVolMas == False and volumen < 100:

    volumen = volumen + 10
    vol = volumen
    data["volumen"] = volumen

    with open("replayScript.json", "w") as jsonFile:
         json.dump(data, jsonFile)
     player.audio_set_volume(volumen)

# Control sobre la pulsación para el volúmen, y que no sea inferior a
0 el valor de la variable
elif StatusVolMenos == False and volumen > 0:

    volumen = volumen - 10
    vol = volumen
    data["volumen"] = volumen

    with open("replayScript.json", "w") as jsonFile:
         json.dump(data, jsonFile)
    player.audio_set_volume(volumen)

# Control de la pulsación de pausa, la acción de pausar y la
activación de la palabra clave
if StatusPausa  == False and pausa == 1 or pausa == 3:

    data["pausar"] = "2"

    with open("replayScript.json", "w") as jsonFile:
         json.dump(data, jsonFile)

    pausa = 2
    player.pause()


elif StatusPrev == False or prev == 1:

    data["anterior"] = "0"

    with open("replayScript.json", "w") as jsonFile:
         json.dump(data, jsonFile)

    stop = "True"
    nextprev = 1
    i = i - 2
elif StatusStop == False or s == 1:

    data["stop"] = "0"

    with open("replayScript.json", "w") as jsonFile:
         json.dump(data, jsonFile)

    stop = "True"
```

```
    i = i + len(url)

if pausa == 1:

    j = j + 1
    time.sleep(0.5)

else:

    time.sleep(0.5)

if volumen is not vol:

    volumen = vol
    player.audio_set_volume(volumen)
```

También se controla mediante condicionales la activación de las acciones o peticiones de una canción o una *PlayList*. En este caso, al accionar el pulsador de Play/Pausa, se pausará la canción y si está pausada se volverá a reproducir desde el punto pausado.

```
# Control de la pulsación de pausa, la acción de reproducción y la no
acción.
elif StatusPausa == False and pausa == 2 or pausa == 4:

    data["pausar"] = "1"

    with open("replayScript.json", "w") as jsonFile:
         json.dump(data, jsonFile)

    pausa = 1
    player.play()

elif StatusNext == False or next == 1:

    data["siguiente"] = "0"
    with open("replayScript.json", "w") as jsonFile:
         json.dump(data, jsonFile)

    stop = "True"
    nextprev = 1
```

2.3 Autoarranque de la aplicación

Para poder realizar la ejecución automática en el arranque de la Raspberry Pi, es necesario realizar un Script de Linux y añadirlo en el fichero de inicio del sistema operativo. Para ello en el editor de texto se crea un fichero llamado **"script.sh"** que se ubicará en la carpeta de usuario, por facilidad de acceso, ya que es necesario saber su ruta para añadirla en el fichero de arranque. El contenido del fichero es el siguiente.

```
#!/bin/Bash
# Antes de ejecutar el programa, se espera 60 segundos.
sleep 60 && lxterminal 'cd /home/pi/"CARPETA PROGRAMA" && sudo python3
"PROGRAMA.py"'
```

Se realiza la ejecución mediante un terminal, para así poder ver mediante escritorio compartido los datos de ejecución del programa, ya que el sistema carece de pantalla propia. Aunque no es necesario, se ha realizado así para ver toda la ejecución y controlar posibles errores, por alguna actualización de algún módulo, ya que este tipo de errores se han producido a lo largo de la realización de este proyecto.

Para poderlo ejecutar, como se describió en la parte teórica es necesario hacer el fichero ejecutable con el siguiente comando **"chmod +x script.sh"**. Una vez hecho esto se puede realizar una ejecución para comprobar que el Script funciona correctamente.

Para poderlo programar en el arranque es necesario editar el fichero de arranque de Linux ubicado en la siguiente ruta **"/etc/xdg/lxsession/LXDE-pi/autostart"** y se añadirá de la siguiente manera **"@/home/pi/script.sh"**. Una vez se reinicie la Raspberry Pi y transcurrido un minuto para que el dispositivo Wifi se inicialice correctamente se ejecutará la aplicación del proyecto.

2.4 Código en Android

La aplicación de configuración de red Wifi para Android se ha diseñado de forma sencilla, como único objetivo el poder configurarlo de manera rápida y sin complicaciones. Por tanto, cuenta con lo básico que son los textos que definen dónde hay que escribir el nombre de la red Wifi y su contraseña de acceso y dos botones uno para conectar con el Bluetooth de la Raspberry Pi y otro para enviar la información. En el fichero **"AndroidManifest.xml"** se ha configurado para que la orientación de la pantalla sea del tipo **"portrait"**, de forma que el espacio situado debajo de los botones permite que cuando se desee escribir el nombre de la red Wifi o la contraseña pueda salir el teclado virtual sin superponerse sobre los botones.

A continuación se muestra el diagrama de flujo, del programa.

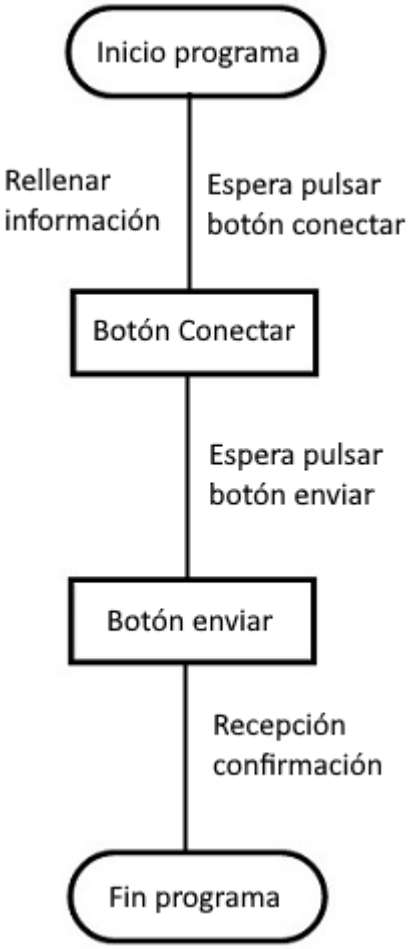

Ilustración 19: Diagrama de flujo programa Android

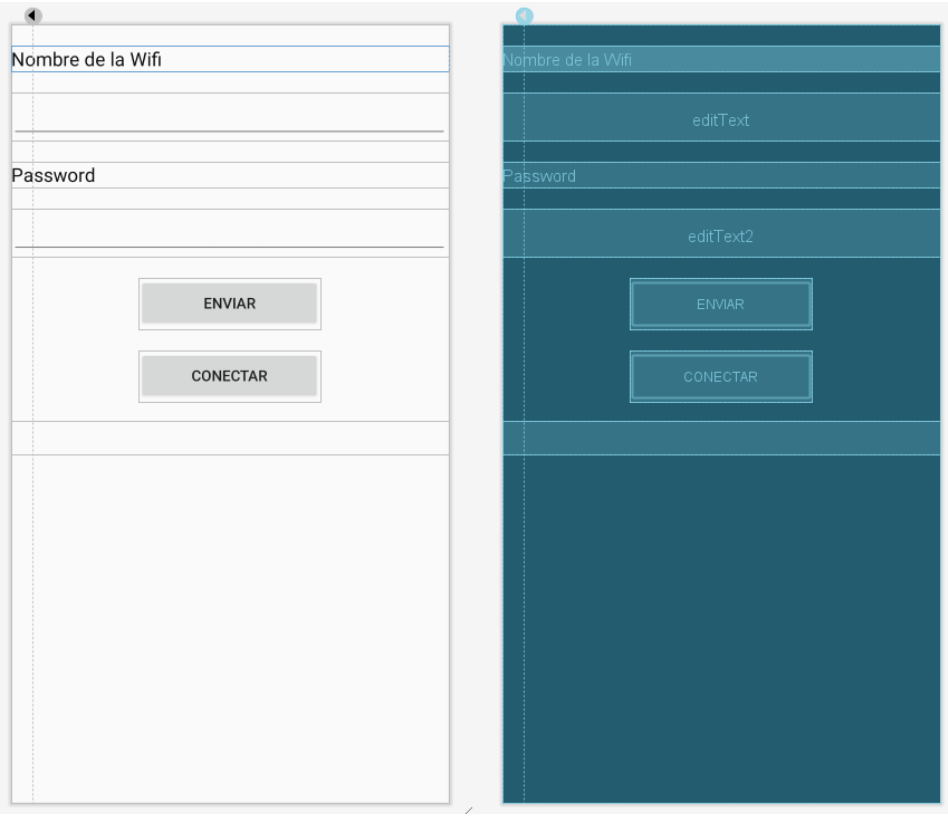

Ilustración 20: Entorno gráfico aplicación Android

Para poder acceder al Bluetooth del sistema Android es necesario solicitar permisos de acceso, dicha solicitud se realiza desde el fichero **"AndroidManifest.xml"** de la siguiente forma.

```
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
```

Para el uso de los sistemas del dispositivo móvil es necesario crear las variables que definen su uso, una de ellas define el UID que ha de ser el mismo que el del código de la configuración Wifi en la Raspberry Pi, ya que si no es el mismo la conexión no será posible. Al ser algo preconfigurado en el código de ambos sistemas no existe posibilidad de ser modificado.

```
private static final UUID MY_UUID_INSECURE =
        UUID.fromString("94f39d29-7d6d-437d-973b-fba39e49d4ee");

private static final int REQUEST_ENABLE_BT = 1;
BluetoothAdapter bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
private BluetoothDevice mmDevice;
private UUID deviceUUID;
ConnectedThread mConnectedThread;
private Handler handler;
```

```
String TAG = "MainActivity";
EditText send_data_name;
EditText send_data_pass;
TextView view_data;
StringBuilder messages;
String deviceName;
String namedevice="raspberrypi";

setContentView(R.layout.activity_main); // Muestra el entorno gráfico de la
aplicación

// Definición de variables de captura de texto
send_data_name = (EditText) findViewById(R.id.editText);
send_data_pass = (EditText) findViewById(R.id.editText2);
// Definición de variable para mostrar texto
view_data = (TextView) findViewById(R.id.textView);
// Comprobación esta activado el Bluetooth
if (bluetoothAdapter != null && !bluetoothAdapter.isEnabled()) {
    Intent enableBtIntent = new
            Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
    startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
```

Una vez el sistema está listo para su uso, el dispositivo móvil se queda a la espera para conectar con la Raspberry Pi. Pulsado el botón de conectar, genera una orden del tipo **"onClick"** que llama a la función para realizar la conexión. Dicha función revisa los dispositivos Bluetooth a los que se puede conectar, y buscará dentro de la lista la Raspberry Pi entre ellos el que tiene como nombre **"raspberrypi"**. Mediante un bucle se buscará el dispositivo Bluetooth de la Raspberry Pi para obtener su nombre y MAC. Una vez se tiene dichos datos, se realiza la llamada a la función que configura la conexión.

```
Object[] devices = pairedDevices.toArray();
for (int i = 0; i < pairedDevices.size(); i++) { // Se busca en la lista

    BluetoothDevice device = (BluetoothDevice) devices[i];
    deviceName = device.getName(); // Se obtiene nombre del dispositivo

    if (deviceName.equals(namedevice)){

        devicepair = (BluetoothDevice) devices[i]; // Dirección MAC
    }
}
// Se realiza la conexión
ConnectThread connect = new ConnectThread(devicepair, MY_UUID_INSECURE);
connect.start();
```

Con los datos que se le envía a la función **"ConnectThread**" se configura el dispositivo móvil para realizar la conexión con la Raspberry Pi, creando un socket con el UID y la MAC. Una vez creado el socket con los datos almacenados en **"mmSocket"** se pasa a la función para crear el hilo que realizará la conexión.

```
mmDevice = device;
deviceUUID = uuid;
BluetoothSocket aux = null;

// BluetoothSocket para una conexión con el dispositivo Bluetooth
aux = mmDevice.createRfcommSocketToServiceRecord(MY_UUID_INSECURE);
mmSocket = aux;
mmSocket.connect();

// Hilo para administrar la conexión y realizar transmisiones
mConnectedThread = new ConnectedThread(mmSocket);
mConnectedThread.start();
```

En este punto la aplicación está a la espera de enviar el nombre de la red Wifi y la contraseña lo que realizará una vez se pulse el botón de enviar, el cual mediante una orden **"onClick"** que llama a la función que crea una variable de tipo **"String"** con los datos de la red Wifi. Para poder ser enviado se convierte a una variable tipo **"byte"**. Una vez los datos son enviados la aplicación permanecerá a la espera para recibir la confirmación de recepción de datos, que se mostrara en pantalla para que el usuario sepa que todo ha funcionado correctamente.

```
String data = send_data_name.getText().toString() + "," +
send_data_pass.getText().toString();

byte[] bytes = data.getBytes(Charset.defaultCharset());
mmOutStream.write(bytes);
```

Capítulo 3. Construcción del prototipo

El prototipo ha sido diseñado como complemento a la aplicación, porque todo el conjunto fue pensado para trabajar al unísono. Se pensaron diferentes diseños, inspirados en diferentes reproductores de música existentes en el mercado y se escogió el más adecuado en cuanto a practicidad y elegancia, ya que un diseño más cuadrado es bastante obsoleto y muy utilizado.

Aunque también es cierto que el diseño ha provocado algunas complicaciones a la hora de integrar todos los componentes, debido a que algunos de éstos como en el caso batería, no coincidían sus dimensiones descritas con las reales y se tuvo que colocar dentro del prototipo cuando en realidad se pensó colocar bajo el pie, para darle más estabilidad por su peso y así bajar el centro de gravedad de todo el conjunto.

En el siguiente punto se hablará de los materiales empleados para su construcción. El precio aproximado de los materiales se trató de que no excediera de cien euros sin contabilizar el coste de la Raspberry Pi 4, porque este componente, ya se compró para el uso en otros proyectos y por tanto no será contabilizado. En caso de contabilizarse, es probable que no fuese necesario el uso de una Raspberry Pi 4 con unas especificaciones tan elevadas, aunque con una Raspberry Pi con especificaciones inferiores al modelo 3 no puede ser útil, ya que carecen de Wifi y Bluetooth o simplemente carecen de potencia suficiente. En el punto 4.1 Raspberry Pi Zero a Raspberry Pi 4, se hablará de un problema relacionado con el uso de una Raspberry Pi Zero W y su carencia de potencia.

Ilustración 21: Prototipo terminado y en funcionamiento

3.1 Materiales

Se pensó fabricar el prototipo integro de diferentes materiales, pero tras pensar los diferentes diseños en relación con los materiales, se optó por el uso de PVC como material principal, sin embargo, para la colocación de los altavoces se empleó madera para darle mejor vistosidad. Pero al tratarse de un prototipo está pensado para poderse desmontar en pocos minutos y así poder revisar cualquier posible problema a la hora de integrar todos los componentes.

El coste de los materiales tanto del PVC como de la madera ha sido bastante reducido. El PVC es tubería de desagüe que se vende en secciones de un metro y la madera está compuesta de contrachapado de 40 x 60 cm.

Referente a los altavoces, se buscó un conjunto que pudiera ofrecer una calidad de sonido, dentro del presupuesto y las dimensiones del prototipo, por ello se pensó en una combinación de altavoces de gama completa de 2’5 pulgadas. Para complementar las altas frecuencias, se usa una pareja de Tweeters. También se pensó en complementar el sonido de baja frecuencia, por el tamaño reducido de los altavoces. De esta forma se adquirió una pareja de membranas a modo de altavoces pasivos, que mediante la presión interna generaran movimiento en las membranas y el consecuente sonido, no obstante, debido a que el prototipo no es cien por cien estanco, el funcionamiento de las membranas no es del todo correcto. A pesar de ello sí se detecta vibración, haciendo pensar que, si fuese estanco, podrían realizar su función de una manera óptima.

El sistema necesita un amplificador externo con una potencia no superior a los diez vatios, porque la potencia de sonido que aporta la Raspberry Pi sólo es capaz de funcionar con auriculares o con sistemas auto amplificados y según especificaciones del vendedor la potencia máxima que son capaces de gestionar los altavoces es de 10 vatios. Por tanto, se tiene que amplificar la salida de sonido.

Ilustración 22: Sistema de sonido

El coste del sistema de sonido se trató de reducir el máximo posible para ajustarse al presupuesto preestablecido.

- Altavoces de rango completo: 13,64€
- Tweeter: 7,74€
- Altavoces pasivos: 6,21€
- Amplificador: 2,41€
- Total: 30€

Uno de los beneficios de este proyecto es su portabilidad, para ello es necesario el uso de una batería con la capacidad de soportar una Raspberry Pi 4 con un consumo máximo de tres amperios más el sistema de sonido, el cual tiene un voltaje de doce voltios.

Durante la revisión de los componentes, se buscó un amplificador de cinco voltios, pero con esa tensión de alimentación no se puede conseguir la potencia deseada, por lo que la batería debe tener un valor superior y para obtener la tensión de alimentación de la Raspberry Pi se emplea un regulador a 5V.

Se estableció que la mejor opción sería la de una batería de 12V. Se buscó una batería con capacidad suficiente para poder soportar durante largo tiempo todo el sistema, así que su capacidad es de nueve mil ochocientos miliamperios. En el apartado 4.7 Batería del prototipo se explica la razón de por qué no se compraron las piezas sueltas de la batería y se ensambló.

Ilustración 23: Batería y transformador

El coste de estos materiales es el siguiente:

- Batería 12v 9800mAH: 15,88€
- Transformador 12v a 5V 10A: 6,70€
- Total: 22,58€

El resto de los materiales que se compraron para el prototipo, consiste en pulsadores, para realizar las acciones sin necesidad de comandos de voz, como una alternativa. Leds RGB para controlar el funcionamiento de la aplicación de la Raspberry Pi. Un interruptor para realizar la función como interruptor general del sistema de alimentación. Una pantalla LCD para medir la capacidad de la batería mediante una representación en tanto por cien y el cableado para realizar las conexiones.

Ilustración 24: Materiales varios

El coste de estos materiales oscila porque algunos los materiales no tienen un elevado coste, pero en su conjunto como los pulsadores, sí que es considerable.

- Pulsadores: 20,44€
- Led RGB pack de 10: 1,22€
- Pantalla LCD: 3,20€
- Cableado vario: 10€
- Total: 34,86€

El coste total de todos los materiales es el siguiente.

- Coste sistema sonido: 30€
- Coste del sistema de alimentación: 22,58€
- Coste del resto de componentes: 34,86€
- Coste de materiales para la construcción: 6€
- Coste total: 93,44€

Como se dijo anteriormente no se ha tenido en consideración el coste de la Raspberry Pi 4, ni tampoco el coste del micrófono, ya que son componentes que se habían adquirido para otros proyectos. En caso de tener en cuenta los costes de dichos componentes la cuantía podría llegar a la cifra de los doscientos euros, porque una Raspberry Pi 4 de 4GB en el momento de la compra tenía un coste de sesenta y cinco euros sin incluir gastos de envió y el micrófono, cuando se compró tenía un coste de entre veinte y veinticinco euros. De igual forma únicamente se tiene en consideración los costes de los componentes utilizados en el proyecto, porque se probaron diferentes micrófonos, que sí que fue necesario adquirirlos, aunque al final el empleado fue uno ya comprado con anterioridad. En el apartado de problemas y soluciones se dan más detalles al respecto.

3.2 Construcción

Para realizar la construcción del prototipo, se plantearon diferentes diseños. Una vez se decidió el diseño y los materiales, se dibujó un diseño provisional sobre la tubería de PVC para comprobar si las dimensiones eran correctas, tanto de los materiales como de la longitud del tubo, ya que no se ha querido realizar un prototipo excesivamente grande, pero era necesario unas dimensiones mínimas para que todos los componentes que tienen que integrarse en el interior, cupieran. Por tanto, la longitud del tubo una vez cortado no supera los cuarenta centímetros, aunque con las tapas laterales puede llegar a incrementarse hasta unos cincuenta centímetros. La longitud con las tapas puede variar debido a la profundidad a la que se quieran poner.

Ilustración 25: Dibujo previo sobre la tubería

Una vez se ha comprobado que las dimensiones sobre el tubo son las adecuadas y que todo ajusta a la perfección, se procede al corte de la "ventana", donde se colocará la madera en la que irán los altavoces y los Tweeters. Para realizar el corte se empleó una multiherramienta tipo Dremel, que permite realizar cortes y lijados con bastante precisión. En la búsqueda de un soporte para la tubería, se llegó a la conclusión que la misma pieza que se cortó puede ser utilizada como base, ya que encontrar una pieza con las mismas dimensiones del tubo resulta complicado y al colocarse en la base del tubo la pieza cortada, sorprendentemente, mejoraba el diseño. Aunque se anclará mediante tornillos, porque todo está planteado para poderse desmontar fácilmente.

Ilustración 26: Corte de la "ventana" para los altavoces

El siguiente paso en la construcción es la preparación de la madera, para ello se toman las medidas de la "ventana" y se plasma sobre la madera. Una vez las medidas están dibujadas sobre la madera, se dibuja el contorno de los altavoces y los Tweeters, dejando la distancia suficiente para mantener la consistencia de la madera, ya que el peso de los altavoces es considerable y ha de tener la suficiente solidez, para no romperse.

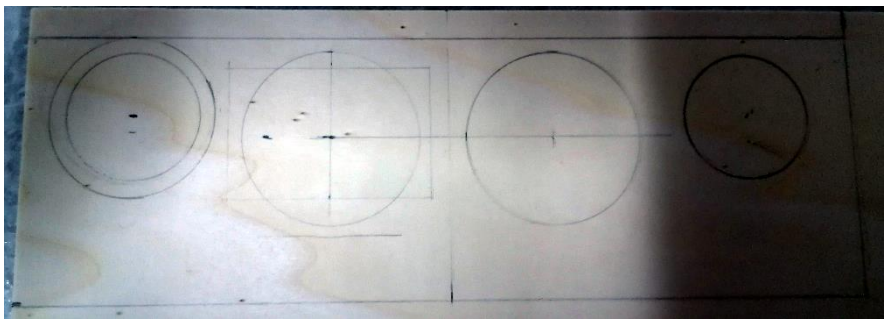

Ilustración 27: Dibujo sobre la madera

Una vez realizado todo el dibujo y comprobado que se mantiene la solidez de la madera se procede a su corte para realizar la pieza y tener los huecos de los altavoces y lijarlos para ajustarlos a las dimensiones de los altavoces. Para cortar la madera y realizar los huecos, se emplea una sierra de calar y un taladro con herramienta de corte circular. Porque la multiherramienta quemaba la madera y resultaba peligroso. Pero para el lijado de los bordes de los huecos para los altavoces sí que se empleó la multiherramienta.

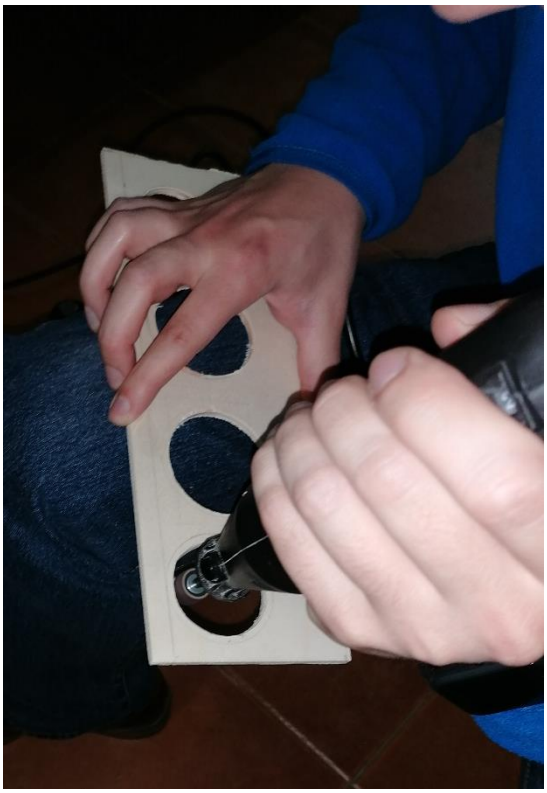

Ilustración 28: Lijado hueco altavoces

En las tapas laterales se quiere colocar los altavoces pasivos, para ello es necesario realizar el corte circular de las tapas. Se utiliza el taladro con la herramienta de corte circular empleado en la madera. Como no tienen nada para poderlos atornillar, sino que tienen un borde de goma, se decidió pegarlos directamente sobre las tapas. De igual forma se realizó el dibujo antes de realizar los cortes. A la hora de pegarlos se decidió ponerlos por fuera, debido al grosor del PVC y porque al ponerlos por la parte interior no se podían apreciar.

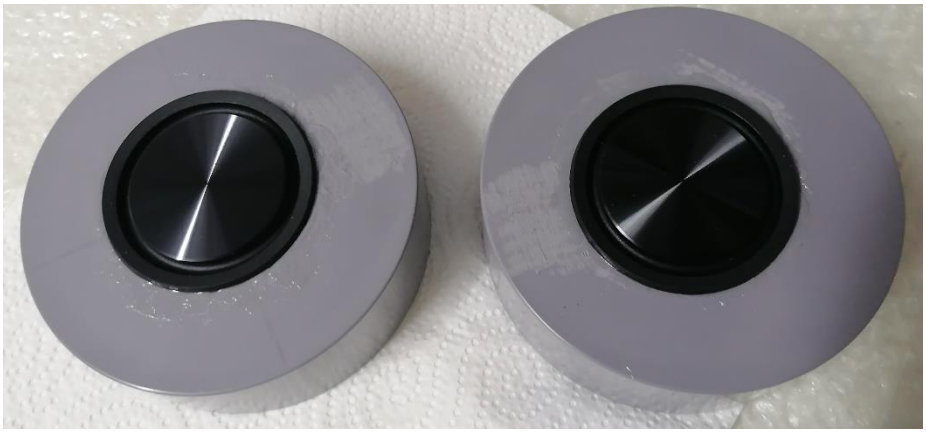

Ilustración 29: Altavoces pasivos pegados

La realización de los agujeros para colocar los pulsadores resultó realmente complicada, porque no se tenía ninguna herramienta que pudiera realizarlos con las dimensiones adecuadas, por tanto, se necesitó mediante la multiherramienta, realizar un agujero previo para después ir agrandando poco a poco, hasta alcanzar la dimensión adecuada. Se buscó la equidistancia y la simetría entre pulsadores. No fue necesario pegarlos, ya que tienen la posibilidad de ser atornillados siempre que el agujero se ajuste a las dimensiones. Porque los mismos pulsadores llevan rosca y tuerca.

Ilustración 30: Agujeros de los pulsadores

Ilustración 31: Codos sujeción madera lateral

Para tapar los laterales de la "ventana", donde va colocada la madera con los altavoces y los Tweeters, también se pensó utilizar diferentes materiales, pero como el uso de madera dio un resultado realmente bueno, se decidió que también podría ser un material adecuado. Para ello se cortaron dos piezas, una para cada lado, empleando la sierra de calar. Para poderlas fijar y que al mismo tiempo mantuvieran fijas la madera de los altavoces, se plantearon varias opciones de las cuales una fue la que se puso en práctica. Como uno de los objetivos del prototipo es la de tener un fácil despiece, se desarrolló la más acertada y disimulada. Consiste en emplear dos codos de metal pegados sobre la tubería, que después irían atornillados sobre la madera. Así se consigue una fijación sólida y discreta, pudiendo dejar caer la madera con los altavoces sobre la "ventana" y las maderas laterales serían las que sujetaran todo.

Antes de realizar un montaje previo y una prueba de sonido se tenía que soldar unos condensadores en los Tweeters porque carecían de ningún tipo de filtro paso alto, para restringir el paso de frecuencias inferiores 4 kHz, por ejemplo

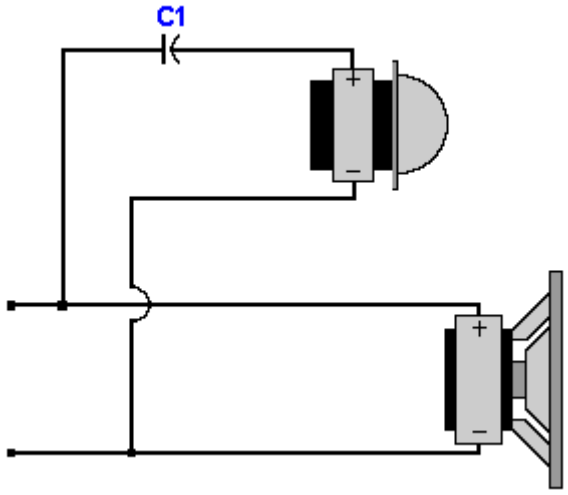

Ilustración 32: Cruce de primer orden

Se obtiene que la capacidad del condensador es de 6,625 μF. Así que se buscó un condensador para cada Tweeter que tuviera esa capacidad o fuera similar. Para los altavoces no era necesario ningún filtro, ya que serán de gama completa. Pero con los condensadores soldados y el resto de cableado listo, ya era posible realizar una prueba de sonido para comprobar su correcto funcionamiento.

Ilustración 33: Condensadores y cableado de los altavoces y Tweeters

Como se puede ver en la imagen se le aplicaron varias capas de barniz tanto a la madera de los altavoces como a la madera de los laterales para protegerla de la humedad ambiental, porque al no poder emplear elementos plásticos para el mismo fin que la madera, se pensó aplicar el barniz como aislante de la humedad y también darle robustez y un diseño más elegante.

Una vez realizado todo y teniendo éxito en la prueba de sonido se procede a realizar un montaje previo al montaje final para comprobar que todas las piezas encajan a la perfección.

Ilustración 34: Montaje previo

Una vez realizado el montaje previo, dentro del prototipo queda soldar el cableado de los pulsadores de la alimentación y comprobar la colocación de los componentes internos para que no se estorben unos a otros. De todas las tareas, la más complicada es la colocación, ya que todo ha de encajar perfectamente y tiene que quedar fijo en su lugar. Para ello se emplea cinta de doble cara y en caso de ser necesario unos topes colocados para que no se mueva nada.

Como es necesario pasar cables de la base al interior del prototipo, se realizó un agujero pasante que permite el paso del cable perteneciente al micrófono, los cables de alimentación de la pantalla LCD y los cables de los dos LED RGB que gestionan la interacción con el usuario.

La integración del micrófono no ha sido posible, porque el empleado se extrajo de otro proyecto y una vez defendido este, tiene que ser devuelto a su proyecto original. Además, la longitud del cableado del micrófono dificulta la integración en su interior.

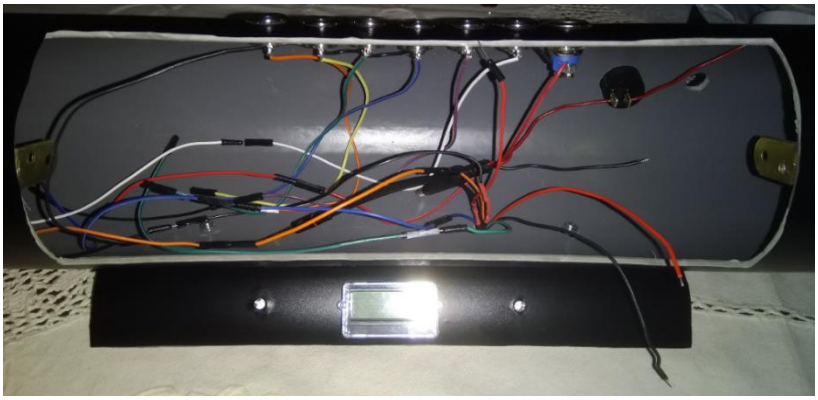

Ilustración 35: Cableado interno y base

Una vez pasados todos los cables se realizó una comprobación sobre los LED para que todas las soldaduras estuvieran correctas y se encendieran los pines con los colores correctos. Una vez realizada esa pequeña prueba de control ya sólo queda realizar el montaje de todos los componentes que van integrados en el interior del prototipo y realizar la prueba definitiva.

A la hora de montarlo todo es necesario hacerlo con delicadeza, ya que en la ilustración 35 únicamente se pueden ver los cables de los componentes integrados en el mismo prototipo, pero falta el cableado del sistema de sonido, la batería, el transformador y la Raspberry Pi con sus componentes, como es la tarjeta de sonido externa para conectarle el micrófono y el cableado que une los diferentes sistemas, que en algunos casos no se pudo realizar a medida y tienen una extensión excesiva. De este modo se realizó el montaje de todos los componentes con sumo cuidado y sin que ningún cable se quedara fuera o pinzado, para no provocar futuros problemas como cables pelados.

Se realizaron pruebas de ejecución de la aplicación con todos los componentes unidos, para comprobar que tanto los altavoces como los LED funcionaran correctamente. Fue necesario realizar algunos ajustes, ya que aparecieron problemas de ruido eléctrico, producido por la Raspberry Pi que, si se le da demasiado volumen al amplificador externo, puede llegar a ser molesto. Por tanto, se realizó un ajuste sobre el amplificador externo y sobre el micrófono, para reducir el ruido capturado. También se revisó antes de conectar todo al sistema de alimentación que la tensión fuera la adecuada, ya que la Raspberry Pi es un dispositivo bastante delicado y un pico de tensión al encender el sistema de alimentación puede causar fallos.

Una vez comprobado que todo el conjunto funcionaba correctamente y todo ha sido ajustado, para no causar problemas ni ningún fallo del sistema, se realizó una serie de pruebas de campo, que consistían en la ejecución del sistema y la prueba de la aplicación. Realizando para ello las peticiones de canción, y el control de volumen por comandos de voz y mediante la pulsación de los botones. También de la petición de una *PlayList* mientras está reproduciendo la canción de la petición anterior. De igual forma se comprobó el volumen de ambos métodos y de las acciones por comandos de voz y pulsación de los botones, para pasar a la siguiente canción, a la anterior canción y el paro de la reproducción de la *PlayList* en su totalidad. Todas las pruebas resultaron satisfactorias.

Ilustración 36: Prototipo en ejecución

Capítulo 4. Problemas y soluciones

4.1 Raspberry Pi Zero W a Raspberry Pi 4

El proyecto empezó en una Raspberry Pi Zero W porque era el único dispositivo del que se disponía en ese momento y para comenzar se consideraba una opción válida. El desarrollo se realizó por fases, de tal forma que, primero se pretendía buscar un programa con librerías de Python que permitiera la reproducción de música y un sistema que permitiera el acceso a sus servicios mediante un programa externo, para poder realizar la reproducción con el programa anteriormente mencionado.

Esas primeras pruebas dieron un resultado correcto tratándose de un hardware tan limitado como es una Raspberry Pi Zero W, sin embargo, los primeros problemas aparecieron, cuando se realizó otra parte del proyecto, en este caso el reconocimiento de voz. Al ejecutarlo, el periodo de carga de los dispositivos de sonido, en este caso el micrófono era demasiado largo. A pesar de ello, se consideró un problema menor ya que aún quedaba bastante para tener el programa desarrollado y los periodos de carga no se tenían en consideración.

No obstante, el principal problema que planteó una actualización del hardware fue la conjunción de ambas partes, lo cual producía problemas de sonido al reproducir música y al mismo tiempo estar el reconocedor de voz en segundo plano, provocando una serie de crepitaciones en el sonido de los altavoces y un alto consumo de la CPU, que no pudieron ser solventados de ninguna forma. Por tanto, se tomó la decisión de buscar alternativas con mayor potencia y memoria, tanto de almacenamiento como de RAM, ya que la Raspberry Pi Zero W, únicamente cuenta con un procesador de un núcleo 900 MHz y 512 MB de memoria RAM. El almacenamiento no suponía un problema porque este tipo de hardware permite mediante lector de tarjetas SD, ser incrementado fácilmente.

Vistos los buenos resultados de los dispositivos Raspberry, en un primer momento se pensó en un hardware de la misma marca, pero los elevados precios respecto a la competencia de otras marcas hicieron buscar alternativas más económicas. Se buscó un hardware de la compañía Orange Pi, Odroid y otras menos conocidas pero que compartían especificaciones. La marca que mejor cumplía con los requisitos era Orange Pi, ya que Odroid no tenía una placa en ese momento con unas dimensiones adecuadas. Así que se buscó un modelo de Orange Pi que cumpliera con los requisitos. Pero, aunque se encontró un modelo, la marca no disponía de distribución en España, al menos de manera oficial, con precios prefijados, y donde poder adquirir una placa pidiéndote lo que les parecía oportuno.

De tal forma que al final se optó por adquirir una Raspberry Pi 4, por las siguientes razones. Cumplía con las especificaciones sobradamente, con la posibilidad de ser utilizada en otros proyectos en el futuro. También tenía las medidas necesarias para caber en el interior del prototipo y el funcionamiento de su sistema operativo Linux y la forma en la que trabaja junto con el hardware. Aunque no deja de ser cierto, que no cumple con el requisito de ser una placa de prototipo económica, este hándicap queda suplido por el resto de los puntos positivos.

4.2 Reconocimiento de voz *offline* a *online*

La imposibilidad del uso de un reconocedor de voz *offline* fue algo que se tuvo que asumir, por las limitaciones de hardware. Se probaron diferentes sistemas, no sólo en el hardware del proyecto, sino en ordenadores convencionales con sistemas operativos más completos y hardware más potente en todos los aspectos. Porque, dentro del módulo de Python de reconocimiento de voz, hay diferentes posibilidades de uso, como se nombra en la parte de teoría, pero concretamente uno de ellos **"CMU Sphinx"**, por ser *offline* tiene la posibilidad de usar diccionarios externos de diferentes idiomas. Por defecto lleva un pequeño diccionario de unas pocas palabras en inglés que permite probarlo. Pero existe la posibilidad de implementar más.

Así que se investigó su uso, tratando de instalar todas las librerías para poder usarlo, ya que se trata de un conjunto de librerías a las que accede el módulo de reconocimiento de voz y necesita ser implementado aparte. Tras intentar varias instalaciones y pruebas, no dio buenos resultados en la Raspberry Pi y hubo problemas en su ejecución. Sin embargo, sí que fue posible ejecutarlo en un ordenador con sistema Linux, sin demasiada dificultad. Se realizó la prueba para comprobar si se estaba realizando alguna parte de la instalación o ejecución de forma incorrecta. Aun así, al tratar de añadirle el diccionario de castellano no lo reconocía o no era capaz de detectar las palabras que se decían a través del micrófono. Ya que, aunque en el ordenador sí que se ejecutaba correctamente, no siempre reconocía las palabras que se decían en inglés, mostrando por pantalla frases o palabras que nada tenían que ver con lo dicho por voz.

Existe otro sistema *offline* de la empresa **"Snowboy"** que también se trató de emplear para el proyecto, ya que parecía mucho más llamativo y la empresa daba facilidades para crear un diccionario de manera sencilla y rápida. Pero ocurrió lo mismo que con el anterior sistema *offline*. Por mucho que se intentaba no llegó a funcionar, además, en este caso no incluía un diccionario inicial para probar. De tal forma que nunca funcionó. Actualmente esta empresa está enfocada al uso de su servicio *online* mediante inteligencia artificial, como el resto de los servicios de reconocimiento *online*.

Todo ello provocó que se acabara desechando el uso de un servicio *offline*, aunque posiblemente si los diccionarios que se pudieran añadir funcionaran, la capacidad de esos diccionarios imposibilitaría su uso en un hardware del tipo de la Raspberry Pi, ya que en el momento que se cargaran en la memoria RAM del dispositivo, el rendimiento bajaría considerablemente. Porque al parecer los diccionarios almacenan las diferentes posibilidades en las que es posible reconocer una palabra.

La elección del uso del servicio que aporta Google fue rápida. Porque al tener que elegir una opción *online*, las más conocidas en la lista del reconocedor de voz, son la de Microsoft y Google. Pero como el servicio de Microsoft, según los planes de la empresa, iba a desaparecer y no es tan potente como el de Google, se optó por el más conocido y usado a nivel mundial. Además, porque su servicio de voz ya había sido probado con anterioridad y se conocía su correcto funcionamiento de primera mano.

4.3 API de YouTube para buscar en listas

Cuando se realizó la búsqueda para gestionar las listas de reproducción en YouTube, se encontraron dos formas: el uso de la API de YouTube, del mismo modo que se usa para la búsqueda de la música, o directamente buscar los enlaces URL sobre el código de la página web donde aparece la lista de reproducción. Se escogió la segunda opción, para así probar algo diferente, siendo los códigos de una longitud similar. Por afán de investigación se tomó esa decisión. El código que realiza la búsqueda sobre la página web, se adjunta a continuación.

```
# Almacena el código HTML y lo almacena en la variable
htmldoc = requests.get(str(playListUrl)).text
soup = BeautifulSoup(htmldoc, 'html.parser')

# Crea una lista de todos los títulos y los videos de la URL de
YouTube que se encuentra en la variable html-doc.
rawList = soup('a', {'class': 'pl-video-title-link'})
url = []

# Bucle para crear la lista con todos los videos de la PlayList
for link in rawList:
    url.append('%s' % ('http://youtube.com' +
'{0}'.format(link.get('href'))))
```

Pero este código, durante el mes de junio de este mismo año, dejó de funcionar. Porque, al parecer, se realizaron modificaciones en la página web y no fui capaz de encontrar los enlaces de los vídeos para reproducir la *PlayList*. Por tanto, se decidió utilizar la API para obtener los enlaces de los vídeos. El código que realiza esta función no debería de fallar nunca, ya que es un código asociado a la API y al realizarse una petición y recibir una respuesta, únicamente es necesario tratar los datos para poder ser usados a lo largo de la ejecución de la *PlayList*. Aunque el anterior código dejó de funcionar, se ha podido aprender cómo obtener datos directamente de una página web, pero su principal problema es que es susceptible a cambios, dejando de funcionar en el momento en que se realiza una modificación. Esto hace pensar que siempre que sea posible el uso de una API para un sistema que no se puede gestionar, es un procedimiento mucho más útil y sencillo, ya que almacenar todo el contenido de una página web puede ocupar demasiado dentro de la variable y el procesado de toda esa información para únicamente extraer unas pocas líneas de todo el código. No es la manera óptima para un proyecto de este calibre.

4.4 Descubrir Bluetooth de la Raspberry Pi

Para poder descubrir el Bluetooth de la Raspberry Pi se buscó documentación para su realización directamente mediante Python, pero no se encontró mucha documentación para realizar esta opción y en la mayoría de los foros y páginas web relacionadas con la Raspberry Pi, recomiendan el uso de **“subprocess.call()”** que realiza la ejecución de comandos de Linux en una

subrutina de Python, muy similar a **"os.system()"**. Sin embargo, se continuó la búsqueda de un código capaz de realizarlo utilizando Python.

Tras buscar durante un largo periodo de tiempo, no fue posible el uso de Python, ya que muchos de ellos, únicamente eran capaces de activarlo, pero no de descubrirlo y en muchas ocasiones no se menciona ni la activación ni el descubrimiento del Bluetooth. Por tanto, se realizó la prueba de los códigos mencionados anteriormente para comprobar que funcionaban correctamente y tras implementarlo en el programa y ver que todo funcionaba correctamente se tomó la decisión de utilizarlos. Aunque hubiera sido interesante haber podido utilizar Python para tal fin.

4.5 Comprobar conectividad de la Raspberry Pi

Cuando se buscó la forma de realizar la comprobación de si existía conectividad, se empezó realizando un código que comprobara si existía conectividad a una red Wi-Fi de la Raspberry Pi, sin embargo cuando estaba conectado por cable no lo detectaba, aunque realmente una vez finalizado el proyecto, el funcionamiento se realizaría mediante la tarjeta de red Wi-Fi, el problema era que en el momento se conectaba mediante cable ethernet, toda la conectividad pasaba por el cable y no se podía realizar bien las pruebas, ya que todo el proceso de programación y pruebas se realizaba mediante escritorio remoto, para mayor comodidad y se utilizaba conexión por cable ethernet, porque mediante Wi-fi, en algunas ocasiones se perdía la conectividad, debido a la distancia del punto de acceso inalámbrico y la Raspberry Pi. Esto suponía un problema para realizar pruebas mediante Wi-fi, porque se tenía que desactivar la red inalámbrica para comprobar la conectividad y al ser la red principal de toda la vivienda, solo se podían realizar pruebas cuando no había nadie haciendo uso de dicha red.

Se encontraron otras soluciones no demasiado elegantes, que comprobaban la respuesta a la ejecución de un comando **"ping",** en este caso y en función de los datos devueltos dicho comando ofrecía una respuesta. Este tipo de código fue descartado porque, aparte de no resultar elegante, tenía que tratar las diferentes respuestas que devuelve el comando **"ping"**, pudiendo inducir a error al programa si no se trataban correctamente, y ocasionaba gran cantidad de código no necesario, únicamente para tratar dicha información. De igual forma pasaba con cualquier comando de similares características.

Aunque el código cumpliría su función de forma correcta en el momento de su ejecución, se prefirió buscar una alternativa igualmente válida, pero que pudiera facilitar su uso a lo largo del desarrollo del proyecto, sin demasiados problemas tanto internos como externos. Así que se optó por el código que aparece en el programa, que revisa si existe conexión a Internet mediante una petición a una URL en este caso la de Google. Se pensó usar esa URL, ya que no es una página web que se encuentre "caída" fácilmente, debido a la empresa que está detrás del servicio que ofrece.

Sin embargo, se trató de buscar un código que pudiera funcionar de una forma mejor, que los mencionados anteriormente, pero como suele suceder, la manera más simple suele ser la mejor y en este caso no fue diferente. Y como el código no ha dado problemas a lo largo de todo el desarrollo, se decidió dejarlo como definitivo.

4.6 Elección del mejor micrófono

Las primeras pruebas de micrófono con el reconocimiento de voz se realizaron con un micrófono USB semiprofesional. Si la persona se encontraba a una distancia de pocos centímetros funcionaba perfectamente, pero al carecer de un preamplificador integrado o por software, no permitía alejarse más de esos pocos centímetros, si se quería conseguir algún resultado. Además, las dimensiones del dispositivo imposibilitaban la integración con el prototipo, ya que es una bola más grande que una pelota de tenis y resultaba muy complicado, por no decir casi imposible, integrarlo en el interior junto con el resto de los componentes. Además de que, si luego no es capaz de capturar la voz a mayor distancia, carece de sentido utilizarlo.

La elección de un micrófono supuso bastantes problemas, porque al parecer la Raspberry Pi 4 tiene mucho ruido eléctrico que se transfiere al sonido que capta el micrófono (se dispone de un modelo antiguo sin conectividad inalámbrica y no sucede), provocando que el ruido natural se incremente según se use un micrófono USB o uno conectado mediante conector jack a una tarjeta de sonido externa, ya que la Raspberry Pi carece de entrada de audio integrada.

Pero ese no fue el primer problema que se encontró a la hora de escoger un micrófono. Como se quería que todo el sistema estuviera integrado, como en la mayoría de los dispositivos que se encuentran en el mercado, donde el micrófono está oculto y solo se alcanza a distinguir unos puntos, se decidió comprar un micrófono mini USB que apenas mide unos centímetros y resultaría muy sencillo conectar mediante un cable extensor USB, para poderlo colocar en cualquier parte interna del prototipo. En vídeos y foros hay personas que dicen que para un uso similar al que se le pretendía dar, daba buenos resultados. Sin embargo, cuando el dispositivo llegó y se decidió probar primeramente en un ordenador, con un sistema operativo Windows 10, más que probado, el dispositivo no funcionaba, es decir, que no captaba ningún sonido, ni tan siquiera lo detectaba el sistema. El coste de este dispositivo es reducido, así que no suponía un gran desembolso y se decidió buscar otros modelos porque no se quería estar durante mucho tiempo probando varios dispositivos similares hasta dar con uno que funcionara.

De esta forma se buscó otro modelo de micrófono también USB para no tener que hacer uso de una tarjeta de sonido externa. Se encontró un modelo muy típico usado en llamadas a través de Internet, ya que tiene una base pesada, para darle solidez y así dejarlo sobre una mesa. Aunque este micrófono era considerablemente más grande que la versión mini, desmontando la base podía caber perfectamente dentro del prototipo. Tras una prueba exitosa en el ordenador antes mencionado y viendo que el sistema operativo daba la posibilidad de amplificarlo mediante software, se procedió a probarlo en la Raspberry Pi, esperando resultados similares en cuanto a

calidad y la posibilidad de amplificarlo por software si fuera necesario. Sin embargo, no fue así: el micrófono funcionaba aunque no era posible amplificarlo. Pero se detectó que, cuando la persona se alejaba del micrófono, el reconocedor de voz no funcionaba. Se pensó que podía ser algo similar a lo ocurrido con el primer micrófono. Así que se realizó una serie de pruebas de sonido con el programa Audacity, integrado en el sistema operativo de la Raspberry Pi, que también ha sido usado a lo largo de la carrera de Teleco.

Tras esa primera prueba se detectó que el ruido era mayor de lo normal y en el momento en que la persona se alejaba el mismo ruido tapaba la voz. Esto suponía realmente un problema, porque si se necesita hablar para realizar acciones y es el objetivo principal del proyecto, encontrarse con un nivel de ruido que imposibilitara la tarea era demasiado grave. Así que se buscó información de a qué podía deberse ese problema, por si podía ser un problema de todas las Raspberry Pi o sólo de este modelo, ya que cuando se escucha la música sí que se escucha ruido de fondo que también resulta extraño.

Como se tenía de otros proyectos una tarjeta de sonido USB y un micrófono de jack de 2,5 mm existía la posibilidad de realizar la prueba. Aunque ambos dispositivos pertenecían a otros proyectos, sobre todo el micrófono. Al conectarse a la Raspberry Pi y realizar la prueba de ruido se apreció una reducción del ruido y al tener preamplificador el micrófono, el sonido capturado era de mayor volumen y el ruido no suponía un problema si la persona se alejaba, aun estando reproduciéndose música. Pero si se alejaba demasiado, la voz era tapada por el ruido, ya que es un micrófono de solapa y no está pensado para capturar sonido a gran distancia.

No se pensó adquirir otro tipo de micrófono porque este último aun siendo de otro proyecto, era de precio elevado y conseguir un micrófono que pudiera ser mejor, supondría un desembolso demasiado grande y como se dijo anteriormente en este documento, se trataba de no superar una determinada cifra de dinero. A continuación se muestran unas imágenes de los micrófonos probados y dos capturas de los dos últimos micrófonos probados, en las que se puede ver el nivel de ruido generado.

Ilustración 37: Micrófonos probados

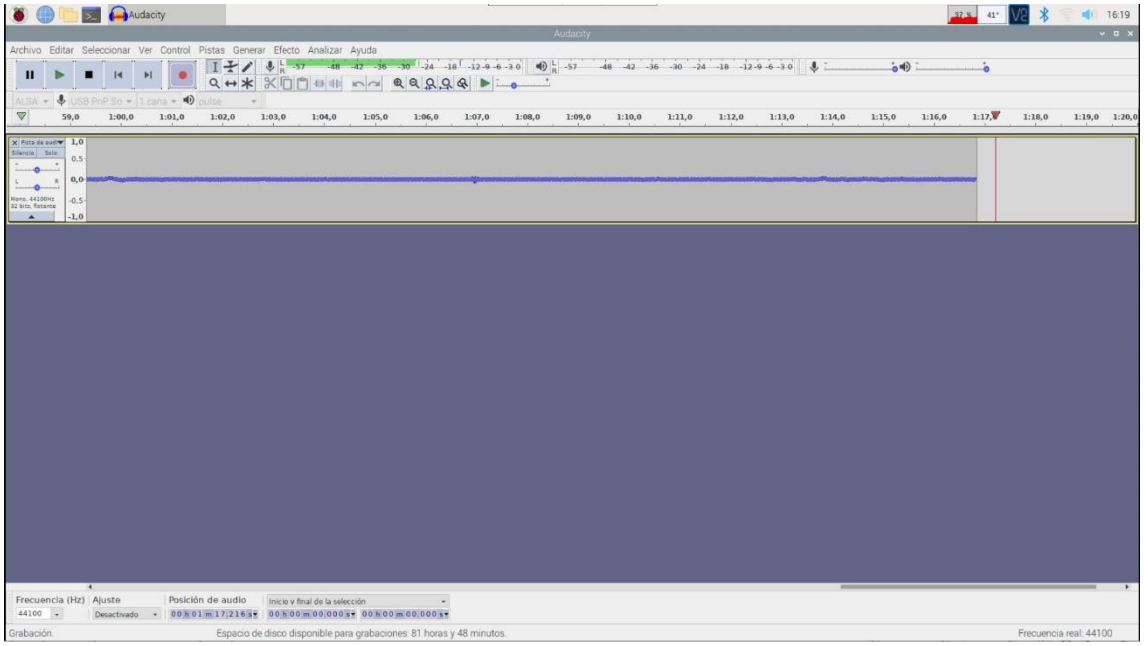

Ilustración 39: Captura ruido micrófono USB

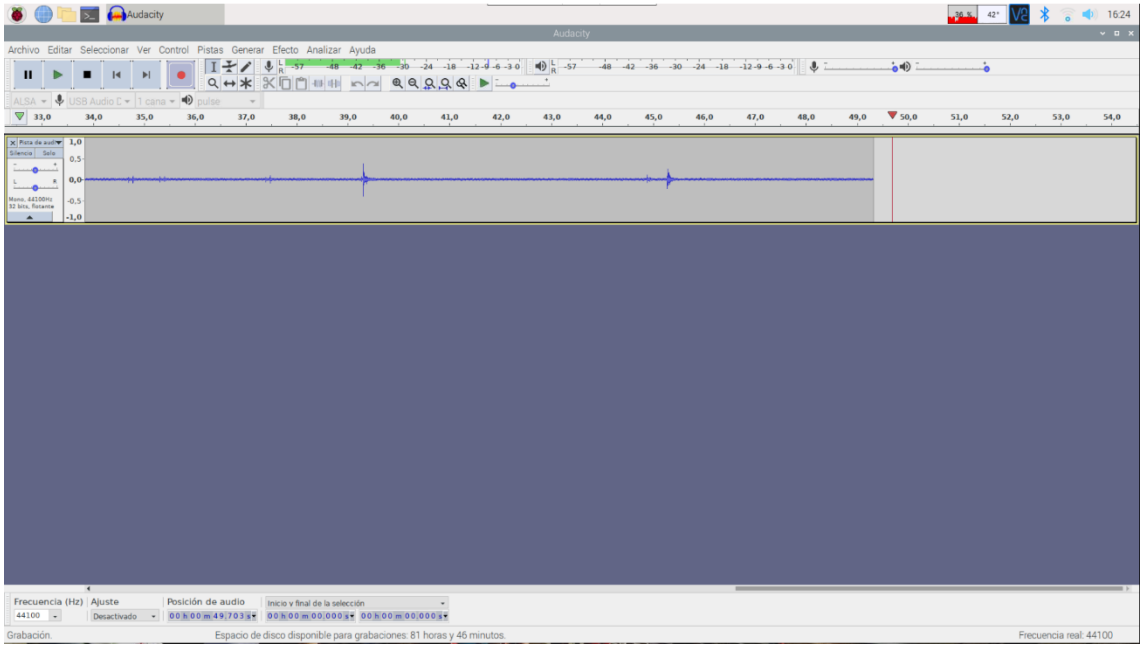

Ilustración 38: Captura micrófono de jack

Como se puede apreciar en las dos capturas, ambos micrófonos generan ruido, pero aparte de que el micrófono USB captura más ruido, no es capaz de capturar sonido. Como sí ocurre con el de jack, donde se puede ver en la barra de captura de sonido los clics del ratón a la hora de realizar las capturas situado a más de un metro de distancia, ya que ambas capturas se realizaron en tiempo real para poder ver el nivel del micrófono. En ambos casos el nivel de micrófono está al máximo en el mezclador de la Raspberry Pi.

4.7 Batería del prototipo

Sobre el uso de una batería para el prototipo, es algo que se consideró desde el inicio porque forma parte del proyecto: tener la posibilidad de situarlo en cualquier lugar de una vivienda o llevárselo a cualquier parte, siempre que se disponga de una conexión a Internet mediante Wi-fi. Por tanto, es crucial una batería con la capacidad suficiente para llevar a cabo dicha tarea. De este modo se pensó la posibilidad de "fabricarla", es decir, comprar los componentes sueltos y adaptarlos a las necesidades del sistema. Se realizó una investigación para comprobar de qué está compuesta una batería, ya que no es únicamente unas "pilas", sino que es necesario una circuitería para gestionar la carga y descarga de la batería. Esto no supondría un problema, porque ese tipo de placas se pueden conseguir de manera fácil y económica, pero el principal problema es cómo soldar las baterías entre sí para aumentar su capacidad y voltaje.

Para poder realizar esas soldaduras no es viable la utilización de un soldador convencional, ya que las soldaduras no son muy resistentes y se sueltan con facilidad. Para ello es necesario el uso de un soldador de puntos, que tiene un precio aproximado de ciento treinta y cinco euros y para realizar una pequeña cantidad de puntos de soldadura no es viable la compra de tal máquina.

Ilustración 40: Máquina soldadura

Por tanto, es necesario la utilización de una batería montada y preparada para su uso. Así que se pensó el uso utilizar una batería denominada Powerbank, ya que son económicas e incluso se disponía de una con una gran capacidad. Pero como se dijo en el punto 3.1 Materiales, sobre el sonido utilizado en el prototipo el problema de usar una batería de cinco voltios era la de no encontrar un sistema amplificador para la potencia deseada. Se disponía de uno, pero en el momento en que se le daba volumen empezaba a dar problemas, hasta que se acabó estropeando. De esta forma se buscó un amplificador de doce voltios y esto suponía también la búsqueda de una batería de igual voltaje, ya que siempre es más sencillo encontrar una solución para bajar el voltaje que para subirlo.

Al final se decidió el uso de una batería montada de doce voltios como solución para reducir el voltaje, pero manteniendo una intensidad lo suficientemente alta para hacer funcionar la Raspberry Pi, ya que, si no fuera así, ni siquiera arrancaría el dispositivo y bloquearía todo el sistema.

Capítulo 5. Conclusiones y Bibliografía

El desarrollo de este TFG ha supuesto todo un desafío, pero a la vez una gran satisfacción, porque el ver cómo una idea va cogiendo forma y desarrollándose con el paso del tiempo hasta llegar a su finalización, contemplar los resultados, que todo funciona como debe y que se resolvieron todos los problemas deja abierta la posibilidad a futuros proyectos de similares características. Ya que actualmente nos encontramos en la era de la inteligencia artificial, a pesar de que la inteligencia artificial de este sistema es poca, porque únicamente comprueba si dentro de unas frases se encuentran unas palabras en concreto para realizar una serie de acciones.

Aunque en la Escuela no se enseña a programar en Python, sí se enseña qué hay que hacer para adquirir dichos conocimientos. Y por esa razón al principio del proyecto fue necesario adquirir conocimientos en el caso de Python y algunas otras. Como puede ser el funcionamiento de los pines de la Raspberry Pi.

Además, este proyecto que no es solo una aplicación, sino que aúna dos mundos que son el virtual y el físico, ha servido también para desarrollar la creatividad y el ingenio, ya que, aunque el prototipo está inspirado en otros diseños, hubo ciertos problemas que se han ido resolviendo conforme se han encontrado. No se han nombrado en Problemas y Soluciones, por ser problemas menores que no afectan al resultado final. Como ha podido ser la situación de algunos componentes o la colocación del micrófono. Pero la conjunción de ambos mundos también ha servido para poder desarrollar el proyecto de una manera más dinámica, porque cuando no se trabajaba en la parte del código, se trabajaba en la parte del prototipo y eso ha provocado que no sea demasiada la saturación en una parte del proyecto.

Por tanto, el haber realizado este proyecto ha servido para mejorar los conocimientos que se han adquirido y además obtener nuevos. Llegando a ser mejor en este ámbito, que es lo que todo proyecto debería provocar en las personas que lo desarrollan. Porque si un proyecto no es un desafío para la persona o las personas y así generar mejores profesionales en su trabajo. Haciendo que todo mejore, ya no solo en el mundo de las telecomunicaciones sino en cualquier faceta de la vida de las personas. Porque después de haber hecho este proyecto, la sensación de haber "evolucionado" en el ámbito de las telecomunicaciones ha sido mayor, ya que como proyecto final de Grado toca todas o casi todas las ramas de las telecomunicaciones. Sistemas de comunicaciones, sonido, desarrollo de aplicaciones y electrónica. Evidentemente en diferentes proporciones. De igual manera que en el Grado se tenía que escoger una especialidad. También se decanta más por la especialidad escogida, pero sin olvidar el resto, porque tienen igual importancia en el desarrollo de un Ingeniero de Telecomunicaciones.

Resumiendo. Despertar el sistema con la palabra clave. Pedirle una canción, por ejemplo, que el sistema te reconozca y empiece a reproducirla, saca una sonrisa de satisfacción.

Debido a la pandemia y ante la imposibilidad de saber si la defensa será presencial, se han grabado dos vídeos, mostrando el funcionamiento del sistema por comandos de voz y el envió de configuración Wi-fi mediante la aplicación para móviles a través de Bluetooth.

- https://www.youtube.com/watch?v=-N6lMHb_HU0
- https://www.youtube.com/watch?v=Dck3hnIbcoM

5.1 Bibliografía

Hoy en día se realizan muchas consultas *online*, aparte de que, debido a la situación de pandemia en la que nos encontramos estos meses, la mayoría de las consultas se realizaron *online*. Además de que se puede encontrar información más actualizada.

[1] Luciano Ramalho, "Fluent Python: Clear, Concise, and Effective Programming" *1st Edición*, O'Reilly Media; Edición: 1 (25 de agosto de 2015).

[2] The MagPi," https://magpi.raspberrypi.org/ [Online].

[3] Foro Raspberry Pi, https://www.raspberrypi.org/forums/ [Online]

[4] Foro Stackoverflow, https://stackoverflow.com/questions/ [Online]

[5] API YouTube, https://developers.google.com/youtube/v3 [Online]

[6] Speech Recongnition, https://pypi.org/project/SpeechRecognition/ [Online]

[7] Pafy, https://pypi.org/project/pafy/ [Online]

[8] VLC (Python), https://pypi.org/project/python-vlc/ [Online]

[9] Pyttsx3 (TTS), https://pypi.org/project/python-vlc/ [Online]

[10] Android, Apuntes de la asignatura de telemática de Aplicaciones Telemáticas, sección Android. Transparencias de los temas.

[11] Android, https://developer.android.com/ [Online]

[12] Scripts Linux, https://diocesanos.es/blogs/equipotic/2015/07/26/bash-teoria-del-interprete-de-ordenes-scripts/ [Online]

Capítulo 6. Anexo: Cancelador de eco.

En este anexo se hablará de una parte del proyecto que se pensó realizar, ya que podía resultar interesante: se investigó, se escribió código relacionado y se realizaron pruebas. Pero se acabó por descartar por diferentes motivos que se explicarán aportando capturas y documentación al respecto. El resultado final no era satisfactorio en muchos casos por la nula reacción del sistema a la cancelación de la música y provocaba fallos en el reconocimiento de voz del sistema porque creaba interferencia de sonido y la respuesta de texto no era correcta y simplemente no reconocía la voz.

Lo primero que se realizó fue una investigación de cómo funciona un cancelador de eco. Se buscó información de canceladores de eco basados en dos archivos de sonido y se encontró un módulo de Python llamado **"Speechdsp"**. Uno de ellos con la música y otro con la música mezclada con la voz. El programa revisaba ambos archivos, anulaba la música y amplificaba el resultado final. De esta forma se conseguía que la voz se escuchara casi perfecta. El sistema no daba problemas porque se trataba de dos ficheros de sonido de igual duración y sincronizados, ya que el que contenía la música y la voz únicamente se editó para añadirle la voz.

Una vez se supo el funcionamiento de un sistema que trabaja con archivos, se buscó documentación para realizar uno que funcionara en tiempo real y que también funcionara en la Raspberry Pi. Pero lo que recomendaban en los foros era emplear el programa anteriormente mencionado para realizar dicha función. Así que antes de realizar una integración de prueba con el programa del proyecto se realizaron pruebas con códigos independientes.

Uno de los problemas que se encontró al realizar un primer código fue la de poder grabar la salida de audio de la Raspberry Pi. Ya que no se podía de una manera directa. Se buscó la manera de realizarlo mediante un puenteado por software de sonido de la salida a una entrada virtual, pero no llegó a funcionar, incluso ocasionó problemas en el sistema de sonido de la Raspberry Pi, dejando de funcionar y teniendo que restablecer todo como estaba de serie.

Así que se empleó un método un tanto complejo, que consistía en emplear varias tarjetas de sonido para capturar el sonido directamente de la salida. No es el mejor método, pero era el único que dio resultados. Afortunadamente Linux permite realizar una mezcla de sonido y es posible emitir el mismo sonido por todas las tarjetas que estén conectadas. Eso facilitó el realizar las primeras pruebas de grabación. Aunque las grabaciones no eran en tiempo real, ya que sufrían pequeños retardos o uno de ellos era de diferente duración. Pero la idea era comprobar que se pudiera realizar la cancelación de la música, por tanto no era un problema al menos en ese instante la desincronización de las grabaciones.

Una vez se tenían los ficheros de sonido con el programa Audacity se podían editar, sin aparentemente problemas. Pero aparecieron los primeros problemas. La calidad de sonido que se grababa por la salida de sonido era perfecta, con un volumen muy bueno, pero el capturado por el micrófono, no era de muy buena calidad, teniendo mucho ruido de fondo. A continuación, se adjunta una captura de la música y de la música grabada con el micrófono. El "01" es la música

grabada desde la tarjeta de sonido y el “02” es el grabado desde el micrófono, que se amplifico, ya que la grabación original era muy baja y hacía difícil su edición. Ambos están sincronizados y cortados para que tengan la misma duración, ya que para que el programa de cancelación de eco funcionara debían tener la misma duración.

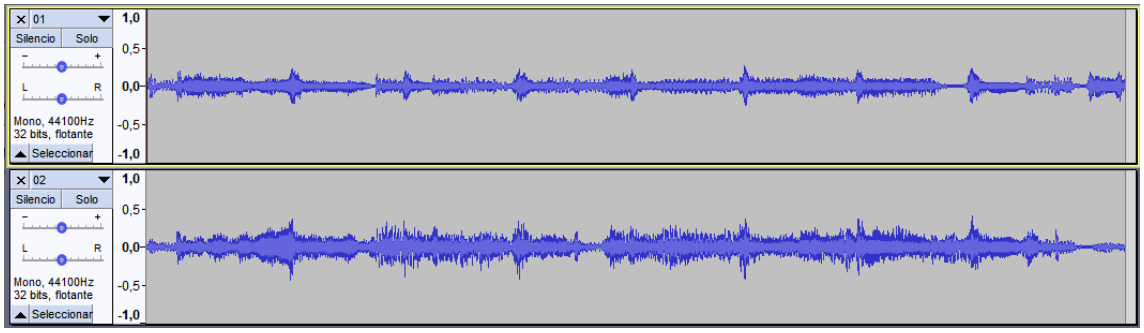

Ilustración 41: Grabaciones música y micrófono

Una vez se tuvo estos dos archivos de audio se ejecutó el programa para ver si realizaba la cancelación de eco, pero el resultado fue un fichero idéntico al grabado por el micrófono. Así que se realizaron varias pruebas tratando de buscar una sincronización más perfecta, pero el resultado era el mismo. Se buscó información al respecto, pero no se encontró nada, que pudiera ayudar.

Se buscaron diferentes opciones que pudieran dar un paso adelante, editando los archivos para pasarlos por diferentes filtros de sonido que eliminaran el ruido, pero no lo eliminaban del todo. Así que se aplicaron filtros de sonido tanto de paso alto como de paso bajo en ambos ficheros, para acotar el rango de frecuencias del sonido, pero tampoco dio resultado. En algunas ocasiones parecía que aplicaba bien el filtro de cancelación y se apreciaba una atenuación leve de la música, pero en el momento en que se aplicaba un sonido externo en la grabación, no funcionaba.

Pero como se mencionó anteriormente, uno de los problemas que también aparecía era la grabación en tiempo real de ambos ficheros de sonido. Porque, aunque se trató de sincronizarlos al comienzo de la grabación, siempre se producía un desfase de unos milisegundos que hacía que no estuvieran sincronizados, aparte de que también la duración cambiaba en cada grabación. La sincronización se podía realizar al milisegundo, gracias al mixer pulse de Linux que permite crear un retardo en la salida de la señal. Esto permitía adecuar el tiempo de las grabaciones, pero como se menciona, no era posible porque siempre se desincronizaba, y a pesar de que en muchos foros se decía que siempre que el desfase no fuera superior a veinticinco milisegundos, se podía aplicar o filtro de cancelación de eco, pero el desfase en muchas ocasiones era superior, ya que un desfase tan pequeño no sería perceptible por el oído humano, sin embargo en las grabaciones el desfase era bastante claro. Como el desfase no siempre es el mismo, no es posible aplicar un corrector de desfase en el mezclador de Linux , ni crear un programa que gestione ese desfase. Por tanto, el cancelador de eco no funciona. Dando como resultado un fichero igual al grabado por el micrófono.

Se buscó información para corregir el problema del desfase en las grabaciones. Se encontró muchas entradas en foros con preguntas similares para realizar una cancelación en el retorno de la voz de dos interlocutores, pero no aparecía una respuesta con una solución clara al respecto. Hasta que se encontró una respuesta en el foro de Raspberry Pi oficial del año 2017. En ella se dice que la razón principal por la que un sistema en tiempo real nunca funcionará en una Raspberry Pi es porque el sistema operativo es multitarea, detallando que la tarea a realizar, en este caso grabación de sonido, puede detenerse repentinamente durante varios milisegundos para dar tiempo a otras tareas que siempre están corriendo. Siendo necesario emplear un sistema operativo basado en tiempo real para poder realizar dicha tarea. Como por ejemplo FreeRTOS. Pero estos sistemas no son compatibles con Python, ya que están pensados para realizar una única tarea.

Se investigó más al respecto de estos sistemas especializados en procesado de señal digital. Se encontró una empresa que se dedica a realizar placas para las tareas de cancelación de ruido y de eco. Pero esto provocaba otro problema si se emplea una placa externa. Es el tema del espacio, ya que hay un espacio limitado dentro del prototipo. Aparte del coste de estos dispositivos, porque la empresa se encuentra en Canadá y solo con el envió se sale del presupuesto establecido. Este tipo de placas se emplea en los sistemas para realizar conferencias en las empresas. Pero también hay otro problema y es el tiempo de respuesta, es decir, cuantos más sistemas se sitúen entre la persona que quiere utilizar el sistema y la activación del sistema, mayor es el retardo en la activación y no es deseable que tarde demasiado en responder. Al igual que ocurre en algunas ocasiones debido a la velocidad de respuesta de Internet que puede retrasar la activación.

Por tanto, a la conclusión que se llegó fue que implementar un sistema en tiempo real para realizar la función de cancelación de eco integrado en la Raspberry Pi, es complejo. Sí que es posible aplicar un cancelador de eco en dos archivos previamente preparados y sin ruido de fondo que provoque que no realice bien la tarea. Pero realizar una cancelación de eco en tiempo real, es excesivamente complicado para las capacidades del sistema en su conjunto.

A continuación se adjunta el código empleado para la grabación de la música y del micrófono. Lo que se empleó fue la ejecución de dos hilos, que se ejecutasen en paralelo para así grabar al mismo tiempo, aunque cada uno graba un dispositivo diferente.

```
def main():
#Realiza la llamada mediante hilos a las funciones de grabación de los
dispositivos
     t = threading.Thread(target=microfono, name="reproductor")
     t.daemon
     t.start()

     t = threading.Thread(target=altavoces, name="parecancel")
     t.daemon
     t.start()

main()
```

```
import sys
import time
import getopt
import alsaaudio

import wave


def microfono / altavoces(): #el código es el mismo, solo cambia el
contenido de la variable device

    #la variable device, define el dispositivo que se grabara

    device = 'plughw:2'


    inp = alsaaudio.PCM(alsaaudio.PCM_CAPTURE, alsaaudio.PCM_NONBLOCK,
device)

    inp.setchannels(1)
    inp.setrate(44100)
    inp.setformat(alsaaudio.PCM_FORMAT_S16_LE)

    inp.setperiodsize(160)

    f = wave.open('test_pafy.wav', 'w')
    f.setnchannels(1)
    f.setsampwidth(2)
    f.setframerate(44100)


    loops = 5000
    while loops > 0:
        loops -= 1

        l, data = inp.read()

        if l:

            f.writeframes(data)
            time.sleep(.001)
```