

UNIVERSIDAD DE MÁLAGA
ESCUELA TÉCNICA SUPERIOR DE
INGENIERÍA DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

SISTEMA DE APERTURA DE PUERTA
MEDIANTE RECONOCIMIENTO DE VOZ
USANDO RASPBERRY PI

GRADO EN INGENIERÍA DE
SISTEMAS DE TELECOMUNICACIÓN

JESÚS HERMOSO DE MENDOZA HERNÁNDEZ
MÁLAGA, 2018

E.T.S. DE INGENIERÍA DE TELECOMUNICACIÓN, UNIVERSIDAD DE MÁLAGA

Sistema de apertura de puerta mediante reconocimiento de voz usando Raspberry Pi

Autor: Jesús Hermoso de Mendoza Hernández

Tutor: Lorenzo José Tardón García

Departamento: Ingeniería de comunicaciones

Titulación: Grado en Ingeniería de Sistemas de Telecomunicación

Palabras clave: reconocimiento de voz, Raspberry Pi, Google Cloud, Python, cerradura eléctrica

Resumen

Este proyecto implementa un sistema de apertura/bloqueo de una cerradura eléctrica mediante identificación por contraseña.

La contraseña será introducida por voz, y se transcribe a texto usando el servicio en la nube Google Cloud Speech-to-Text. La plataforma hardware es una Raspberry Pi y se usa el lenguaje de programación Python.

Se desarrolla la funcionalidad de identificación así como de modificación de contraseña del sistema. Respecto a la interacción el usuario, se usan 2 LEDs y un zumbador como aviso al usuario usando para ello los pines GPIO, que dotan a la Raspberry Pi de la habilidad de interaccionar con la realidad, mediante sensores y dispositivos electrónicos.

**Door opening system triggered by speech recognition
using Raspberry Pi**

Author: Jesús Hermoso de Mendoza Hernández

Supervisor: Lorenzo José Tardón García

Department: Communications Engineering

Degree: Telecommunications Systems Engineering

Keywords: Speech recognition, Raspberry Pi, Google Cloud, Python, electric door lock

Abstract

This project implements a system for opening/blocking an electric lock by means of password identification. The password will be entered by voice, and transcribed into text using the Google Cloud Speech-to-Text cloud service. The hardware platform used is a Raspberry Pi 3B and the programming language chosen to perform the project is Python.

Identification functionality is developed as well as system password modification. Regarding the user interaction, 2 LEDs and a buzzer are used as indication to the user. For this, GPIO pins are used, which gives Raspberry Pi the ability to interact with reality, using sensors and electronic devices.

Agradecimientos

A lo largo de la realización de este trabajo he aprendido mucho, tanto a nivel técnico como otros aspectos y problemas más prácticos que surgen a la hora de desarrollar un proyecto relativamente grande.

En primer lugar me gustaría agradecer desde aquí a mi padre y a mi madre, pues gracias a ellos estoy vivo y he tenido la oportunidad de cursar estos estudios. Un gran abrazo a mis hermanos David y Ester.

También quiero dar mi sincero agradecimiento a mi tutor Lorenzo Tardón, que me guió todas las veces que estaba muy perdido y con dudas, que no han sido pocas.

Terminar este proyecto, que supone el fin de mi etapa universitaria, me causa una gran ilusión, y espero poder aprovechar tanto todo lo aprendido, como los errores cometidos, durante los años de mi carrera profesional.

Contenido

Capítulo 1. Introducción.....	1
1.1. Introducción.....	1
1.2. Objetivos.....	2
1.3. Estructura del Trabajo.....	3
Capítulo 2. Tecnologías y software utilizados.....	5
2.1. Raspberry Pi	5
2.1.1. Raspbian.....	7
2.1.2. GPIO.....	8
2.2. Python	9
2.2.1. Biblioteca pyaudio.....	10
2.2.2. Biblioteca google-cloud-speech.....	10
2.2.3. Biblioteca RPi.GPIO.....	10
2.3. Reconocimiento de voz. Google Cloud.....	10
2.4. SSH.....	12
2.5. VNC.....	13
2.6. Angry IP Scanner.....	13
2.7. Editor Vim.....	13
2.8. Fritzing.....	13
Capítulo 3. Diseño del sistema.....	15
3.1. Requisitos del sistema.....	15
3.2. Diagrama de bloques.....	16
Capítulo 4. Implementación del sistema.....	19
4.1. Conexión a Raspberry Pi mediante SSH.....	19
4.2. Activación y conexión al escritorio remoto mediante VNC.....	21

4.2.1. Instalación y configuración del servidor vncserver.....	21
4.2.2. Instalación del cliente xtightvncviewer.....	22
4.2.3. Conexión a la Raspberry Pi.....	22
4.3. Descripción detallada de los bloques del sistema.....	23
4.3.1. Captura de audio.....	23
4.3.2. Uso de Google Cloud.....	26
4.3.3. Verificación de contraseña.....	29
4.3.4. Apertura de cerradura eléctrica.....	30
4.4. Interfaz con el usuario.....	32
4.4.1. LEDs de indicación.....	32
4.4.2. Zumbador piezoeléctrico.....	33
Capítulo 5. Evaluación del sistema.....	37
5.1. Funcionamiento	37
5.2. Consumo.....	40
Capítulo 6. Conclusiones y trabajo futuro.....	41
Apéndice A. Código del programa.....	43
Referencias.....	52

Lista de Acrónimos

IoT.	Internet of Things
SSH	Secure Shell
GPIO	General Purpose Input Output
UART	Universal Asynchronous Receiver-Transmitter
SPI	Serial Peripheral Interface
WER	Word Error Rate
FTP	File Transfer Protocol
PCB	Printed Circuit Board
IP	Internet Protocol
VNC	Virtual Network Computing
RMS	Root Mean Square
JSON	JavaScript Object Notation

Lista de Imágenes

Imagen 1. Raspberry Pi y sus principales componentes hardware

Imagen 2. Diagrama de pines GPIO de la Raspberry Pi

Imagen 3. Diagrama de bloques del sistema

Imagen 4. Lista de IPs activas en la red encontradas por Angry IP Scanner

Imagen 5. Conexión mediante SSH;Error! No se encuentra el origen de la referencia.

Imagen 6. Inicio automático del servidor VNC en la Raspberry Pi

Imagen 7. Dirección IP y escritorio remoto al que se conecta xtightvncviewer

Imagen 8. Tarjeta de sonido y micrófono utilizados

Imagen 9. Lista de dispositivos USB detectados

Imagen 10. Lista de dispositivos de grabación detectados

Imagen 11. Creación de nuevo proyecto en Google Cloud

Imagen 12. Creación de credenciales para el proyecto en Google Cloud

Imagen 13. Generación de la clave privada en formato JSON

Imagen 14. Archivo ~/.bashrc tras añadir la variable de entorno

Imagen 15. Circuito de activación de cerradura eléctrica mediante relé

Imagen 16. Sistema completo montado en protoboard

Imagen 17. Leds verde y rojo conectados a los pines GPIO 20 y 21 respectivamente

Imagen 18. Zumbador conectado al pin GPIO 18

Imagen 19. Identificación correcta en el sistema

Imagen 20. Identificación incorrecta en el sistema

Capítulo 1. Introducción

En esta introducción se expondrá el contexto tecnológico del proyecto, y una descripción de los objetivos perseguidos con el mismo. Por último se comentará la estructura de la presente memoria.

1.1. Introducción

Actualmente vivimos en una época en la que la tecnología se ha hecho más accesible y barata a todos los públicos, y se prevee que esta tendencia siga en aumento en los próximos años.

Cada vez es más barato, y hay más opciones para desarrollar todo tipo de soluciones tecnológicas para multitud de tareas. Las principales tendencias tecnológicas de las que se alimenta este trabajo son tres:

- **Nuevas interfaces de usuario.** En los últimos años han surgido una gran multitud de métodos de interacción con las aplicaciones software. Algunos ejemplos son interfaces táctiles, de realidad virtual, o mediante el reconocimiento de la voz, la elegida para este trabajo.
- **Uso de hardware libre.** Desde la irrupción de Arduino en, que trajo al mundo el concepto de hardware libre, se ha producido una revolución en el campo de la electrónica, que permite implementar proyectos de forma relativamente sencilla y barata para cualquier persona. Pocos años más tarde, llegaría la Raspberry Pi, el equivalente a un ordenador en el tamaño de una tarjeta de crédito, con toda la potencia de Linux y la capacidad de

Arduino de interaccionar con el mundo físico mediante sensores y actuadores.

- **Internet de las cosas (IoT).** Actualmente cada vez más dispositivos tienden a estar interconectados, pudiendo proporcionar y recibir información de la red, de un proveedor o de otro dispositivo. En este trabajo, la conectividad a la red se usa para el proceso de reconocimiento de voz, y para la posibilidad de un posible administrador de acceder al log de entradas de forma remota (mediante SSH).

1.2. Objetivos

El proyecto tiene el objetivo general de elaborar una aplicación que haga uso de la tecnología de Reconocimiento de voz, así como de la placa de hardware de bajo coste Raspberry Pi. La apertura de una puerta por identificación por voz mediante contraseña cumple estos requisitos.

Este objetivo se alcanzará por medio de:

- Un hardware de bajo coste, concretamente una Raspberry Pi.
- La utilización del lenguaje de programación Python, y bibliotecas específicas para captura de audio, reconocimiento de voz y control de los pines GPIO.
- La utilización de la plataforma de computación en la nube Google Cloud, a través del sistema que funcionará en Raspberry Pi.

1.3. Estructura del trabajo

A continuación se describe brevemente el contenido de cada capítulo:

- En este primer capítulo se realiza una introducción al contexto del proyecto, y los objetivos perseguidos con el mismo.
- En el segundo capítulo se enumeran el hardware, las tecnologías y el software utilizado, cual es su objetivo, y fuentes de información de los mismos. No hay detalle del uso de estas tecnologías, pues esto se reserva para el capítulo 4.

- En el tercer capítulo se especifica los requisitos que debe cumplir el sistema y el diseño del mismo.
- En el cuarto capítulo se profundiza en el detalle de todos los bloques del sistema, su configuración, funcionamiento y código desarrollado.
- En el quinto capítulo se realiza una breve evaluación del sistema, en relación a tres aspectos principales (funcionamiento, coste y consumo).
- En el sexto y último capítulo se realiza una valoración del trabajo realizado, y posibles mejoras que se podrían llevar a cabo.

Capítulo 2. Tecnologías utilizadas

En este capítulo se van a describir brevemente las tecnologías y herramientas fundamentales empleadas en el desarrollo de este proyecto, tanto las herramientas software como la plataforma hardware.

2.1. Raspberry Pi

La Raspberry Pi es un computador de placa reducida, que contiene todo lo necesario para poder funcionar de forma autónoma, una CPU, una GPU, memoria RAM y almacenamiento mediante una tarjeta microSD.

Sus principales ventajas son:

- Pequeño tamaño (igual a una tarjeta de crédito)
- Consumo contenido
- Precio

El concepto de la Raspberry surgió en 2006. Eben Upton, profesor de la Universidad de Cambridge junto con otros colegas comenzó a preocuparse del bajo que nivel con el que estaban entrando los estudiantes, comparado con años anteriores.

Con este objetivo, y teniendo en cuenta las pocas herramientas disponibles para los jóvenes para experimentar con la programación y los proyectos de

robótica, electrónica y otras disciplinas, se creó el primer modelo de Raspberry Pi en febrero de 2012.¹

En la Imagen 1 observamos los principales componentes y características que incorpora esta placa.

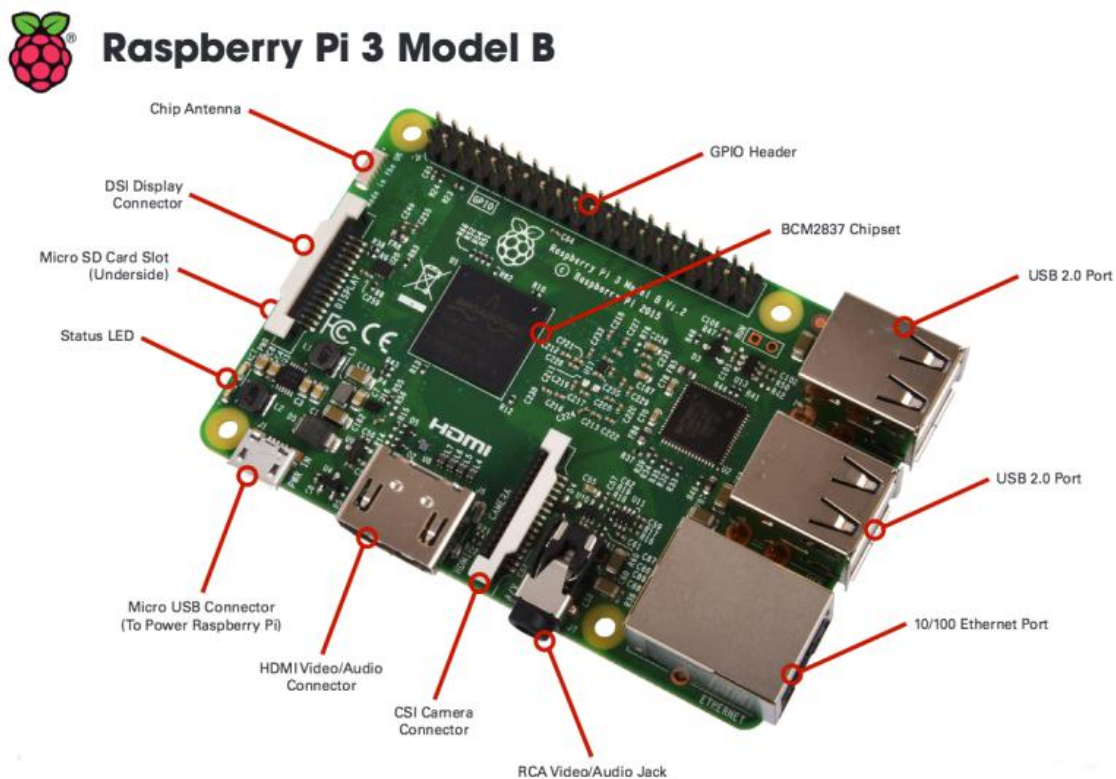


Imagen 1. Raspberry Pi y sus principales componentes hardware

El Chipset BCM2837 es el componente más importante, contiene un procesador ARM Cortex A53 a 1.2 GHz [1] (convierte a la RPi 3 en un 50% más rápida que el modelo RPi 2). También cuenta con un procesador gráfico VideoCore IV a 400 MHz.

4 puertos USB. En principio se puede conectar todo lo que se podría conectar en un Sistema Operativo Linux estándar, cuentan con gran compatibilidad con todo de periféricos. La limitación principal que hay que mencionar es que están diseñados para cargas que extraigan en torno a 100 mA. Aparatos que

¹BBC News, "The Raspberry Pi Computer goes on general sale" , febrero 2012, <https://www.bbc.co.uk/news/technology-17190918>

requieran hasta 500 mA podrían conectarse y funcionar, pero desde la Raspberry Pi Foundation no garantizan la fiabilidad con esos niveles de corriente [2]. En este proyecto sólo se usa un puerto USB, para conectar la tarjeta de sonido necesaria para usar un micrófono con Raspberry Pi.

Conector Ethernet. Da la posibilidad de conectarse a redes cableadas. Aunque cuenta con la versión de 100Mbps.

Wi-Fi. Posibilidad de conectarse a redes inalámbricas. En este proyecto esto será de vital importancia pues necesitamos conexión a la red para usar el la API que realiza el reconocimiento de voz.

Conector microUSB. A él se conecta la fuente de alimentación.

Ranura para tarjeta microSD. La Raspberry no tiene ningún medio de almacenamiento incorporado, en su lugar se le debe introducir una microSD con el Sistema Operativo y todo lo que precisemos instalado.

Salida de audio de 3.5 mm (Jack) y Vídeo compuesto. Permite reproducir audio y video. Sin embargo, como hemos comentado no incluye entrada de audio.

Conector HDMI. Permite conectar la Raspberry Pi a un monitor y manejarla como si fuera un PC completo.

Pines GPIO. Son un conjunto de pines de propósito general que pueden servir tanto de entrada o de salida, completamente programables y reconfigurables por el usuario. Para más información, se profundiza en el apartado 2.1.3.

2.1.1. Raspbian

Raspbian es el Sistema Operativo oficial proporcionado por Raspberry Pi Foundation. Incluye varias aplicaciones educativas y para el aprendizaje de la programación. No entraremos en más detalles a este respecto por considerarlo básico. Para la descarga de este SO visitar [3] y una guía de instalación se puede encontrar en [4].

2.1.2. GPIO

Los pines GPIO (General Purpose Input Output) son un conjunto de conexiones que permiten interactuar con el mundo físico, tanto leyendo datos de sensores o componentes electrónicos, como actuando sobre cualquier dispositivo o activándolo [5]. Estos pines son reprogramables, de forma análoga a los pines de entrada/salida de la placa Arduino; la principal diferencia es que con la Raspberry podemos usar la flexibilidad que proporciona un SO Linux, así como un lenguaje de alto nivel como es Python, que proporciona muchas posibilidades al programador. En este proyecto sólo se usan pines como salida (encender leds, zumbador, y activar un relé).

Hay 2 aspectos que debemos destacar.

El primero es mencionar las precauciones que debemos tener para no dañar el hardware [6]. No debemos conectar ninguna tensión de más de 3.3 V a ningún pin GPIO, en cuyo caso es muy probable que el bloque entero de pines quede destruido. Asimismo, tampoco se deben extraer más de 16 mA por cada pin y se debe mantener el total de salidas por debajo de 50 mA (para los modelos de 40 pines, como es el 3B). No debemos conectar una alimentación de más de 5 V. Y por último, no debemos extraer más de 250 mA en total de los pines de 5 V de salida.

El segundo es conocer bien el esquema o localización de los pines de nuestra placa. El primer modelo tenía 26 pines, pero a partir de la versión A+ y B+ se amplió a 40 pines, aunque los primeros 26 son compatibles con versiones anteriores. En la Imagen 2 observamos un esquema de los diferentes tipos de pines GPIO que incluye la Raspberry Pi 3B.

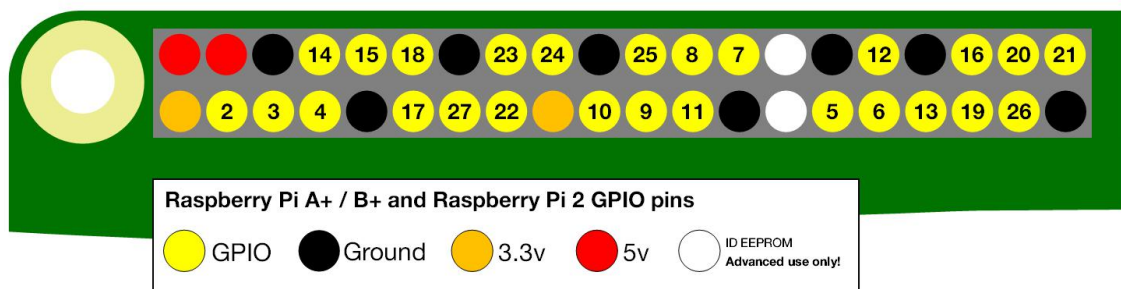


Imagen 2. Diagrama de pines GPIO de la Raspberry Pi

- Pines GPIO generales. Se pueden configurar como entrada o como salida, y se pueden establecer a un valor alto (3.3 V) o un valor bajo (0 V). Como se comentaba antes, el usuario debe proteger y tomar las medidas necesarias para no solicitar más corriente a un pin de este tipo, pues la Raspberry Pi no implementa ninguna precaución a nivel eléctrico en este sentido.
- Pines de GND (Toma de tierra).
- Pines que proporcionan una salida fija a 5V y a 3.3V. Recordamos que se debe tener en cuenta lo comentado para los pines generales de entrada/salida, si bien estos soportan mayores demandas de corriente.
- Varios de los pines realizan también otras funciones, como comunicaciones mediante UART, SPI, etc, aunque no se entrará en detalles porque no es relevante para este proyecto.

Una referencia interesante con diagramas interactivos de los pines GPIO y explicaciones detalladas es [7].

2.2. Python

Python es un lenguaje creado por el holandés Guido Von Rossum a finales de los ochenta, con el objetivo de ser un lenguaje de muy alto nivel. Es multiparadigma (programación orientada a objetos, programación imperativa y programación funcional).

En los sistemas Linux viene incluido por defecto. En particular en la Raspberry es uno de los lenguajes más utilizados, pues va en la línea con su filosofía de aprendizaje. Es importante mencionar que hay 2 versiones ampliamente extendidas, la 2 y la 3, y ambas son en gran parte incompatibles entre sí. En este proyecto se ha usado la versión Python 2.7, presente por defecto en la mayoría de sistemas Linux hoy en día. La documentación se encuentra en [8].

2.2.1. Biblioteca pyaudio

Es la encargada de gestionar la captura de audio con el micrófono. Es una adaptación a Python de la librería multiplataforma PortAudio.

La documentación se puede encontrar en [9].

2.2.2. Biblioteca google-cloud-speech

Esta biblioteca es la encargada de realizar el proceso de transcripción de la voz a texto. Es necesario conexión a Internet cada vez que se ejecute código de esta biblioteca. La página de documentación se puede encontrar en [10].

2.2.3. Biblioteca RPi.GPIO

Es la encargada de gestionar las conexiones de los pines GPIO. Su función principal es configurar los pines como entrada o salida, y poner un nivel alto o nivel bajo en ellos. La documentación se puede encontrar en [11].

2.3. Reconocimiento de voz. Google Cloud

Aunque para los humanos el hecho de oír, reconocer y dar una interpretación a las señales de audio que percibimos es algo innato y muy natural, esto no es así para un ordenador. En la última década las investigaciones han avanzado mucho en este campo. Actualmente se usan complejas técnicas de inteligencia artificial, basadas principalmente en redes neuronales, y para poder generar los modelos necesarios se recolectan grandes conjuntos de datos. Sólo el pasado año 2017 se consiguió hasta llegar a la tasa de error (en inglés WER) de los humanos [12], y empresas como IBM y Google están constantemente aportando mejoras en esta tecnología [13] [14], por lo que se prevee que siga mejorando.

Por tanto, debido a la complejidad de esta tarea, y que por la naturaleza de este sistema, no está limitado a un conjunto finito de palabras (el usuario podría elegir cualquier palabra o conjunto de palabras) se ha optado por una tecnología que implemente el reconocimiento automático de voz.

Google Cloud Platform es una plataforma que reúne todos los servicios de desarrollo y computación en la nube de Google. Estos servicios usan las mismas tecnologías e infraestructuras que Google usa para sus productos finales, como Gmail o YouTube.

Se ofrecen servicios modulares, como computación, almacenamiento en base de datos, análisis de datos y diversas aplicaciones de inteligencia artificial (reconocimiento de voz, reconocimiento de imagen, traducción multilenguaje). Así, en concreto usaremos Google Cloud Speech Speech-to-Text API en este proyecto. La página de documentación se puede encontrar en [15]

La idea es que los desarrolladores puedan delegar la parte más técnica y compleja de una aplicación, y centrarse en la lógica de negocio. Esto proporciona innumerables ventajas, en la mayoría de los casos:

- No hay necesidad de gestionar servidores
- Ahorro en salarios
- A medida que el proveedor (en este caso Google) mejora su infraestructura y sus servicios, esa mejora llega al usuario de servicios en la nube sin tener que haber invertido en Investigación.

Citando de la página de información oficial [16]:

Google Cloud Speech-to-Text enables developers to convert audio to text by applying powerful neural network models in an easy-to-use API. The API recognizes 120 languages and variants to support your global user base. You can enable voice command-and-control, transcribe audio from call centers, and more. It can process real-time streaming or prerecorded audio, using Google's machine learning technology.

Como mencionábamos, la potencia y flexibilidad (mayor robustez al ruido, varios lenguajes) en cuanto a la tarea de Reconocimiento de voz es mucho mayor de lo que se podría alcanzar desarrollando algo de forma individual. Así como el tiempo de desarrollo también se reduce considerablemente.

Finalmente, en cuanto al coste, hay que mencionar que los primeros 60 minutos de transcripción de audio son gratis, pero a partir de ese punto hay que pagar. La unidad mínimo de audio que computa son 15 segundos, por lo que aunque enviemos audios más cortos, no se reduce el precio. Para el objetivo de este proyecto, con esto tenemos más que suficiente.

2.4. SSH

SSH (Secure Shell) es un protocolo que permite la comunicación entre 2 computadores, con el modelo de cliente-servidor. Usa por defecto el puerto 22. Se destaca respecto a FTP o Telnet en que encripta la sesión de conexión. En este trabajo se usa el comando de terminal *ssh*, otra opción muy utilizada es el software multiplataforma Putty.

Gracias a SSH podemos usar la Raspberry Pi 'headless', esto es, sin conectarla a un monitor, teclado ni ratón. Además, proporciona la posibilidad de manejar en remoto desde cualquier parte el sistema.

2.5. VNC

VNC (Virtual Network Computing) es un sistema que permite compartir visualizar y controlar completamente el escritorio de un equipo remoto. Es multiplataforma. En este trabajo se usa el servidor *vncserver* [17] y el cliente *xtightvncviewer* [18].

2.6. Angry IP Scanner

Este software lo usamos puntualmente en este proyecto para localizar qué equipos están conectados a una misma red, con ello conseguimos averiguar la IP de la Raspberry Pi y conectarnos por SSH. Se puede descargar de [19]

2.7. Editor Vim

Vim (Vi IMproved) es un editor de texto en consola, basado en el editor Vi, que mejora sus capacidades añadiendo funciones como coloreado de sintaxis, selección visual, uso del ratón o completado de archivos. La documentación se encuentra en [20].

2.8. Fritzing

Fritzing es un programa open source para diseñar y realizar esquemas de circuitos electrónicos. La web de este software es [21].

Capítulo 3. Diseño del sistema

En este capítulo se van a presentar los requisitos específicos del sistema, así como una visión global de la metodología empleada para cumplir con dichos requisitos y un diagrama de bloques de los que constará el sistema.

3.1. Requisitos del sistema

Identificación. El sistema sólo usará reconocimiento de contraseña y no distinguirá entre usuarios. Esta decisión viene motivada principalmente porque la tecnología usada (GCS) no incluye esta función [22]. Se intuye que internamente está implementado, pues como podemos leer en [23], se puede fragmentar el audio según quién habla en cada momento. No obstante, esto no nos sirve para identificar al hablante teniendo solo una grabación. Dado que no tendría sentido que para distintos usuarios (que el sistema no puede distinguir entre sí) se usaran diferentes contraseñas, es natural establecer que sólo existirá una contraseña.

Grabación. El sistema estará monitorizando continuamente el audio ambiente. Es decir, estará tomando muestras de audio con el micrófono, pero sin salvar a archivo, hasta que se super un umbral, momento en el cual empezará a grabar 3 segundos de audio, tiempo razonable para pronunciar una palabra o frase corta.

Variedad de reconocimiento de voz. Se usará reconocimiento de voz síncrono, esto es, se realiza la grabación y sólo entonces se envía a Google Cloud para realizar la transcripción a texto [24][25]. De forma asíncrona (siempre transcribiendo voz, esperando identificar la contraseña) resultaría en

un mucho mayor uso de envío/recepción de datos, pero sobre todo incrementaría mucho el coste en reconocimiento de voz, pues sólo hasta 1 hora de audio es gratuito en Google Cloud Speech-to-Text.

Interfaz de usuario. Se avisará al usuario si la verificación ha sido correcta o incorrecta mediante una señal visual (LED verde o LED rojo, respectivamente) y auditiva (zumbador que emite un tono largo o 3 tonos cortos intermitentes).

3.2. Diagrama de bloques

Se ha realizado un diseño del sistema de forma modular. Esto ha permitido encontrar errores y depurar de forma más sencilla, así como una organización más clara del código implementado.

En la Imagen 3 se presenta un diagrama desde un punto de vista de general, los bloques de los que consta el sistema, y se pasa a comentar cada uno.

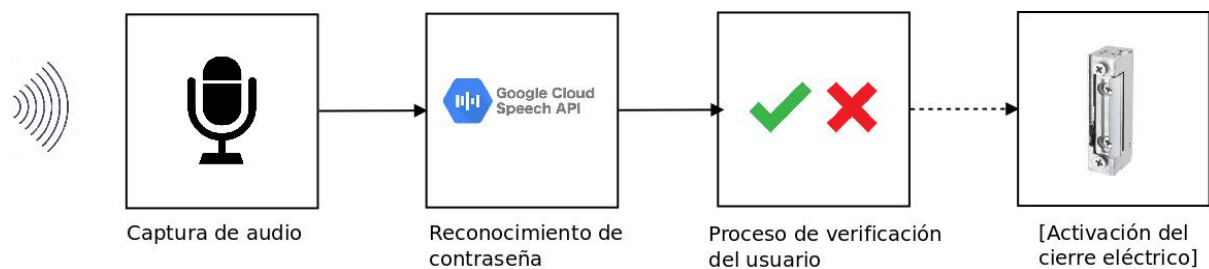


Imagen 3. Diagrama de bloques del sistema

Captura de audio. El usuario que se quiere identificar debe decir alguna palabra o hacer algún sonido para ‘activar’ el sistema, entonces se grabarán 3 segundos de audio y se guarda a archivo. El sistema tendrá un umbral, por debajo del cual no empieza a grabar, para evitar el ruido ambiente.

Reconocimiento de contraseña. El archivo de audio que se ha grabado, se enviará mediante la API de Google Cloud Speech-to-Text y ésta devuelve un String con la transcripción más probable.

Proceso de verificación del usuario. Una vez tenemos la contraseña que el usuario ha pronunciado en formato texto, la comparamos con la que

previamente tenemos almacenada (en un archivo de texto), y sólo si es correcta, se activará la cerradura eléctrica, que permitirá entrar al usuario.

Activación de la cerradura eléctrica. Como es de esperar, si las contraseñas coinciden, se manda una señal de nivel alto al pin que controla el circuito que finalmente conecta la cerradura eléctrica momentáneamente a su alimentación, abriéndose ésta.

Cada módulo Python implementa una parte de la funcionalidad (grabación, reconocimiento de voz, activar un relé, encender leds, activar zumbador) y además de los bloques mostrados en el diagrama, las 2 funciones del sistema (a nivel de usuario) tienen archivos específicos en código Python:

- ***Identificacion.py***. Identifica si un usuario ha pronunciado la contraseña correcta o no.
- ***Modifica_contraseña.py***. Establece una nueva contraseña, tras comprobar que la pronunciación coincida con la introducida por teclado, para asegurar que el sistema funcionará correctamente.

Capítulo 4. Implementación del sistema

En este capítulo se exponen todos los detalles de la implementación del sistema de reconocimiento, trabajo con la Raspberry Pi, la conexión eléctrica con el sistema de cierre y la interfaz de usuario desarrollada.

4.1. Conexión a Raspberry Pi mediante SSH

En primer lugar, nos conectaremos mediante SSH a la Raspberry Pi, para poder trabajar de forma cómoda desde otro PC. Es requisito que el cliente desde el que usamos SSH y la Raspberry estén conectados a la misma red.

Para realizar la conexión a la Raspberry lo único que necesitamos conocer es su IP en la red. Como en este caso no se dispone de un monitor, se puede conectarla al router directamente mediante un cable Ethernet.

Usamos el software Angry IP Scanner para encontrar la IP de la Raspberry. En la Imagen 4 se observa como la IP asignada en nuestro caso es 192.168.0.176.

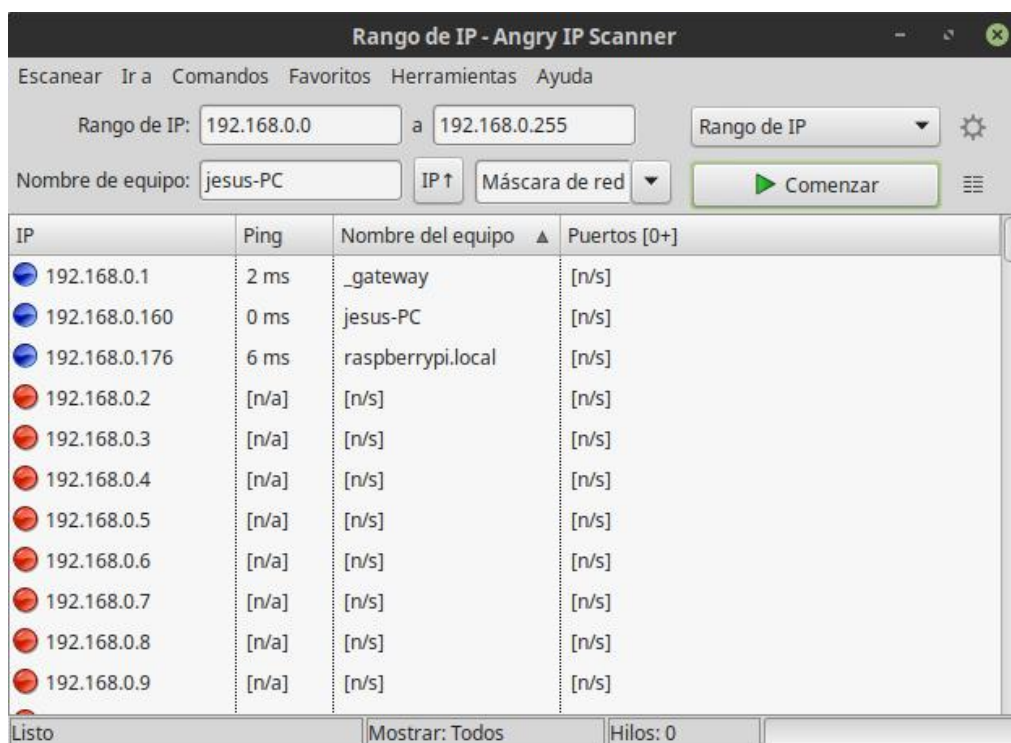


Imagen 4. Lista de IPs activas en la red encontradas por Angry IP Scanner

El usuario por defecto en la Raspberry es *pi* y la contraseña es *raspberry*. El comando a introducir es `ssh pi@192.168.0.176`. En la Imagen 5 se observa un inicio de sesión mediante SSH exitoso.

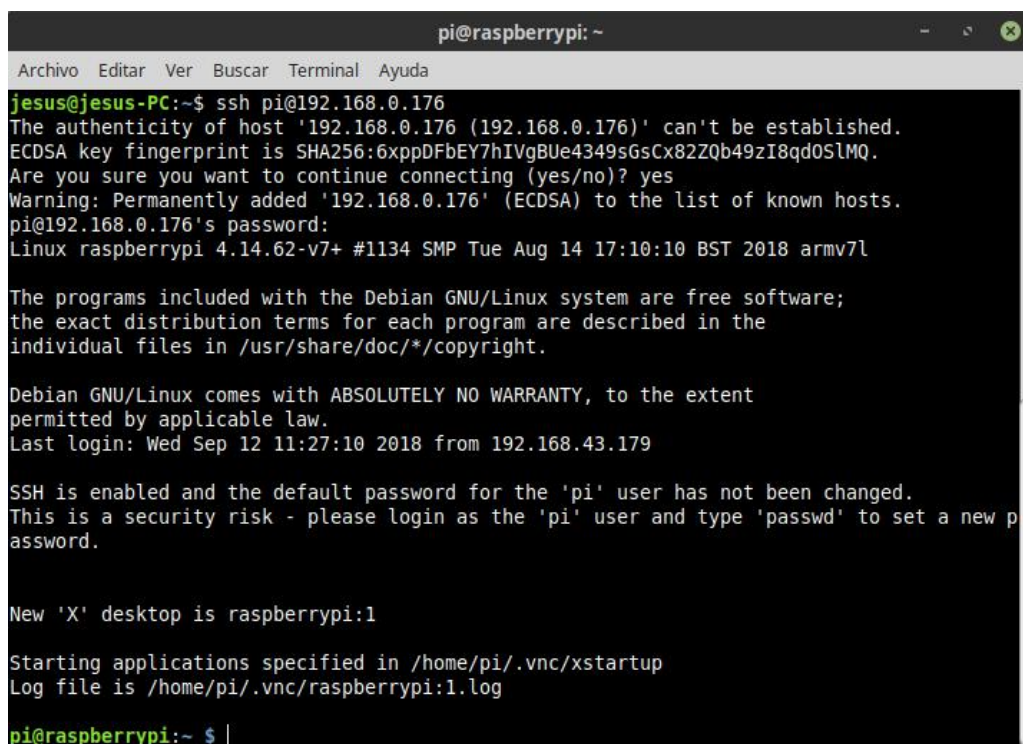


Imagen 5. Conexión mediante SSH

4.2. Activación y conexión al escritorio remoto mediante VNC

VNC es un software que sigue la arquitectura cliente/servidor y permite establecer una conexión a un equipo remoto para controlarlo y ver su pantalla en tiempo real, como si estuviese delante del usuario. Se hace uso de VNC para configurar y programar la Raspberry Pi. Esto proporciona más comodidad, pues podemos trabajar desde el mismo PC en el que se desarrolla el proyecto, y permite ahorrarse una conexión a otro monitor, teclado y ratón. Como se mencionó en el capítulo 2, para mayor información consultar las referencias [17] y [18].

4.2.1. Instalación y configuración del servidor vncserver

En primer lugar debemos instalar el servidor de escritorio remoto en la Raspberry Pi, en este trabajo hemos usado vncserver. Así, debemos ejecutar el siguiente comando:

```
sudo apt install vncserver
```

Editamos el archivo `~/.bashrc` para que automáticamente se cree el punto de acceso (en este caso el número 1):

```
vim ~/.bashrc
```

Debemos añadir al final del archivo la siguiente línea, como se observa en la Imagen 6.

```
vncserver :1 -geometry 1300x670
```

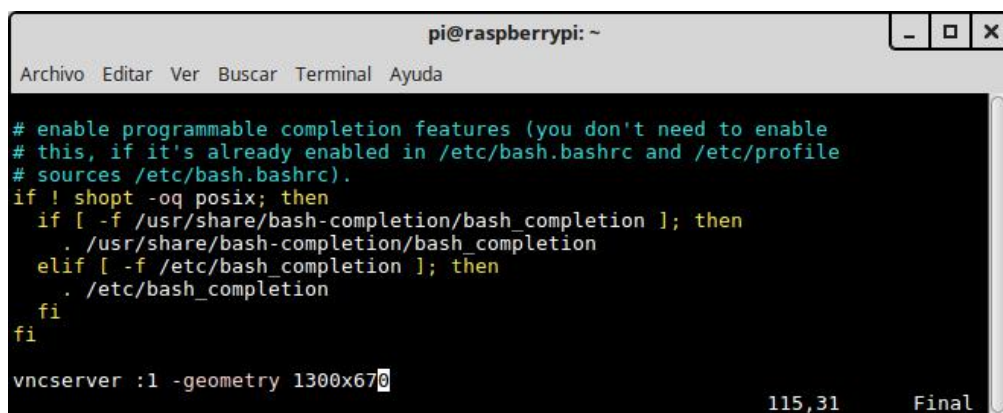


Imagen 6. Inicio automático del servidor VNC en la Raspberry Pi

Un último paso que debemos realizar es activar el uso de VNC en la configuración. Para ello usamos el siguiente comando, y seleccionamos la opción 5 (Interfacing options):

```
sudo raspi-config
```

4.2.2. Instalación del cliente xtightvncviewer

Instalamos el cliente en el PC desde el que nos conectaremos al escritorio de la Raspberry Pi mediante el comando:

```
sudo apt install xtightvncviewer
```

4.2.3. Conexión a la Raspberry Pi

Para conectarnos y poder trabajar con el escritorio de la Raspberry, necesitamos la IP mencionada en el apartado 4.1 y el punto de acceso, en nuestro caso el 1. Ejecutamos xtightvncviewer y en el cuadro de diálogo que aparece debemos escribir IP:puntodeacceso, como aparece en la Imagen 7.

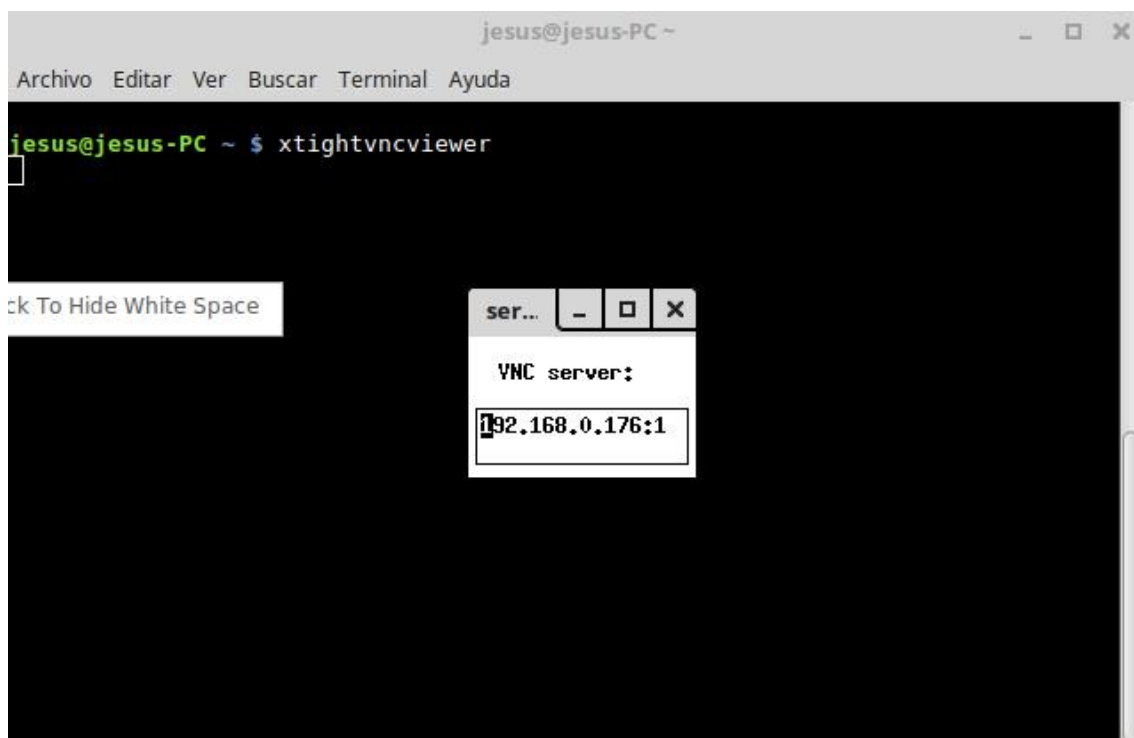


Imagen 7. Dirección IP y escritorio remoto al que se conecta xtightvncviewer

4.3. Descripción detallada de los bloques del sistema

En este apartado se describen los subsistemas implementados en Raspberry Pi para realizar la grabación del audio, el posterior reconocimiento de voz y la interfaz con Raspberry Pi necesaria para la apertura de la cerradura eléctrica.

4.3.1. Captura de audio

Como comentamos en el capítulo 2, usaremos la librería de Python PyAudio para realizar la grabación de audio. La instalamos con:

```
sudo apt install python-pyaudio
```

Pero si la instalamos directamente, nos aparecerá un error. Esto ocurre porque PyAudio es una implementación en Python de la librería PortAudio [9]. Ésta última no se encuentra instalada por defecto en Raspberry Pi, por lo que debemos instalarla primero (junto con todas sus dependencias) usando el comando:

```
sudo apt install libasound-dev portaudio19-dev libportaudio2  
libportaudiocpp0
```

Raspberry Pi no tiene entrada de audio, sólo cuenta con una salida jack. Por tanto, es necesario usar una tarjeta de sonido externa, a la que poder conectar un micrófono. En la Imagen 8 se muestra la usada en este trabajo, junto con el micrófono.

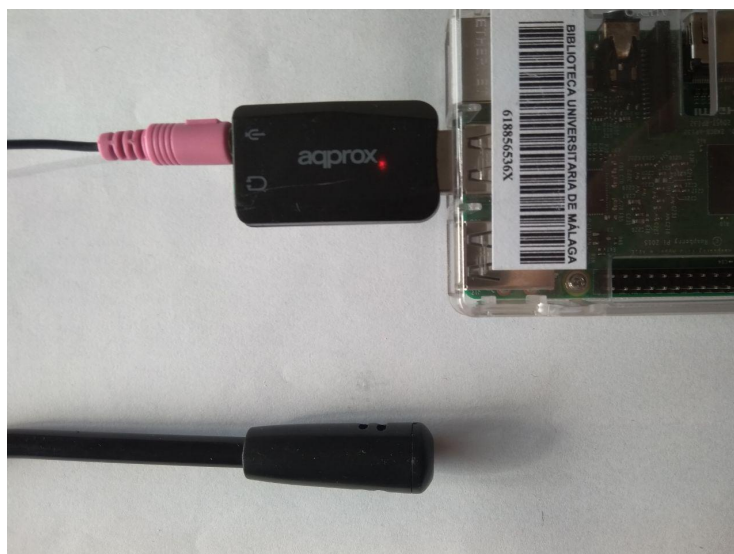
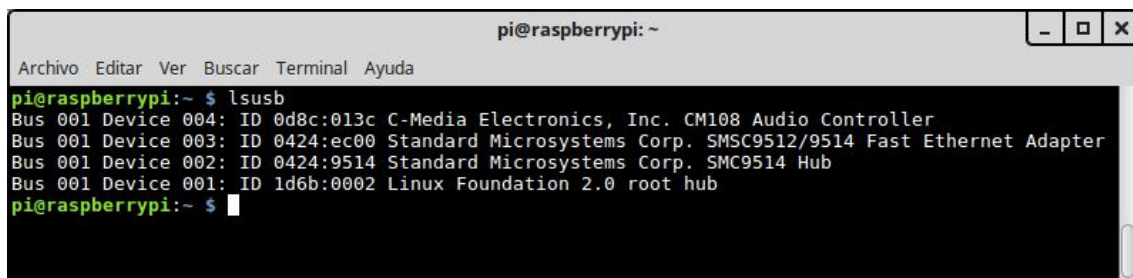


Imagen 8. Tarjeta de sonido y micrófono utilizados

Para comprobar que la Raspberry Pi la detecta correctamente, usamos el comando:

lsusb

En la Imagen 9, podemos observar como la controladora interna que usa la tarjeta es CM108, de C-Media Electronics.



```
pi@raspberrypi:~ $ lsusb
Bus 001 Device 004: ID 0d8c:013c C-Media Electronics, Inc. CM108 Audio Controller
Bus 001 Device 003: ID 0424:ec00 Standard Microsystems Corp. SMC9512/9514 Fast Ethernet Adapter
Bus 001 Device 002: ID 0424:9514 Standard Microsystems Corp. SMC9514 Hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
pi@raspberrypi:~ $
```

Imagen 9. Lista de dispositivos USB detectados

Aunque el puerto USB la reconoce, en principio encontramos que no era capaz de grabar audio. Hay que actualizar el archivo `~/.asoundrc` con el siguiente contenido:

```
pcm.!default {
    type asym
    playback.pcm {
        type plug
        slave.pcm "hw:0,0"
    }
    capture.pcm {
        type plug
        slave.pcm "hw:1,0"
    }
}
```

Este código configura los dispositivos de audio que se usarán por defecto para reproducción (*playback.pcm*) y para grabación (*capture.pcm*). Lo importante aquí es que el dispositivo de grabación por defecto será el "1,0" como

observamos en la última línea. Esto significa que es la tarjeta 1, y el dispositivo 0. Esta información la podemos extraer usando el siguiente comando, que lista todos los dispositivos de captura detectados (tanto internos, como externos en este caso):

```
arecord -l
```

En la Imagen 10 observamos el resultado de ejecutar este comando



```
pi@raspberrypi: ~
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
pi@raspberrypi:~ $ arecord -l
**** List of CAPTURE Hardware Devices ****
card 1: Device [USB PnP Sound Device], device 0: USB Audio [USB Audio]
  Subdevices: 1/1
    Subdevice #0: subdevice #0
pi@raspberrypi:~ $
```

Imagen 10. Lista de dispositivos de grabación detectados

En cuanto al código en Python, este bloque está implementado en *grabacion.py*. Pasamos a describir los aspectos más importantes.

Los parámetros de grabación son:

```
FORMATO=pyaudio.paInt16
CANALES=1
TASA_MUESTREO=16000
SEGMENTO=1024
DURACION=3
NOMBRE_ARCHIVO="audio.wav"
```

Como indicamos en el apartado de requisitos (Capítulo 3.2) el sistema está siempre escuchando, esperando que se supere cierto umbral, esto se realiza mediante el siguiente código.

```
UMBRAL = 3000
print "\nValor umbral actual: RMS =", UMBRAL
print "Esperando superar umbral...\n"
bytes = stream.read(SEGMENTO)
bloque = array('h', bytes)
valorRMS = RMS(bloque)
```

```

print "Valor RMS =", valorRMS
valorRMSmax = valorRMS
while (valorRMS < UMBRAL):
    bytes = stream.read(SEGMENTO)
    bloque = array('h', bytes)
    valorRMS= RMS(bloque)
    if valorRMS > valorRMSmax:
        valorRMSmax = valorRMS
    print "Valor RMS maximo detectado =",
valorRMSmax

```

Se está grabando continuamente un trozo de audio de SEGMENTO muestras (1024). Se calcula el valor cuadrático medio de la señal (RMS), y se comprueba si supera a la variable UMBRAL. Hasta que no tenga un valor mayor, el bucle seguirá iterando, y cuando termine empezará a grabar durante 3 segundos.

La función implementada que calcula el valor RMS de un segmento de audio es:

```

def RMS(vector):
    return sqrt( sum(n*n for n in bloque) / len(bloque) )

```

4.3.2. Uso de Google Cloud

Lo primero que debemos hacer es crear un nuevo proyecto en la plataforma de Google Cloud. Esto se muestra en la Imagen 11.

A continuación es necesario estar autenticados de cara a la plataforma. En la pestaña *API y Servicios* debemos seleccionar *Credenciales* y darle a *Crear Credenciales* (Imagen 12) y a continuación elegiremos el formato JSON (Imagen 13).

Google Cloud Platform

Nuevo proyecto

⚠

Te quedan 21 projects en la cuota. Solicita un aumento o elimina proyectos.
[Más información](#)
[MANAGE QUOTAS](#)

Nombre del proyecto *

Reconocimiento de voz - TFG

ID del proyecto: utility-operand-220320. No se puede cambiar más adelante.
[EDITAR](#)

Cuenta de facturación *

Mi cuenta de facturación

Todos los cargos de este proyecto se realizarán en la cuenta que selecciones aquí.

Ubicación *

Ninguna organización

[EXPLORAR](#)

Carpeta u organización principal

CREAR

CANCELAR

Imagen 11. Creación de nuevo proyecto en Google Cloud

API

Credenciales

Credenciales

Pantalla de autorización de OAuth

Verificación de dominio

Crear credenciales

Eliminar

Crea credenciales para acceder a tus API habilitadas. Si quieres obtener más información, [consulta la documentación sobre las API.](#)

Claves de cuenta de servicio

ID	Fecha de creación	Cuenta de servicio
<input type="checkbox"/> 9a39fb20766ae730dd962315fd9c509df1c8fe59	24 nov. 2017	Service Account

Imagen 12. Creación de credenciales para el proyecto en Google Cloud

25

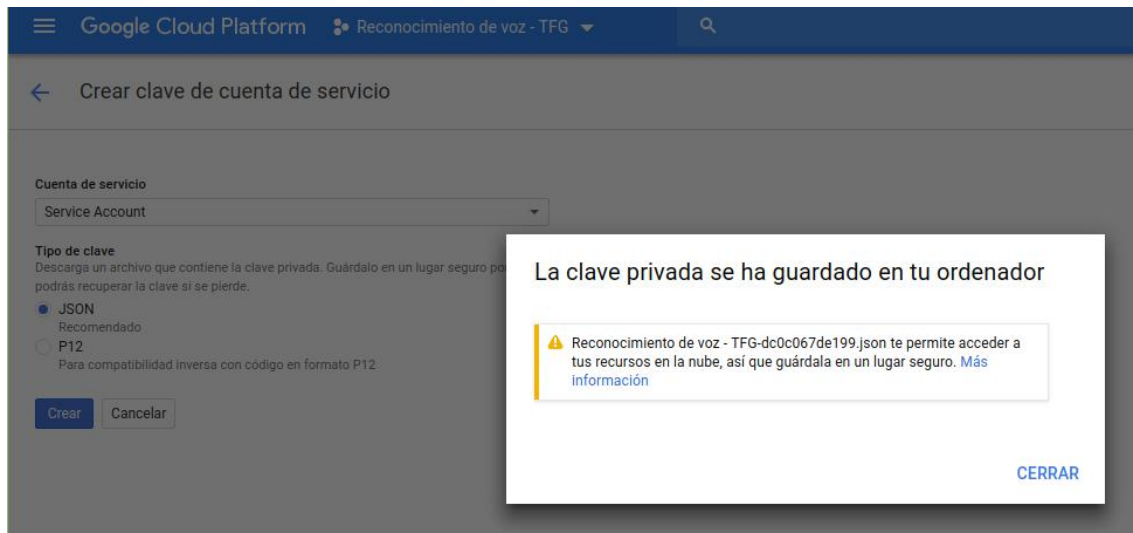


Imagen 13. Generación de clave privada en formato JSON

Esto nos genera un archivo de descarga con formato JSON, que debemos establecer como Variable de entorno. El programa entonces estará autenticado correctamente en Google Cloud. El comando usado es la siguiente (incluye la ruta al archivo, ésta podría variar):

```
export GOOGLE_APPLICATION_CREDENTIALS="private_key.json"
```

Aunque esto se podría ejecutar cada vez que se use, es mucho más cómodo como hicimos con vncserver, añadirlo al archivo ~/.bashrc, que se ejecuta al principio de cada inicio de sesión. Esto se muestra en la Imagen 14. En cualquier caso, si queremos asegurarnos de que está asignada correctamente, podemos listar las variables de entorno. Para ello sólo debemos ejecutar en un terminal *env*.

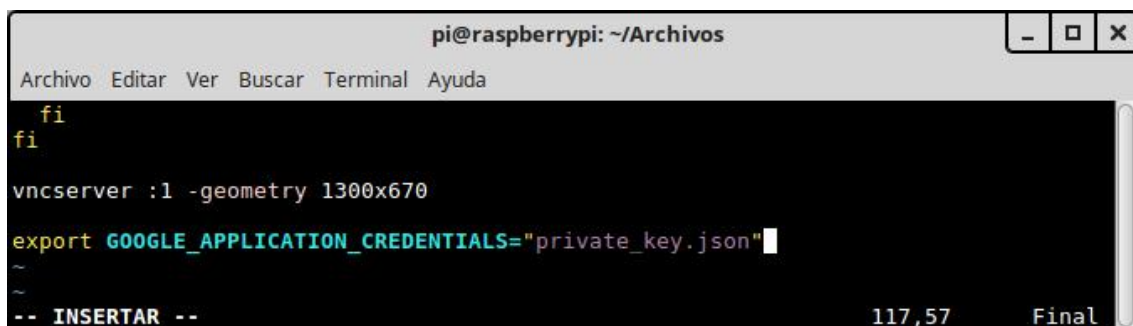


Imagen 14. Archivo ~/.bashrc tras añadir la variable de entorno

El módulo Python encargado del bloque de reconocimiento de voz es *reconocimiento.py*. Pasamos a describir los aspectos principales.

El siguiente bloque de código establece la configuración del audio que se enviará (codec, tasa de muestreo e idioma de transcripción)

```
config = types.RecognitionConfig(  
    Encoding                                     =  
    enums.RecognitionConfig.AudioEncoding.LINEAR16,  
    sample_rate_hertz = 44100,  
    language_code     = 'es-ES')
```

El siguiente bloque carga el archivo de audio en memoria, necesario para poder enviarlo posteriormente.

```
with io.open(ARCHIVO_ENTRADA, 'rb') as audio_file:  
    content = audio_file.read()  
    audio = types.RecognitionAudio(content=content)
```

Este bloque de código es el más importante. Se crea una instancia cliente, se le pasa los 2 parámetros anteriormente establecidos (configuración y archivo de audio), y se recibe el resultado en el String 'transcripcion'.

```
client = speech.SpeechClient()  
response = client.recognize(config, audio)  
transcripcion =  
response.results[0].alternatives[0].transcript
```

4.3.3. Verificación de contraseña

En el siguiente bloque de código se leen los archivos que contiene la contraseña del sistema, y la que se ha reconocido del audio.

```
fd_contrasena = open('contrasena.txt', 'r')  
contrasena = fd_contrasena.read()  
fd_transcripcion = open('transcripcion.txt', 'r' )  
introducida = fd_transcripcion.read()  
print "La contraseña introducida es:", introducida  
time.sleep(0.5)
```

Se comparan las cadenas y en caso de que coincidan se ejecuta el `activa_rele.py`, que como veremos en el próximo apartado se encarga de accionar la cerradura eléctrica.

```
if introducida == contrasena:
    print "Contrasena correcta => Puerta abierta\n"
    execfile("contrasena_correcta.py")
else:
    print "Contrasena incorrecta => Puerta cerrada\n"
    execfile("contrasena_incorrecta.py")
```

4.3.4. Apertura de cerradura eléctrica

La cerradura eléctrica que se ha usado se acciona cuando es conectada a una tensión de 12 V. Se ha usado para ello una pequeña pila.

Pero ocurre que no podemos conectar una tensión mayor de 3.3 V a la salida de un pin GPIO (en cuyo caso los dañaríamos dejando inutilizable la bahía entera de pines). Para solucionar esto, debemos conectar la salida del pin GPIO a la entrada de un relé (con tensión de activación igual a 3 V). Cuando la contraseña sea correcta, se pondrá un nivel alto en el pin GPIO, por lo que el relé conmutará y cerrará un circuito que conecta la pila con la cerradura eléctrica.

Una vez resuelto esto, el problema que nos encontramos, es que un pin GPIO no puede proporcionar la suficiente corriente como para conmutar el relé. El relé necesita como mínimo 120 mA de corriente en la bobina (parámetro Rated Coil Consumption en el datasheet del relé [26]). Y como mencionamos en 2.1.3, un pin GPIO puede proporcionar como máximo 16 mA. Por tanto, debemos poner un circuito que amplifique la corriente. A este efecto se ha colocado el circuito mostrado en la Imagen 15, basado en el transistor bipolar NPN 2222N.

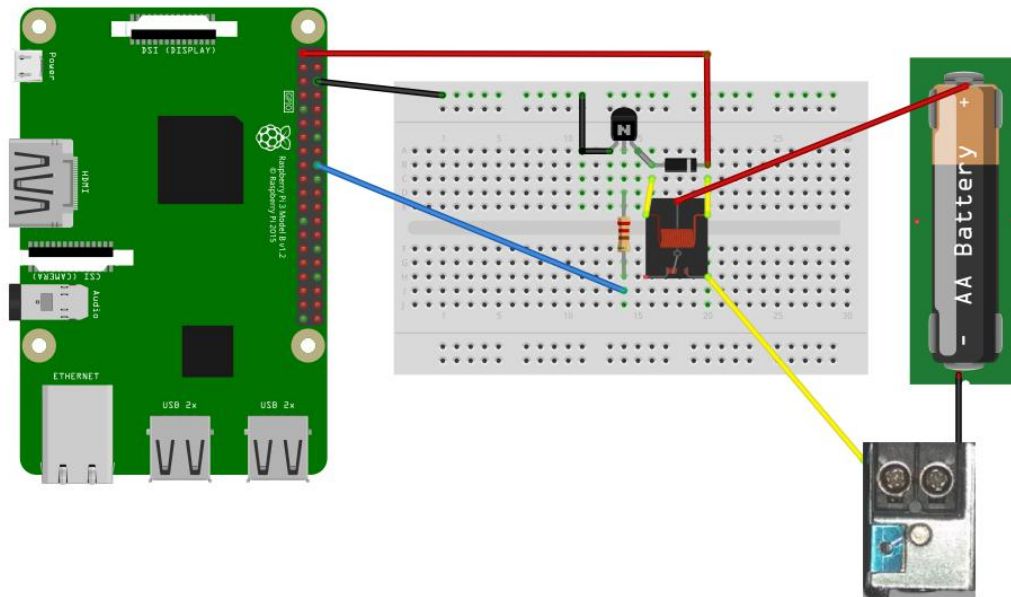


Imagen 15. Circuito de activación de cerradura eléctrica mediante relé

El pin GPIO 24 (cable azul) se conecta a la base del transistor a través de una resistencia de $1k\Omega$ para limitar la corriente. El emisor se conecta a tierra y el colector se conecta al pin que proporciona 3.3V fijos, a través de un diodo rectificador, así como a un lado de la bobina del relé. El otro lado de la bobina va conectado a los 3.3V.

Por otro lado, la cerradura eléctrica está conectada a un extremo de la pila, y al terminal Normalmente Abierto del relé. Por lo que cuando desde el pin 24 se mande una señal con nivel alto, el transistor amplificará la corriente y en los terminales de la bobina del relé se generará una tensión, que cerrará el circuito que conecta la pila de 12V con la cerradura eléctrica.

Sólo se enviará la señal a nivel alto al relé durante 1 segundo, y el código necesario es el siguiente (forma parte de *contrasena_correcta.py*):

```
SENAL_RELE = 24 #La senal que conmutara el rele
GPIO.setup(SENAL_RELE, GPIO.OUT)

GPIO.output(SENAL_RELE, 1)
time.sleep(1)
GPIO.output(SENAL_RELE, 0)
```

A continuación en la Imagen 16 se muestra el sistema completo montado en una protoboard.

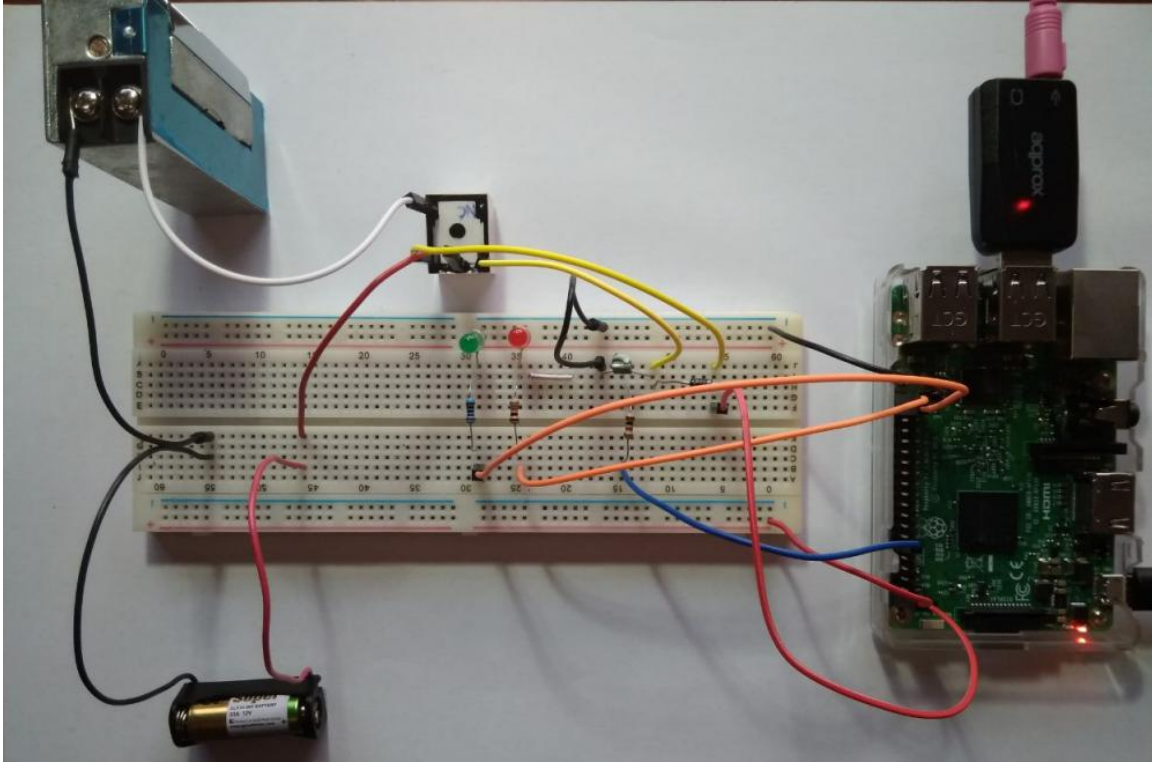


Imagen 16. Sistema completo montado en protoboard

4.4. Interfaz con el usuario

Desde un punto de vista general de interacción con el usuario, las 2 acciones fundamentales que puede realizar el sistema son:

- Modificar la contraseña de acceso.
- Realizar el proceso de identificación.

Cada una está implementada de forma separada en los archivos *Modifica_Contrasena.py* e *Identificacion.py*, respectivamente.

4.4.1. LEDs de indicación

Se han instalado 2 leds para notificar visualmente al usuario si la identificación realizada es correcta o incorrecta.

Si la verificación de contraseña ha sido correcta, se encenderá el led verde, en caso contrario el led rojo. Los leds se deben conectar a los pines GPIO a través de resistencias, con el objetivo de limitar la corriente. Se han usado resistencia de $1k\Omega$. Como la tensión que proporciona un pin GPIO es 3.3 V y la tensión que cae en el led es aproximadamente 2V , la tensión que caerá en la resistencia será

$$V_r = 3.3 - 2 = 1.3\text{V}$$

La intensidad que cirula por la resistencia y por tanto por el led será

$$I_{led} = \frac{V_R}{R} = \frac{1.3}{1000} = 1.3\text{ mA}$$

la cual es suficiente para encender un led.

En la imagen 17 se observa el diagrama de conexión de los leds a la Raspberry Pi. Por brevedad, sólo expondremos el código para encender el led verde, pues para el led rojo es completamente análogo sin más que cambiar el número de pin GPIO.

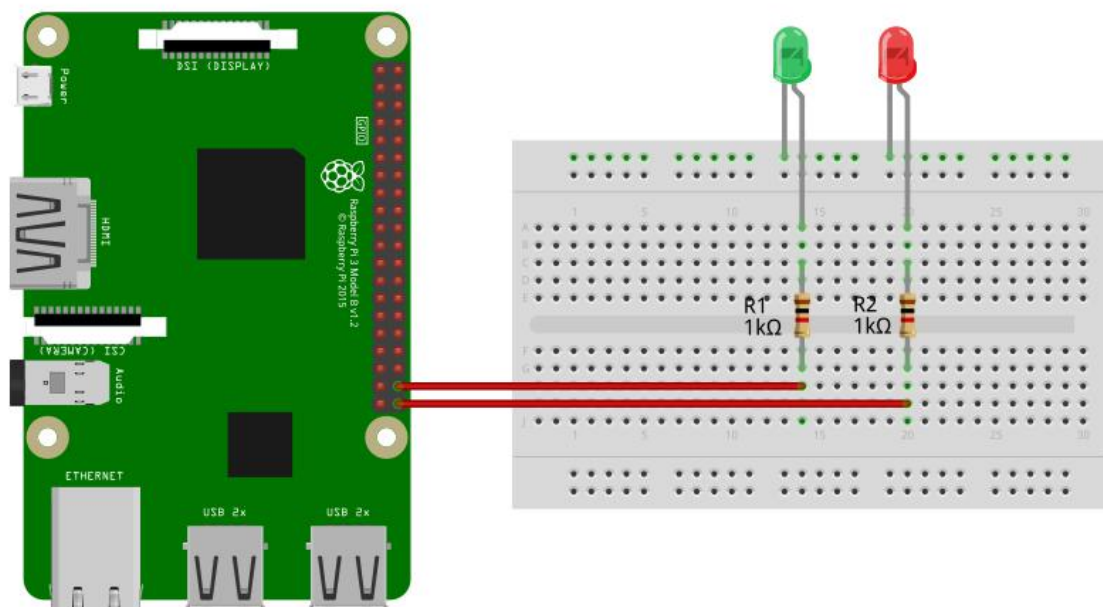


Imagen 17. Leds verde y rojo conectados a los pines GPIO 20 y 21 respectivamente

El código necesario para asociar el led verde al pin 20 y mantenerlo encendido 1 segundo es:

```
LED_VERDE= 20
GPIO.setup(LED_VERDE, GPIO.OUT)
GPIO.output(LED_VERDE, 1)
time.sleep(1)
GPIO.output(LED_VERDE, 0)
```

Este código forma parte de *contrasena_correcta.py*. Análogamente el código para el led rojo forma parte de *contrasena_incorrecta.py*.

4.4.2. Zumbador

Se ha usado un zumbador piezoeléctrico para avisar con una señal acústica al usuario. En la Imagen 18 se muestra el diagrama de conexión. El zumbador se conecta directamente a un pin GPIO, en este caso el 18. El código para vincular el zumbador al pin 18 es:

```
ZUMBADOR = 18
GPIO.setup(ZUMBADOR, GPIO.OUT)
```

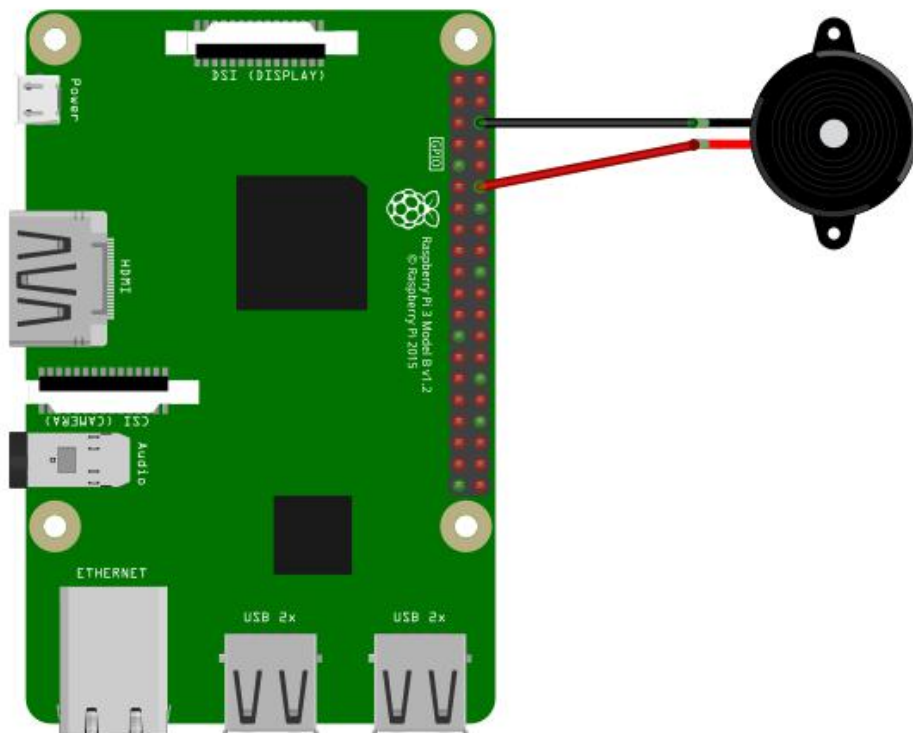


Imagen 18. Zumbador conectado al pin GPIO 18

Si la contraseña es correcta el zumbador emitirá un pitido durante 1 segundo.
El siguiente código lo implementa (pertenece a *contrasena_correcta.py*):

```
GPIO.output(ZUMBADOR, 1)
time.sleep(1)
GPIO.output(ZUMBADOR, 0)
```

Y si es incorrecta emitirá 3 pitidos cortos de 0.2 segundos de duración
cada uno. El siguiente código lo implementa (forma parte de
contrasena_incorrecta.py)

```
for i in range(3):
    GPIO.output(ZUMBADOR, 1)
    time.sleep(0.2)
    GPIO.output(ZUMBADOR, 0)
    time.sleep(0.2)
```

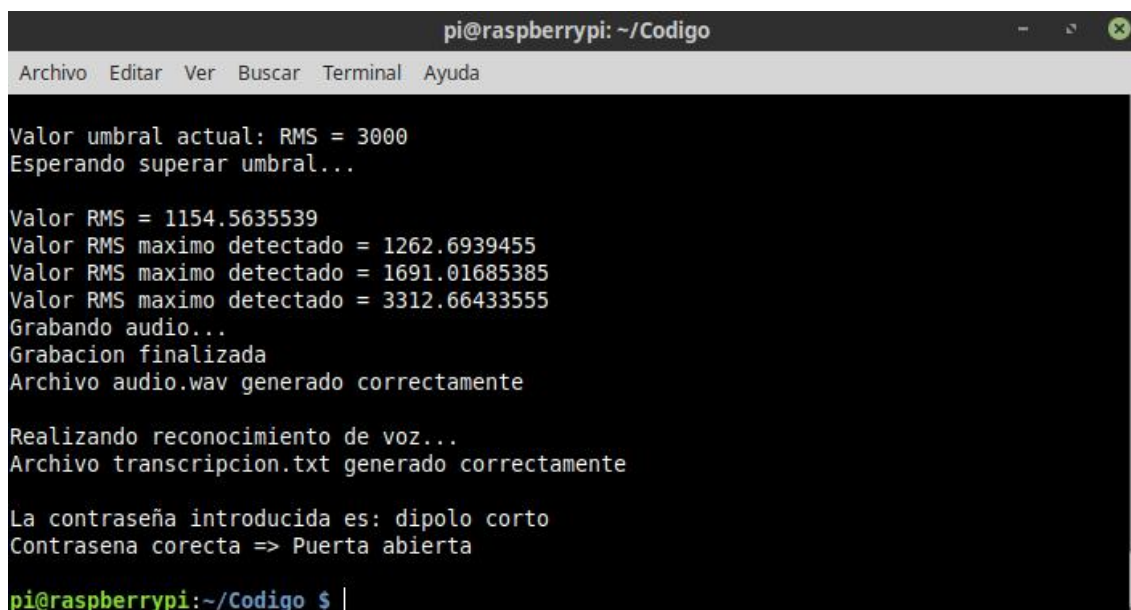

Capítulo 5. Evaluación del sistema

En este capítulo se describen las pruebas de funcionamiento realizadas y se hace un cálculo del consumo del sistema.

5.1. Funcionamiento

Se han realizado las siguientes pruebas de funcionamiento del sistema:

- **Contraseña correcta.** Si se pronuncia la misma contraseña que está escrita en el archivo *contrasena.txt*, la transcripción coincide, la cerradura se activa correctamente (la puerta se abriría), así como el led verde se enciende y el zumbador emite un pitido largo, el resultado esperado. En la Imagen 19 se observa la ejecución del programa.



```
pi@raspberrypi: ~/Codigo
Archivo  Editar  Ver  Buscar  Terminal  Ayuda

Valor umbral actual: RMS = 3000
Esperando superar umbral...

Valor RMS = 1154.5635539
Valor RMS maximo detectado = 1262.6939455
Valor RMS maximo detectado = 1691.01685385
Valor RMS maximo detectado = 3312.66433555
Grabando audio...
Grabacion finalizada
Archivo audio.wav generado correctamente

Realizando reconocimiento de voz...
Archivo transcripcion.txt generado correctamente

La contraseña introducida es: dipolo corto
Contrasena corecta => Puerta abierta

pi@raspberrypi:~/Codigo $ |
```

Imagen 19. Identificación correcta en el sistema

- **Contraseña incorrecta.** Si pronunciamos cualquier contraseña diferente a la que está escrita en el archivo *contrasena.txt* , la cerradura sigue inactivada (la puerta seguiría bloqueada), así como el led rojo se enciende y el zumbador emite 3 pitidos cortos, el resultado esperado. En la Imagen 20 se observa el resultado de la ejecución del programa.

```

pi@raspberrypi: ~/Codigo
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
Esperando superar umbral...
Valor RMS = 457.551090044
Valor RMS maximo detectado = 501.460865871
Valor RMS maximo detectado = 768.936928493
Valor RMS maximo detectado = 1123.48965282
Valor RMS maximo detectado = 2186.42424977
Valor RMS maximo detectado = 3125.80549619
Grabando audio...
Grabacion finalizada
Archivo audio.wav generado correctamente

Realizando reconocimiento de voz...
Archivo transcripcion.txt generado correctamente

La contraseña introducida es: cualquier contraseña
Contrasena incorrecta => Puerta cerrada

pi@raspberrypi:~/Codigo $ |

```

Imagen 20. Identificación incorrecta en el sistema

- **Variación de la variable *UMBRAL*.** Cada vez que se va a realizar una grabación (cada vez que se ejecuta el archivo *grabacion.py*), se espera a que el nivel de sonido detectado por el micrófono sea mayor que un cierto umbral. El valor que se compara no es el valor exacto leído en cada instante, sino que se van tomando segmentos de 1024 bytes y se calcula su valor RMS (para más detalle sobre esto, ir al capítulo 4.3.1). Así, se ha experimentado con distintos valores de umbral, ya que si este es muy bajo, con cualquier ruido del ambiente la grabación se podría disparar; y si es demasiado alto, habría que elevar la voz innecesariamente. Se ha encontrado como comentamos empíricamente, que 3000 es un buen valor para este parámetro.
- **Características de la contraseña.** Una gran fuente de imprevistos fue el contenido de la contraseña. Se ha probado con contraseñas que son una sola palabra y con varias palabras, ambas con resultados positivos. Con el caso de los números, la API de Google suele transcribir un número

pronunciado en formato texto (“tres” en vez de “3”) por lo que si hemos puesto que nuestra contraseña tenga caracteres numéricos, esto podría resultar en una identificación errónea.

Por otro lado, inicialmente si la contraseña contiene alguna palabra con tilde el sistema fallaba, esto es así porque no se detectaban como válidos los caracteres con tilde. Para solucionar esto simplemente debemos usar la codificación de caracteres UTF-8, la primera línea del archivo contiene dicha codificación. Para más información visitar [28].

5.2. Consumo

Según varias fuentes [29] [30], la Raspberry Pi consume en reposo unos 400 mA. Teniendo en cuenta que la fuente de alimentación es de 5 V, el consumo de potencia en reposo será aproximadamente:

$$P = V * I = 5 * 0,4 = 2 \text{ W}$$

Por lo que si está todo el tiempo encendida, tenemos que el consumo de energía en un mes aproximadamente sería:

$$E = 0,002 \text{ kW} * 24 \text{ h} * 30 \text{ días} = 1,44 \text{ kWh}$$

El tiempo que el sistema estaría fuera del reposo se estima en minutos, por tanto es muy pequeño en comparación al tiempo total; en cualquier caso la Raspberry Pi en funcionamiento sólo requiere unas decenas de mA más (siempre refiriéndonos a este proyecto en particular, donde no hay una gran carga de CPU ni uso de gráficos).

Capítulo 6. Conclusiones y trabajo futuro

Se ha implementado satisfactoriamente el sistema planteado, no sin ello solventar antes diversos imprevistos aprendiendo en el camino, a saber principalmente, con las librerías necesarias para capturar audio, el sistema de codificación de caracteres, los parámetros de grabación y la electrónica necesaria para que la Raspberry Pi pueda actuar sobre un dispositivo a 12 V, como es una cerradura eléctrica (relé y transistor como amplificador de corriente). Consideramos que el objetivos indicado en el apartado 1.2 se ha cumplido adecuadamente.

- Se ha aprendido un nuevo lenguaje, Python.
- Se ha trabajado con la Raspberry Pi, instalando varias librerías y software, realizado configuración, entrada de audio, se han usado los pines GPIO, conectado en remoto por SSH y por escritorio remoto, y por supuesto ejecución de código Python.
- Se ha realizado una aplicación usando la plataforma de computación en la nube Google Cloud, la cual ha suscitado al autor un gran interés por las posibilidades que este moderno paradigma de desarrollo permite.

Como posibles mejoras del proyecto, se vislumbran 2 principalmente. La primera es que el sistema tenga capacidad de distinguir entre diferentes hablantes, en cuyo caso el sistema no sólo identificaría mediante contraseña. La segunda es una implementación en Raspberry Pi Zero, un modelo que permitiría reducir el coste (su precio es entre 12 y 15 €) y tamaño del sistema.

Apéndice A. Código del programa

Identificacion.py

```
#####  
#####  
  
#Codigo que realiza el proceso completo de identificacion,  
ejecutando otros archivos  
  
# python y verificando la contrasena. Se usa la libreria  
PyAudio  
  
#####  
#####  
  
# -*- coding: utf8 -*-  
import time  
  
print  
"  
_____  
_____  
"  
print "Identifícate pronunciando la contraseña"  
raw_input("Presiona enter...")  
  
#1. Grabacion  
execfile("grabacion.py")  
print "Archivo audio.wav generado correctamente\n"  
time.sleep(0.5)  
  
#2. Reconocimiento de voz  
execfile("reconocimiento.py")  
print "Archivo transcripcion.txt generado correctamente\n"  
time.sleep(0.5)
```

```

#3. Lectura de contraseña del sistema
fd_contrasena = open('contrasena.txt', 'r' )
contrasena = fd_contrasena.read()

#4. Lectura de contraseña introducida
fd_transcripcion = open('transcripcion.txt', 'r' )
introducida = fd_transcripcion.read()

#5. Si la contraseña es correcta, activa el relé (durante
1 segundo)
print "La contraseña introducida es:", introducida
time.sleep(0.5)

if introducida == contrasena:
    print "Contrasena corecta => Puerta abierta\n"
    execfile("contrasena_correcta.py")
else:
    print "Contrasena incorecta => Puerta cerrada\n"
    execfile("contrasena_incorrecta.py")

```

Modifica_Contrasena.py

```

#####
#Codigo que realiza la modificacion de contrasena,
# comprobando que la transcripcion del reconocimiento # de
voz coincida correctamente con la introducida por #
teclado
#####
# -*- coding: utf8 -*-
import time

#1. Introduccion nueva contraseña
print
"
_____
"
_____

```

```

print "Cambio de contraseña. Pulsa ENTER para introducir
una nueva..."
nueva_contrasena = raw_input("")

```

```

#2. Realiza la transcripcion de voz de la nueva contrasena
#2(a). Grabacion de audio
raw_input("A continuacion, pronuncia la contraseña en voz
alta. Si la transcripcion no coincide con la que
introdujiste, no se podra actualizar y deberas volver a
intentarlo\n")
execfile("grabacion.py")
print "Archivo audio.wav generado correctamente\n"
time.sleep(0.5)
#2(b). Reconocimiento de voz
execfile("reconocimiento.py")
print "Archivo transcripcion.txt generado correctamente\n"
time.sleep(0.5)
#2(c). Lectura de la transcripcion
fd_transcripcion = open('transcripcion.txt', 'r' )
transcripcion = fd_transcripcion.read()

```

```

#3. Si la transcripcion del audio coincide con la
contrasena escrita => La actualiza
if transcripcion == nueva_contrasena:
    print "La transcripcion coincide con la contraseña
introducida. Contraseña actualizada."
    archivo = open("contrasena.txt", "w")
    archivo.write(transcripcion.encode('utf-8'
'replace') )
    archivo.close()
else:
    print "La transcripcion no coincide con la contraseña
introducida. La contraseña no se pudo actualizar,
intentelo de nuevo"

```

grabacion.py

```
#####
# Código que realiza la captura de audio. Se usa la
# librería PyAudio
#####
# -*- coding: utf8 -*-
#0. Importar librerías
import pyaudio
import wave
from array import array
import time
from math import sqrt

def RMS(vector):
    return sqrt( sum(n*n for n in bloque) / len(bloque) )

#1. Parametros de la grabacion
FORMATO=pyaudio.paInt16
CANALES=1
TASA_MUESTREO=16000
SEGMENTO=1024
DURACION=3
NOMBRE_ARCHIVO="audio.wav"

#2. Inicialización de PyAudio
audio = pyaudio.PyAudio()
stream = audio.open(format = FORMATO,
                    channels = CANALES,
                    rate = TASA_MUESTREO,
                    input = True,
                    frames_per_buffer = SEGMENTO)

frames=[]

#3. Se estará "capturando" todo el rato el sonido con el
microfono (pero sin grabarlo) y comprobando si supera un
umbral...
```

```

UMBRALE = 3000
print "\nValor umbral actual: RMS =", UMBRALE
print "Esperando superar umbral...\n"

bytes = stream.read(SEGMENTO)
bloque = array('h', bytes) #Necesitamos poner los bytes
grabados por el microfono en forma de array, para hallar
el max
valorRMS= RMS(bloque)
print "Valor RMS =", valorRMS
valorRMSmax = valorRMS

while (valorRMS < UMBRALE):
    bytes = stream.read(SEGMENTO)
    bloque = array('h', bytes)
    valorRMS= RMS(bloque)
    if valorRMS > valorRMSmax:
        valorRMSmax = valorRMS
        print "Valor RMS maximo detectado =",
valorRMSmax

#4. ...Y cuando se supere este umbral (se anula la
condicion del while) empezaremos a guardar los datos
print "Grabando audio..."
for i in range(0, TASA_MUESTREO/SEGMENTO * DURACION):
    bytes = stream.read(SEGMENTO)
    frames.append(bytes)

print("Grabacion finalizada")
time.sleep(0.5)

#5. Termina la grabacion
stream.stop_stream()

```

```

stream.close()
audio.terminate()
#6. Escritura a archivo
wavfile=wave.open(NOMBRE_ARCHIVO, 'wb')
wavfile.setnchannels(CANALES)
wavfile.setsampwidth(audio.get_sample_size(FORMATO))
wavfile.setframerate(TASA_MUESTREO)
wavfile.writeframes(b''.join(frames))
wavfile.close()

```

reconocimiento.py

```

#####
#Codigo que toma el audio grabado, lo envia mediante # la
API de Google, y recibe el resultado como String
# Necesario instalar la libreria de Google =>
# sudo pip install --upgrade google-cloud-speech
# (La otra opcion seria hacerlo usando REST
# https://cloud.google.com/apis/docs/overview)
#####
# -*- coding: utf8 -*-

#0. Importar librerias
import io
import os
from google.cloud          import speech
from google.cloud.speech   import enums
from google.cloud.speech   import types

#1. Nombres de archivo
ARCHIVO_ENTRADA = "audio.wav"
ARCHIVO_SALIDA="transcripcion.txt"

#2. Establece la configuracion para la instruccion de
reconocimiento de voz
config = types.RecognitionConfig(

```



```

        encoding =
enums.RecognitionConfig.AudioEncoding.LINEAR16,
        sample_rate_hertz = 16000,
        language_code = 'es-ES')

```

#3. Carga el audio en memoria

```

with io.open(ARCHIVO_ENTRADA, 'rb') as audio_file:
    content = audio_file.read()
    audio = types.RecognitionAudio(content=content)

```

#4. Realiza el reconocimiento de la voz con la configuracion indicada, sobre el audio indicado

```

print "Realizando reconocimiento de voz..."
client = speech.SpeechClient()
response = client.recognize(config, audio)
transcripcion = response.results[0].alternatives[0].transcript
transcripcion = transcripcion.strip() #Aplicamos el
metodo strip() para eliminar los (posibles) espacios
iniciales y finales

```

#5. Escribe a archivo de texto

```

archivo= open(ARCHIVO_SALIDA, "w")
archivo.write(transcripcion.encode('utf-8' , 'replace') )
archivo.close()

```

contrasena_correcta.py

```
#####  
# Hace sonar el zumbador con un pitido largo  
#####  
  
import RPi.GPIO as GPIO  
import time  
  
GPIO.setmode(GPIO.BCM)  
LED_VERDE = 20  
ZUMBADOR = 18  
SENAL_RELE = 24 #La senal de control que conmutara el  
rele  
GPIO.setup(LED_VERDE, GPIO.OUT)  
GPIO.setup(ZUMBADOR, GPIO.OUT)  
GPIO.setup(SENAL_RELE, GPIO.OUT)  
  
try:  
    GPIO.output(LED_VERDE, 1)  
    GPIO.output(ZUMBADOR, 1)  
    GPIO.output(SENAL_RELE, 1)  
    time.sleep(1)  
    GPIO.output(LED_VERDE, 0)  
    GPIO.output(ZUMBADOR, 0)  
    GPIO.output(SENAL_RELE, 0)  
except KeyboardInterrupt:  
    GPIO.cleanup()  
    print "\nPines GPIO puestos en estado seguro  
correctamente"  
  
GPIO.cleanup()
```

contrasena_incorrecta.py

```
#####  
# Hace sonar 3 veces de forma discontinua el zumbador  
#####  
  
import RPi.GPIO as GPIO  
import time  
  
GPIO.setmode(GPIO.BCM)  
LED_ROJO = 21  
ZUMBADOR = 18  
GPIO.setup(LED_ROJO, GPIO.OUT)  
GPIO.setup(ZUMBADOR, GPIO.OUT)  
  
try:  
    GPIO.output(LED_ROJO, 1)  
    for i in range(3):  
        GPIO.output(ZUMBADOR, 1)  
        time.sleep(0.2)  
        GPIO.output(ZUMBADOR, 0)  
        time.sleep(0.2)  
    GPIO.output(LED_ROJO, 0)  
except KeyboardInterrupt:  
    GPIO.cleanup()  
    print "\nInterrupcion, has pulsado CTRL-C. Pines GPIO  
reseteados correctamente"  
  
GPIO.cleanup()
```

Referencias

- [1] Raspberry Pi Foundation, “Chipset BCM2837”
<https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2837/README.md>
- [2] Raspberry Pi Foundation, “USB Power limits”
<https://www.raspberrypi.org/documentation/hardware/raspberrypi/usb/README.md#powerlimits>
- [3] Raspberry Pi Foundation, “Raspbian”
<https://www.raspberrypi.org/downloads/raspbian/>
- [4] Raspberry Pi Foundation, “Installing Operating systems images”
<https://www.raspberrypi.org/documentation/installation/installing-images/README.md>
- [5] Raspberry Pi Foundation, “GPIO”
<https://www.raspberrypi.org/documentation/usage/gpio/>
- [6] Simon Monk, “Ejercicios prácticos con Raspberry Pi”. Marcombo, 2017.
- [7] “Raspberry Pi Pinout”
<https://es.pinout.xyz/>
- [8] Python Software Foundation, “Python 2.7.15 documentation”
<https://docs.python.org/2/index.html>
- [9] PyAudio library documentation <https://people.csail.mit.edu/hubert/pyaudio/>
- [10] Google, “Speech-to-text Client Libraries” <https://cloud.google.com/speech-to-text/docs/reference/libraries>

- [11] RPi.GPIO library documentation
<https://sourceforge.net/p/raspberry-gpio-python/wiki/Examples/>
- [12] Microsoft Research, "Achieving human parity in conversational speech recognition", febrero 2017
<https://arxiv.org/pdf/1610.05256.pdf>
- [13] "IBM beats Microsoft's word error rate in speech recognition, achieves 5.5%", marzo 2017
<https://www.neowin.net/news/ibm-beats-microsofts-word-error-rate-in-speech-recognition-achieves-55/>
- [14] "Google's speech recognition technology now has a 4.9% word error rate", mayo 2017
<https://venturebeat.com/2017/05/17/googles-speech-recognition-technology-now-has-a-4-9-word-error-rate/>
- [15] Google, "Cloud Speech to text documentation"
<https://cloud.google.com/speech-to-text/docs/>
- [16] Google, web principal de Cloud Text to Speech
<https://cloud.google.com/text-to-speech/>
- [17] Manual vncserver
<https://linux.die.net/man/1/vncserver>
- [18] Manual de tightvncviewer
<https://www.tightvnc.com/vncviewer.1.php>
- [19] Página web de Angry IP Scanner
<https://angryip.org/>
- [20] Manual de Vim
<https://linux.die.net/man/1/vi>

- [21] Página web de Fritzting
<http://fritzing.org/home/>
- [22] Referencia en Stackoverflow respecto a la funcionalidad de reconocimiento del hablante en GCS
<https://stackoverflow.com/questions/50847412/google-cloud-speech-to-text-api-multi-speaker-recognition>
- [23] Google, “Separating different audio speakers in an audio recording”
<https://cloud.google.com/speech-to-text/docs/multiple-voices>
- [24] Cloud Speech-to-Text Basics
<https://cloud.google.com/speech-to-text/docs/basics>
- [25] Transcribing Short Audio Files
<https://cloud.google.com/speech-to-text/docs/sync-recognize>
- [26] Datasheet Finder Relay 36 Series
<https://www.finder-relais.net/en/finder-relays-series-36.pdf>
- [27] Definition of Relay Technology
<https://media.digikey.com/pdf/Other%20Related%20Documents/Panasonic%20Other%20Doc/Small%20Signal%20Relay%20Technical%20Info.pdf>
- [28] The Python Software Foundation, “Unicode HOWTO”. 2010
<https://docs.python.org/2/howto/unicode.html>
- [29] Power consumption benchmarks
<https://www.pidramble.com/wiki/benchmarks/power-consumption>
- [30] How Much Power Does Raspberry Pi 3B+ Use? Power Measurements
<https://raspi.tv/2018/how-much-power-does-raspberry-pi-3b-use-power-measurements>