UTN FACULTAD REGIONAL MENDOZA
UNIVERSIDAD TECNOLÓGICA NACIONAL

CÁTEDRA DE PROYECTO FINAL / INGENIERÍA EN ELECTRÓNICA

# *SMART INTERFONDOMOTIC HOUSE CON RECONOCIMIENTO FACIAL BIOMETRICO (SIHRFB)*

# PROYECTO

Versión 1.0

12/03/2024

## INFORMACIÓN DEL PROYECTO

| **Autor** | |
|---|---|
| Nombre Completo del integrante 1 | David Abdala |
| Legajo | 42866 |
| e-mail | davidabdalaa@gmail.com |

| | |
|---|---|
| Tutor | Ing. Sebastián Tobar |
| Director | Ing. Sebastián Tobar |
| Jurado | Ing. Sebastián Tobar |
| Año Académico | 2023 |
| Responsable de la cátedra | Esp. Ing. Antonio Álvarez Abril / Ing. Ana Latucca |

| | |
|---|---|
| Empresa / Cliente / Laboratorio | Publico General |
| Patrocinador (Sponsor) | Sin Patrocinador |

# 1 RESUMEN DEL PROYECTO

## 1.1 RESUMEN

Presentamos el Smart Interfondomotic House, una solución innovadora que simplifica la gestión del hogar mediante una aplicación móvil conectada a Internet de uso diario, siendo la misma Telegram. Este sistema inteligente te permite controlar una variedad de funciones domésticas, desde el timbre hasta las aperturas de puertas, encendido de luces, incluso el acceso con reconocimiento facial biométrico y variedad de funciones más. Con notificaciones instantáneas, estarás al tanto de cualquier actividad en tu hogar y podrás tomar medidas desde cualquier lugar.

La seguridad al ingreso de la propiedad se refuerza con el reconocimiento facial, asegurando que solo personas autorizadas tengan acceso. Esta convergencia de comodidad y seguridad hace que la administración del hogar sea más eficiente y conveniente. Con el Smart Interfondomotic House, se podrá tener un control total sobre tu entorno doméstico, mejorando tu calidad de vida y proporcionándote tranquilidad dondequiera que estés.

## 1.2 SUMMARY

We present the Smart Interfondomotic House, an innovative solution that simplifies home management through a mobile application connected to the Internet for daily use, Telegram itself. This smart system allows you to control a variety of home functions, from the doorbell to door openings, turning on lights, even access with biometric facial recognition and a variety of other functions. With instant notifications, you will be aware of any activity in your home and can take action from anywhere.

Property entry security is reinforced with facial recognition, ensuring that only authorized people have access. This convergence of comfort and security makes home management more efficient and convenient. With the Smart Interfondomotic House, you can have total control over your home environment, improving your quality of life and providing you with peace of mind wherever you are.

# 2 PALABRAS CLAVES

Telegram, Automatización, IoT, WiFi, Reconocimiento Facial.

# 3 ÍNDICE

## Contenido

# 4 INTRODUCCIÓN

## 4.1 IDEA Y DESCRIPCIÓN DEL PROYECTO

El proyecto consiste en la automatización y manejo de varios aparatos de la casa, específicamente del timbre, en el cual al momento de recibir una señal de presencia (debido a un sensor de presencia), o simplemente al apretar el botón del timbre para dar aviso que hay alguien afuera, esté mismo enviara a una señal a un microcontrolador el cual se encargara de enviar un mensaje de texto (aviso de presencia) más una foto a la aplicación del móvil Telegram. Esto permitirá observar quien se encuentra afuera de casa (gracias a una cámara integrada que tendrá el timbre), decidiendo que hacer, si responder o no. En el caso de responder a través de la aplicación móvil, se podrá pedir otra foto, abrir la puerta, prender alguna luz para ver mejor y tener comunicación bidireccional de audio a través de envíos de audio de voz.

Además de esto, el timbre optará por tener reconocimiento facial biométrico, esto me permitirá que a la hora de tener que ingresar a la casa, con solo mostrar el rostro frente a la cámara, se podrá activar la apertura de la puerta de la casa y así ingresar más fácilmente.

Por otra parte, de las funcionalidades extras de la aplicación, se podrá abrir el garaje/puerta a través de la aplicación con solo enviar un mensaje desde el bot de Telegram (ósea no importa que tan lejos uno se encuentre, simplemente hay que tener internet).

Todo esto se armó y programo con una placa de desarrollo denominada Raspberry Pi, en conjunto con la programación en Python, OpenCV y API de Telegram.

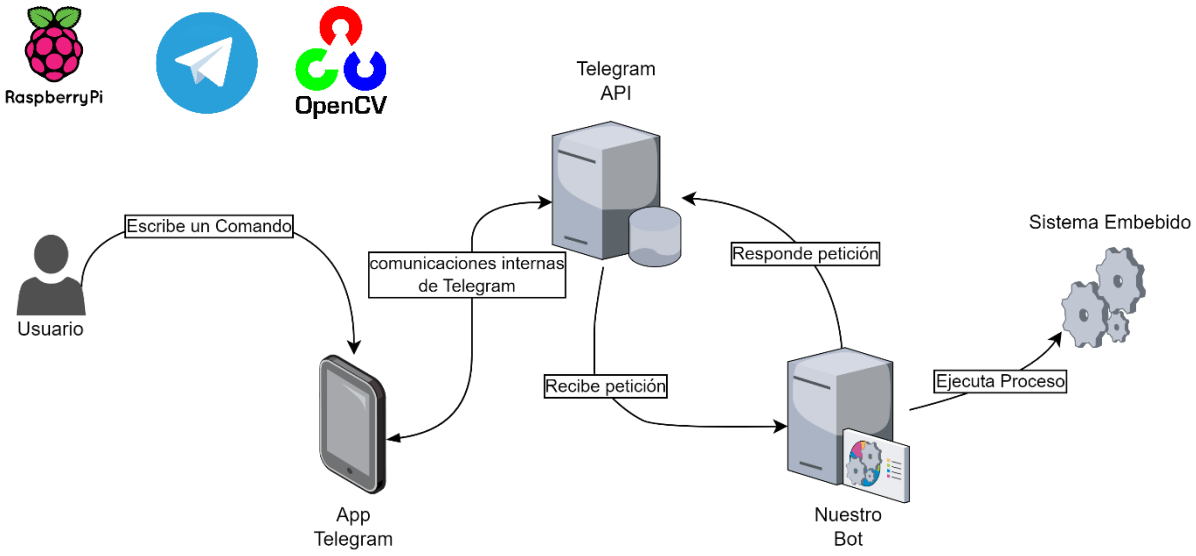



*Figura 1 – Diagrama general de funcionamiento del proyecto*

### 4.1.1 Objetivo general

Como se mencionó anteriormente, se hará uso de una placa de desarrollo llamada Raspberry Pi, es un ordenador de bajo coste y formato compacto destinado al

desarrollado para hacer accesible la informática a todos los usuarios. La Raspberry Pi también se caracteriza por ser muy utilizada para desarrollar pequeños prototipos. Todos los diseños de Raspberry Pi se basan en el hardware libre y habitualmente se utilizan también sistemas operativos libres basados en GNU/Linux, además cuenta con un procesador Cortex-A53.

Se utilizo como App la aplicación gratuita llamada Telegram para su manejo de recepción y envió de mensajes hacia la placa, en donde se va a recibir avisos y manejar los distintos sensores (incluido los mensajes de voz) solamente con enviar unos mensajes que serían nuestros códigos. También cuando alguien toqué el timbre, me va a enviar un Email con una foto que voy a obtener de la cámara y la hora en el cual se tocó el timbre.

Entonces como objetivo general es poder controlar y manejar todos los aparatos de la casa con internet (IoT).

### 4.1.2 Objetivo particular

Como objetivo particular se debería lograr el correcto funcionamiento en cada etapa del proyecto desarrollada, con el fin de cumplir con el objetivo principal del dispositivo en todo su conjunto. Para ello deberemos realizar diversas pruebas en cada etapa y someter al dispositivo a distintas situaciones probables y adversas para obtener un desarrollo completo y confiable.

## 4.2 JUSTIFICACIÓN DEL PROYECTO

### 4.2.1 Antecedentes del proyecto

El origen de la idea del proyecto surgió en pandemia, cuando a la hora de esperar el repartidor o el de correo, no me enteraba de que estaba afuera de mi casa ya que no escuchaba el timbre por estar haciendo otra cosa o escuchando música y eso a veces era malo porque, si era un repartidor de comida le hacía perder el tiempo y en el caso del repartidor del correo, si no le atendía nadie se iba y me dejaba el paquete en el correo y yo después lo tenía que ir a buscar (y ya había pagado el envío a domicilio y por no atenderlo perdí esa plata), entonces la idea era poder recibir lo que sea, sea donde sea que uno este, por eso se plantea este proyecto, y preguntando a gente conocida, a muchos les pasaba lo mismo, entonces me pareció una buena idea para la solución de este problema

### 4.2.2 Estado actual

Si bien existen productos que tienen lo que se va a desarrollar en este proyecto, no hay ni un producto que incluya todas las funcionas descriptas anteriormente y además con la versatilidad de manejar todo con la aplicación de Telegram, lo cual, al mismo tiempo de hacerlo seguro, es gratuito.

### 4.2.3 Necesidad del negocio y definición del problema

Los problemas a resolver son varios, dado que tenemos como un tipo de “timbre inteligente” en nuestra puerta, nos da la posibilidad de ver quien se encuentra afuera (gracias a la cámara), además tener comunicación bidireccional de voz con la persona que se encuentra en ese momento afuera de la casa. Otras de la soluciones es que a la hora de esperar un producto que uno haya comprado por internet, no es necesario quedarse en casa ya que este proyecto nos va a avisar a nuestro Telegram cuando alguien este afuera lo cual como se mencionó anteriormente se podrá responder y una vez que uno está seguro de eso, se le puede abrir una “puerta para paquetes” la cual es activada desde la aplicación de Telegram y el repartidor podrá dejar ahí el paquete y no es necesario que se lleve de vuelta el paquete a la sucursal y tener que uno mismo perder tiempo en ir a buscarlo.

Otra solución que plantea este proyecto es que, gracias al reconocimiento facial biométrico, uno podrá activar esta opción desde la app antes de llegar de la casa y bajarse con las manos llenas (sea por ir a comprar comida o simplemente por ir al supermercado) en donde la cámara estará esperando para reconocer la cara del propietario y abrir la puerta de la casa, esto da comodidad y rapidez.

### 4.2.4 Beneficios del proyecto

Los beneficios que se obtendrán para el sponsor del proyecto es la adquisición de un nuevo producto inteligente y compacto, y su posible comercialización obteniendo beneficios económicos con este por su modelo estructural y hardware de bajo coste, además tendrán los siguientes beneficios:

- Exclusividad A La Hora De Actualizaciones
- Primeros En Probar Las Nuevas Extensiones
- Descuentos Exclusivos

Por otro lado, para los que estén interesados en el proyecto como el público en general, serán beneficiados con un sistema sumamente integrado con todo, incluida la instalación con programación gratuita.

## 4.3 ALCANCE

El prototipo ya desarrollado contará con una central maestra. La función de dicha central es adquirir los datos de sensores, cámara, pulsadores, micrófono, comandos, funcionalidades y transmitirlos inalámbricamente al Bot de Telegram.
Por otra parte, también se encargará de guardar los datos de la cara para su posterior reconocimiento facial, como así también el proceso de enviar audio de voz desde la central al Telegram y viceversa.

### 4.3.1 Alcance

Los requisitos del proyecto son:

- Detección de presencia cuando alguien se acerque al timbre.
- Envió de foto más texto al Telegram al detectar presencia.
- Interacción de Telegram con la placa para poder pedir otra foto y/o prender la luz.
- Detección del pulsador de timbre, enviando foto más texto a Telegram.
- Poder abrir puertas con enviar un comando desde Telegram.
- Detección de pulsador para activar reconocimiento facial biométrico, el cual si da positivo en detección acciona el solenoide para la apertura de la puerta de la casa.
- Enviar comando desde Telegram para activar la detección del reconocimiento facial biométrico y así esperarte para poder la detección y poder abrir la puerta de la casa.
- Detección de pulsador de timbre, enviando también un mail.
- Envió de Audio de voz tanto desde Telegram a la Raspberry como viceversa.

En referencia a lo que nuestro dispositivo final hará y lo que no, se deben aclarar los siguientes alcances y posibilidades brindadas por el mismo:

- La Raspberry al recibir señales de los sensores de entrada (sea el pulsador de timbre o detección de presencia) enviara una foto más un mensaje de aviso al Telegram para poder observar quien se encuentra afuera de la casa.
- Se podrá tener comunicación desde el Bot de Telegram con la Raspberry para manejar la cámara, la apertura de la puerta, luces que se encuentren afuera y demás cosas que se le quieran agregar.
- La Raspberry al recibir la señal del pulsador o el mensaje desde el Bot de Telegram para activar el reconocimiento facial, el cual dará 10 intentos para el reconocimiento positivo o un tiempo de 15 segundos activado a la espera de que se acerque algún rostro. En caso de que diera negativo los dos casos estos se vuelve al estado normal en el sistema y se tiene que volver a activar para poder activar el reconocimiento facial.
- La Raspberry podrá enviar audio de voz para poder tener charla con la persona que se encuentre afuera de la casa.

- No se podrá enviar videostreaming.
- No se podrá ver video y audio en tiempo real.

### 4.3.2 Límites o fuera de alcance

Queda fuera del alcance de este proyecto la interacción de video y audio bidireccional en conjunto en tiempo real, como también el videostreaming.

### 4.3.3 Soluciones y entregables principales

La siguiente tabla muestra un listado de los entregables del proyecto (productos o servicios).

| Entregables principales | Descripción del entregable |
|---|---|
| Central Maestra | Forma parte de la estructura interna del proyecto y es donde se almacenarán los datos del reconocimiento facial y ejecución del código. |
| Email, Telegram y Raspberry | Etapa interna en la central encargada de adquirir datos, comandos, funcionalidades y transmitirlos inalámbricamente a la Raspberry. |
| OpenCV y trasmisión de audio de voz | Etapa interna en la central encargada de adquirir datos de la cara y guardado en la Raspberry, como así también el proceso de enviar audio de voz desde la Raspberry al Telegram y viceversa. |
| Prototipo Final | Se juntará tanto el software como el hardware de los entregables mencionados anteriormente para las pruebas y testeo del producto. |

## 4.4 PLANIFICACIÓN DEL PROYECTO

La estrategia de gestión del proyecto que se seguirá corresponde a la metodología del Project Management Institute (PMI), la cual nos permitirá, mediante constante planificación y control de las etapas dentro del proyecto, alcanzar las metas y objetivos impuestos en cada una de ellas. Para ello debemos:

- Recopilar requisitos: para ello se debe mantener una constante comunicación con los stakeholders para definir aquellos requisitos que se deben cumplir a toda costa, tanto para el producto elaborado como para la gestión y desarrollo del proyecto. Se deben tener en cuenta todos los requisitos, dándoles mayor importancia a aquellos que tengan un nivel de prioridad mayor y, en base a ellos, fijar las especificaciones que deberá tener nuestro producto.

- Definir el alcance de nuestro proyecto, indicando lo que nuestro producto hará y aquello que no, como también los procesos y acciones que se engloban dentro del proyecto y aquellas que quedan fuera del mismo.

- Elaborar un plan de alcance, mediante el cual se definirán las tareas, métodos, herramientas y los pasos a seguir para lograr los requerimientos especificados y los alcances previamente definidos.

- Crear la EDT. Luego de haber definido el alcance y lo que haremos, mediante la estructura de desglose del trabajo dividiremos nuestro proyecto en bloques manejables y les asignaremos recursos, tiempo y costos.

- Se deberán secuenciar aquellas actividades que deban realizarse en un determinado orden y estimar las duraciones de cada una de ellas.

- Elaborar un cronograma que nos permita ordenar y planificar los distintos bloques de trabajo y las actividades a lo largo del tiempo para cumplir con las fechas establecidas.

- También se deberán estimar costos en cada etapa y elaborar presupuestos.

- Se deberá especificar un plan de comunicación, para lograr que los interesados, tanto dentro como fuera del proyecto, se encuentren informados respecto del avance del proyecto.

- Por último, también se deberán identificar los posibles riesgos que pueda presentar nuestro proyecto y elaborar un plan de respuesta para evitarlos o mitigarlos tanto como se pueda.

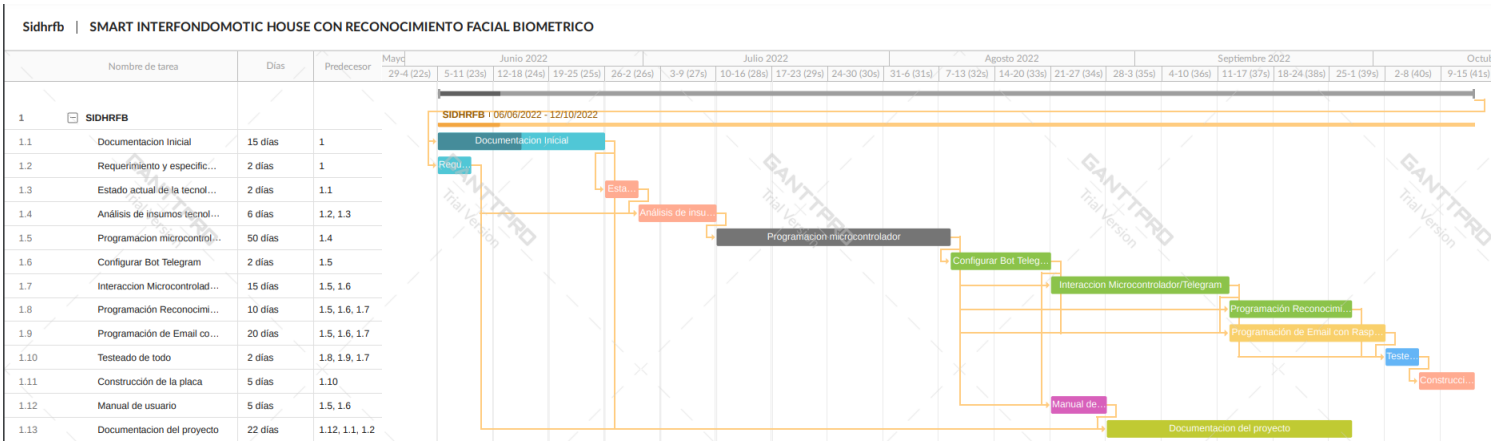
Es importante aclarar que, aunque las distintas fases expuestas para la gestión del proyecto hayan sido presentadas en un determinado orden, no necesariamente se debe

respetarlo, ya que algunas actividades pertenecientes a fases distintas pueden realizarse sin orden premeditado o al mismo tiempo.

### 4.4.1 Cronograma

Previo a armar el diagrama de Gantt del proyecto, organizaremos el proyecto en tareas que deberán cumplirse y las dependencias entre ellas:

| Tarea | Referencia | Predecesora | Duración (horas) |
|---|---|---|---|
| Documentación inicial | 1.1 | 1 | 40 hs |
| Requerimiento y especificaciones | 1.2 | 1 | 20 hs |
| Estado actual de la tecnología | 1.3 | 1.1 | 10 hs |
| Análisis de insumos tecnológicos a usar | 1.4 | 1.2, 1.3 | 30 hs |
| Programación Microcontrolador | 1.5 | 1.4 | 100 hs |
| Configurar Bot Telegram | 1.6 | 1.5 | 100 hs |
| Interacción Microcontrolador/Telegram | 1.7 | 1.5, 1.6 | 50 hs |
| Programación Reconocimiento Facial biométrico con interacción con Telegram | 1.8 | 1.5, 1.6, 1.7 | 100 hs |
| Programación de Email con Raspberry, Telegram | 1.9 | 1.5, 1.6, 1.7 | 100 hs |
| Testeado de todo | 1.10 | 1.7, 1.8, 1.9 | 20 hs |
| Construcción de la placa | 1.11 | 1.10 | 5 hs |
| Manual de usuario | 1.12 | 1.5, 1.6 | 15 hs |
| Documentación del proyecto | 1.13 | 1.1, 1.2, 1.12 | 10 hs |
| TOTAL | | | 600 hs |



### 4.4.2 Hitos

La tabla muestra un listado de hitos generales del proyecto y el cronograma estimado de finalización:

| Hitos | Fecha de finalización |
|---|---|
| $1^{er}$ prototipo funcional microcontrolador/Telegram | Agosto 2022 |
| $2^{do}$ prototipo funcional microcontrolador/Reconocimiento facial biométrico | Noviembre 2022 |
| $3^{er}$ prototipo funcional microcontrolador/Interacción de voz bidireccional en tiempo real | Febrero 2023 |
| Documentación | Marzo 2023 |

## 4.5 RIESGO

**MATRIZ DE RIESGOS**

| RIESGO | Aparición probabilidad | Gravedad (Impacto) | Valor del Riesgo | Nivel de Riesgo |
|---|---|---|---|---|
| Falta de insumos | 4 | 4 | 16 | Muy grave |
| Que la aplicación Telegram deje de tener la opcion de crear un bot o que deje de ser gratuita | 2 | 5 | 10 | Importante |
| No cumplir con las fechas de entrega | 2 | 2 | 4 | Apreciable |
| Que el prototipo funcione "mal" y no responda a las necesidades reales que se propusieron | 3 | 4 | 12 | Importante |
| Exceder el presupuesto | 1 | 3 | 3 | Apreciable |
| Quemar algún componente o dispositivo | 3 | 3 | 9 | Importante |
| Que no alcance la memoria de la Raspberry Pi para hacer correr todo el codigo | 3 | 3 | 9 | Importante |

- **Falta de insumos:** En el caso de que falte insumos lo que se hace es buscar dos componentes que cumplan con los requerimientos y cuyos precios sean similares, aquel de mayor accesibilidad será el elegido. Con esto conseguimos mitigar o directamente evitar el retraso producido por falta de insumos.

- **Que la aplicación Telegram deje de tener la opción de crear un bot o que deje de ser gratuito:** Análogamente como en el caso anterior se debería optar por crear una aplicación propia.

- **No cumplir con las fechas de entrega:** En este caso, para mitigar su efecto se debe planear los tiempos de cada etapa con tiempo de resguardo. Este tiempo de holgura

puede usarse para cubrir estas eventualidades o parte de ellas y que el tiempo total del proyecto no se estire desmedidamente.

- **Que el prototipo funcione “mal” y no responda a las necesidades reales que se propusieron:** Para evitar este “mal” funcionamiento, la manera de mitigar esto es haciendo pruebas a cada momento que se avanza en cada etapa del proyecto y en caso de que sea necesario ir mejorando el código lo que más se pueda para la optimización de los recursos de la placa de desarrollo para que pueda funcionar de la manera más optima.

- **Exceder el presupuesto:** En este caso se debe poseer un cierto margen de resguardo, poseer un fondo económico de resguardo del que sacar dinero en caso de requerir componentes o dispositivos nuevos, o ante el cese o desperfecto de los ya existentes**.**

- **Quemar algún componente o dispositivo:** Para evitar que esto ocurra se debe tener especial cuidado en los niveles de tensión y corriente que se les apliquen a las placas y a los componentes. Para eso sirve el estudio de las tecnologías que se realiza previo al diseño y desarrollo propiamente del dispositivo.

- **Que no alcance la memoria de la Raspberry Pi para hacer correr todo el código:** Para evitar esto se va a optimizar al máximo los recursos gráficos para que pueda correr el programa sin ningún problema, en el caso extremo que no llegase a ser así, se optara por ir a una Raspberry con más memoria RAM.

# 5 DESARROLLO DEL PROYECTO

## 5.1 DESARROLLO TÉCNICO

### 5.1.1 Introducción

El Internet de las cosas (IoT) es un tema importante en los círculos de la industria tecnológica, las políticas y la ingeniería y se ha convertido en titulares tanto en la prensa especializada como en los medios populares. Esta tecnología está incorporada en un amplio espectro de productos, sistemas y sensores en red, que aprovechan los avances en la potencia informática, la miniaturización de la electrónica y las interconexiones de redes para ofrecer nuevas capacidades que antes no eran posibles. Una gran cantidad de conferencias, informes y artículos de noticias discuten y debaten el posible impacto de la “revolución de IoT”, desde nuevas oportunidades de mercado y modelos de negocio hasta preocupaciones sobre seguridad, privacidad e interoperabilidad técnica.

Este proyecto se centra en explotar IoT para garantizar una mejor Sistema de seguridad y automatización de ciertos artefactos en los hogares. Los sistemas de timbre actuales siguen un enfoque tradicional, cuando un visitante presiona el interruptor que suena dentro de la casa. Si hay alguien presente en la casa, abren la puerta y si no hay nadie presente en la casa, el visitante espera un tiempo determinado y abandona el lugar sin ninguna respuesta. En los últimos años la tecnología se ha apoderado de la sociedad. La tecnología es vital hoy en día y lo hace todo más fácil. Uno de esos avances en el campo del timbre es el uso del "sistema de timbre automático" (ADBS). Los timbres de puerta han pasado de interruptores históricos a paneles táctiles modernos y ahora son más sofisticados con el uso de sensores e IOT.

Por otra parte, este sistema aporta un sistema de seguridad ya que te permite hablar con el visitante que se encuentra en la puerta de entrada a través del móvil celular, evitando la comunicación cara a cara, esto tiene una gran importancia para la gente mayor que viven solos.

Por lo tanto, para que este sistema sea eficiente con el desarrollo de la tecnología implementada y tenga un sistema sólido y confiable para sus tareas complejas, se utiliza una Raspberry Pi en conjunto con la aplicación de uso gratuito Telegram.

Si bien la solución propuesta contemplada en el Proyecto Final consiste en realizar solamente un prototipo. Esto no implica que, posteriormente y quedando fuera del proyecto, el trabajo no pueda ser mejorado para que sea producido en serie y se incluya en el mercado actual.

### 5.1.2 Hardware Requerido

Para la construcción de este prototipo se necesitó de varios componentes, todos estos componentes serán descritos en el punto siguiente.

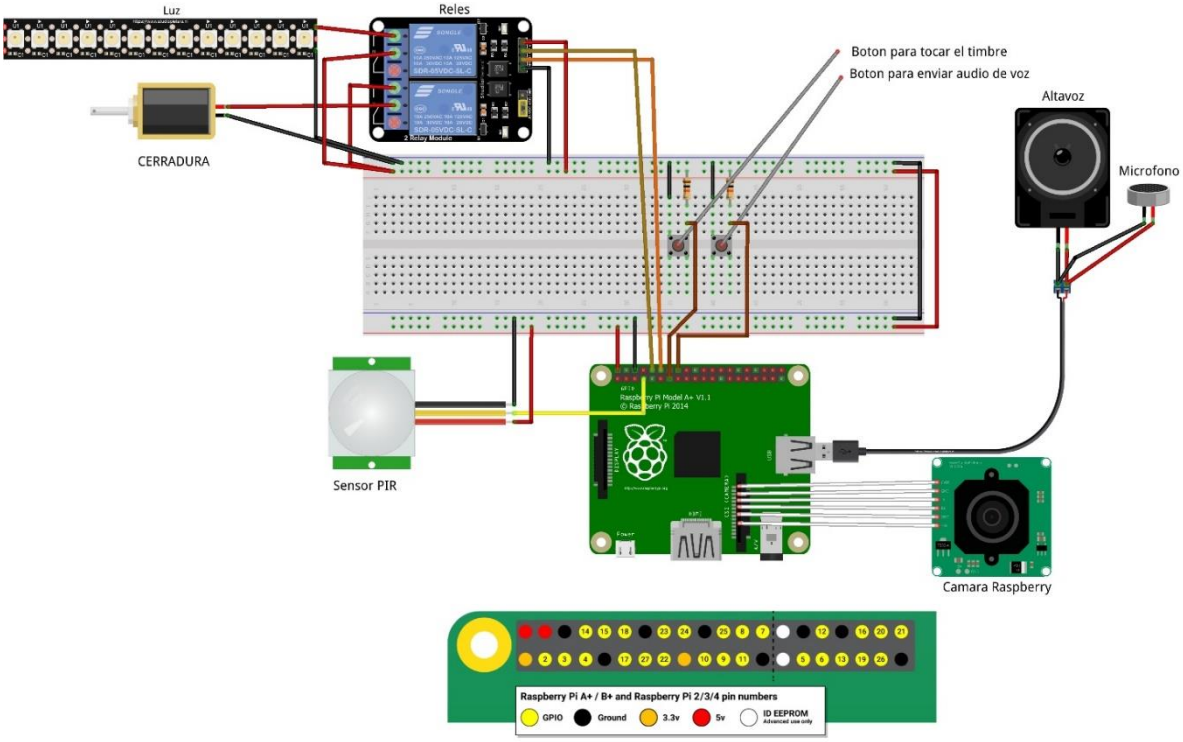



*Figura 2 - Conexionado del hardware en la Raspberry Pi*

#### 5.1.2.1 Raspberry Pi 3 A+

Raspberry Pi 3 Model A+ es el último producto de la gama Raspberry Pi 3, cuenta con un procesador de cuatro núcleos de 64 bits que funciona a 1,4 GHz, LAN inalámbrica de doble banda de 2,4 GHz y 5 GHz y Bluetooth 4.2/BLE. La LAN inalámbrica de doble banda viene con certificación de cumplimiento modular, lo que permite diseñar la placa en productos finales con pruebas de cumplimiento de LAN inalámbrica significativamente reducidas, lo que mejora tanto el costo como el tiempo de comercialización. La Raspberry Pi 3 Modelo A+ tiene un tamaño de 65mm x 56mm.

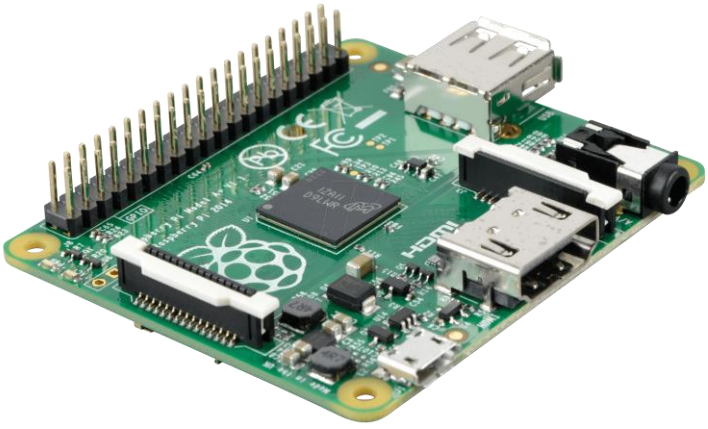


*Figura 3 – Raspberry Pi 3A+*

- **Procesador:** Broadcom BCM2837B0, Cortex-A53 64-bit SoC @ 1.4 GHz
- **Memoria:** 512MB LPDDR2 SDRAM

- **Conectividad:** 2.4 GHz and 5 GHz IEE 802.11.b/g/n/ac wireless LAN, Bluetooth 4.2/BLE 1 × USB 2.0 port
- **Acceso:** Encabezado GPIO de 40 pines extendido
- **Video y Sonido:** 1 × HDMI de tamaño completo, Puerto de pantalla MIPI DSI, Puerto de cámara MIPI CSI, Salida estéreo de 4 polos y puerto de video compuesto.
- **Multimedia:** H.264, decodificación MPEG-4 (1080p30); codificación H.264 (1080p30); Gráficos OpenGL ES 1.1, 2.0
- **Soporte para tarjetas SD:** Formato Micro SD para cargar sistema operativo y almacenamiento de datos
- **Potencia de entrada:** 5 V/2.5 A CC mediante conector micro USB, 5 V CC a través del cabezal GPIO.
- **Ambiente:** Temperatura de funcionamiento, 0–50 °C
- **Cumplimiento**: Para obtener una lista completa de las aprobaciones de productos locales y regionales, visite: www.raspberrypi.org/products/raspberry-pi-3-model-a-plus

#### 5.1.2.2 Tarjeta MicroSD

La Raspberry Pi es una computadora y, como cualquier otra computadora, necesita un sistema operativo (SO) instalado. El Pi no tiene almacenamiento incorporado, por lo que necesitará una tarjeta microSD para instalar su sistema operativo. Instalaremos el sistema operativo en la tarjeta microSD. Se recomienda usar una tarjeta microSD clase 10 con ***al menos*** 16 GB de memoria.


*Figura 4 – Tarjeta MicroSD para Raspberry PI 3A+*

#### 5.1.2.3 Sensor PIR

Los sensores infrarrojos pasivos (PIR) son dispositivos para la detección de movimiento. Son baratos, pequeños, de baja potencia, y fáciles de usar.

*Figura 5 – Sensor PIR*

#### 5.1.2.4 Cámara V2 para Raspberry Pi

El módulo de cámara v2 tiene un sensor Sony IMX219 de 8 megapíxeles. El módulo de cámara se puede utilizar para tomar videos de alta definición, así como fotografías. Es fácil de usar para principiantes, pero tiene mucho que ofrecer a los usuarios avanzados si está buscando expandir su conocimiento. Hay muchos ejemplos en línea de personas que lo usan para "hiperlapse", cámara lenta y otras aplicaciones de inteligencia artificial.

Sera de gran uso para este proyecto para la toma de fotografías, grabación de video y el reconocimiento facial.


*Figura 6 – Cámara V2 Raspberry Pi*

#### 5.1.2.5 Altavoz y micrófono

Se utiliza un altavoz para poder ser escuchado los audios que se envían desde Telegram al sistema embebido y un micrófono para que el visitante pueda hablar por ahí y enviar audios a Telegram y ser escuchado por el propietario.

*Figura 7 – Altavoz de 8 Ohm y micrófono*

#### 5.1.2.6 Mini placa amplificadora digital

Se puso un Amplificador Audio Super Mini Digital 1x 5w 2.5v A 5.5v modelo Xs9871 para que se escuche con más potencia la salida el audio en el parlante.


*Figura 8 – Amplificador Audio Super Mini Digital*

#### 5.1.2.7 Cerradura Solenoide

La cerradura será otro de los actuadores controlables por el usuario para abrir o cerrar la puerta de la vivienda. La cerradura simple de solenoide es la opción más viable en el caso del presente, ya que las otras alternativas constan de software propio, que sólo complicaría el desarrollo del prototipo.

*Figura 9 – Cerradura Solenoide*

#### 5.1.2.8 Modulo Relé de 2 Vías

El relé es un dispositivo que es utilizado para controlar el encendido y apagado de dispositivos que requieren altos voltajes o altas corrientes mediante una señal de corriente continua como las proporcionadas por los pines de una Raspberry Pi.

Para este proyecto se lo utiliza para encender y apagar la Luz y el Solenoide.


*Figura 10 – Modulo Relé*

### 5.1.3 Comunicaciones

Para la transmisión de datos en el sistema embebido, las conexiones entre la Raspberry Pi, los sensores y actuadores son cableadas siguiendo las características propias de cada dispositivo.

La conexión de la Raspberry Pi con el Bot de Telegram se produce a través de Internet; es inalámbrica mediante WiFi para conectar la Raspberry Pi y desde Internet a Telegram depende de la conexión del dispositivo en el que se esté ejecutando Telegram, ya que existe una versión de escritorio y una aplicación para dispositivos móviles en diferentes sistemas operativos.

Para esta conexión hay varios protocolos que se podrían usar, sin embargo, lo que se utilizó en este proyecto es HTTP (Hypertext Transfer Protocol); Es un protocolo con un esquema cliente/servidor, efectivo a la hora de enviar grandes cantidades de información en un ecosistema IoT. Esto lo hace muy recomendable cuando el nodo o Gateway ha de enviar imágenes, vídeos o archivos.

A continuación, se muestra un esquema para ilustrar las comunicaciones del sistema:

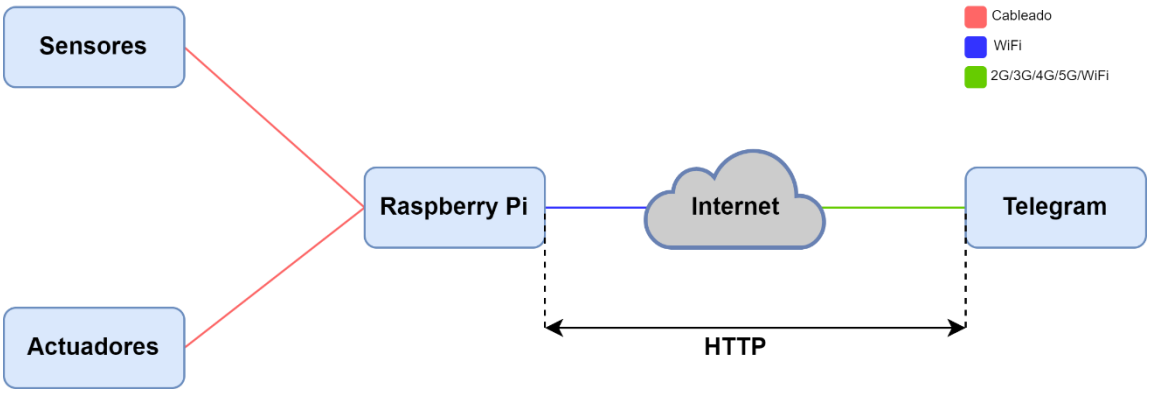



*Figura 11 – Esquema de las comunicaciones del sistema completo*

### 5.1.4 Desarrollo del Proyecto

En primer lugar, se ha de preparar la Raspberry Pi como se indica en el Anexo 1: Guía de instalación.

Una vez el sistema operativo y las librerías necesarias para el funcionamiento del código están instaladas, se puede iniciar la Raspberry Pi. La Raspberry Pi se ha preparado para que, al iniciarse, se ponga en funcionamiento el sistema. Al ponerse en funcionamiento, se inicializan los componentes asociados a las diferentes funcionalidades del sistema. Estas funcionalidades son:

- Control del sistema mediante el bot de Telegram
- Apertura y cierre de la puerta
- Encendido de Luz
- Detección de movimiento mediante sensores PIR
- Reconocimiento facial
- Transmisión en directo mediante nota de voz con Telegram
- Cámara activa para realización de fotos y videos
- Registro de los eventos que suceden mientras el sistema de seguridad está activo (Presencia entrada o alguien pulsa el botón de timbre)

A continuación, se explicarán los componentes del sistema individualmente, acompañados de su implementación en el código y su funcionamiento.

### 5.1.5 Bot de Telegram

El Bot de Telegram es una herramienta útil y accesible que Telegram proporciona. Para la creación del Bot de Telegram, en primer lugar, se tendrá que usar el Bot ***BotFather***. La utilidad de este Bot es la de crear bots nuevos cuando el usuario lo solicita. En la Figura 1, se puede ver los comandos usados para crear el bot SIHRFB (Nombre del Proyecto Final).

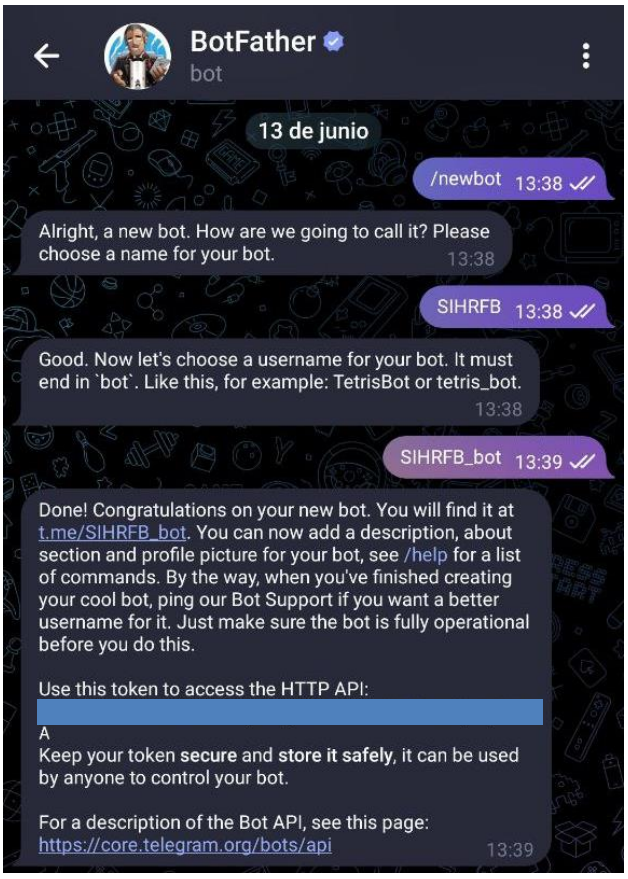



*Figura 12 – Conversación Bot de Telegram*

Una vez creado el bot que se va a usar para nuestro sistema, *BotFather* devuelve un token de acceso. Este token servirá para controlar el bot creado y habrá que implementarlo en el código del mismo. Es importante anotar que el token es privado, ya que, de saberlo, cualquier persona podría tener acceso a él.

Una parte fundamental del bot de Telegram son los comandos. Los comandos permiten al usuario interactuar con el bot y realizar las diferentes funciones disponibles. Por otra parte, desde el *BotFather* se puede agregar foto de perfil, descripción, menú de comandos, etc. a nuestro bot quedando de la siguiente manera:

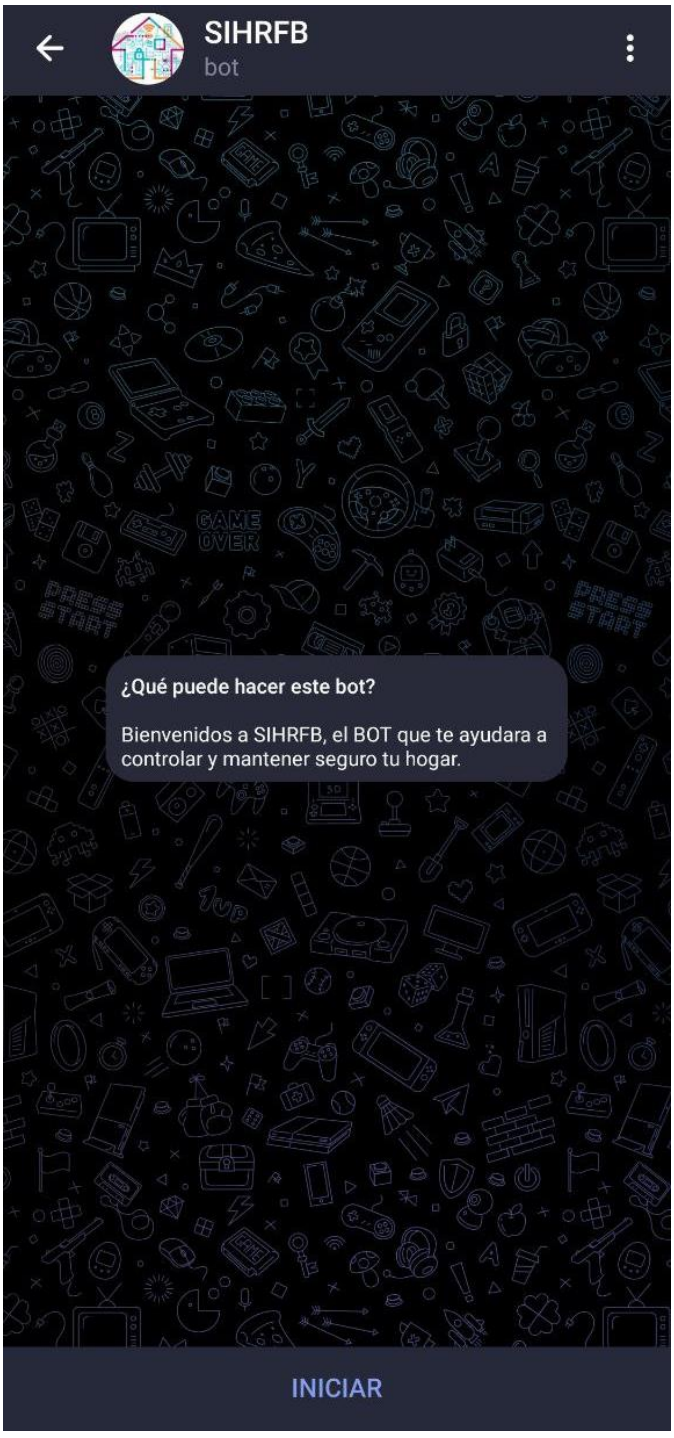


*Figura 13 – Bot SIHRFB configurado*

Para acceder al bot desde un dispositivo móvil con cámara se ha generado un código QR que facilita el acceso:


*Figura 14 – Código QR para acceder al Bot de Telegram*

### 5.1.6 Archivo main.py

Como primer paso lo que se hizo es crear un archivo main.py, en el cual lo que hago es crear un subproceso para ejecutar mi archivo todo.py (archivo donde se encuentra básicamente el control completo del Bot, se explicara más adelante), al ejecutar este archivo lo que hago es registrar su PID y guardarlo en un archivo txt, además, creo un bucle while infinito a la espera de un evento externo, en este caso, al tocar el botón del timbre o al tocar el botón para grabar audio.

Se puede ver en la figura 6, en la línea 15 que se hace una espera de 10 segundos, esto es para que le dé tiempo a ejecutar el proceso todo.py y guardar su PID. Una vez hecho esto, desde mi archivo main.py, lo que hago es abrir ese archivo .txt y leer el PID guardado para después poder mandarles señales.

En la figura 5, se presenta un diagrama de flujo del archivo main.py:

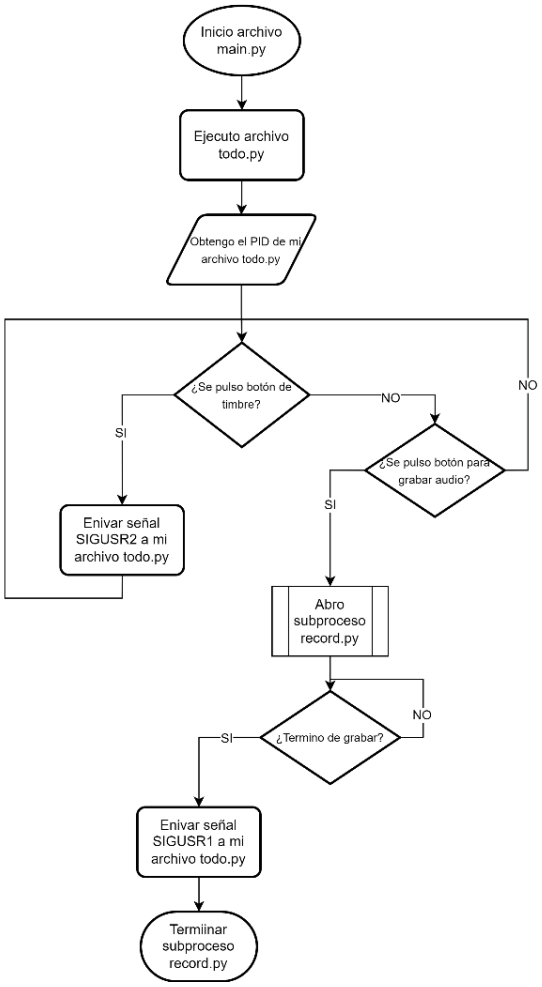



*Figura 15 – Diagrama de flujo archivo main.py*

```python
import multiprocessing as mp
import signal
import os
from time import sleep
import subprocess as sp
from record import grabar
from gpiozero import Button

btn = Button(23)
btn2 = Button(27)

def main():
    proc=sp.Popen(["python","/home/david/facial/todo.py", "2>/dev/null"]) #Abro primero el archivo de Telegram asi tengo su PID.
    grabando = False
    sleep(10) #Espero 10 segundos asi se carga el archivo de telegram y guarda el PID.
    f = open("pid.txt","r")
    pid1 = f.read()
    f.close()
    print(pid1)

    while True:
        #--------------------Boton para grabar audio-------------------------
        if btn.value:
            grabacion = mp.Process(target= grabar, args=(int(pid1), btn))
            grabacion.start()
            grabando = True
        if grabando:
            grabacion.join()
            grabando = False
        #--------------------Boton del timbre----------------------
        if btn2.is_pressed:
            os.kill(proc.pid, signal.SIGUSR2)
            sleep(2)

if __name__ == '__main__':
    main()

```

*Figura 16 - Código archivo main.py*

En la figura 6, se puede observar que se ejecuta un while infinito a la espera de un evento externo. Se observa que en la línea 23 si se presiona el primer botón que es de grabar audio (**btn**), se inicia un proceso **mp.Process** que llama a la función **grabar** de mi archivo record.py, pasando como argumentos el PID del proceso de Telegram y el botón **btn**, esto se muestra en la figura 7:

```
1  import os
2  import signal
3  import wave
4  import pyaudio
5  from gpiozero import Button
6
7  def grabar(pid_bot, btn):
8          CHUNK = 1024
9          FORMAT = pyaudio.paInt16
10         CHANNELS = 2
11         RATE = 44100
12         WAVE_OUTPUT_FILENAME = "voice.wav"
13
14         p = pyaudio.PyAudio()
15         #Abro la entrada de objetos para el stream
16         stream = p.open(format=FORMAT,
17                 channels=CHANNELS,
18                 rate=RATE,
19                 input=True,
20                 frames_per_buffer=CHUNK)
21         print ("* grabando")
22         frames = []
23
24         while btn.value:
25             data = stream.read(CHUNK)
26             frames.append(data)
27
28         print("* grabacion completa")
29         stream.stop_stream()
30         stream.close
31         p.terminate()
32         wf = wave.open(WAVE_OUTPUT_FILENAME, 'wb')
33         wf.setnchannels(CHANNELS)
34         wf.setsampwidth(p.get_sample_size(FORMAT))
35         wf.setframerate(RATE)
36         wf.writeframes(b''.join(frames))
37         wf.close()
38         os.kill(pid_bot, signal.SIGUSR1)
```

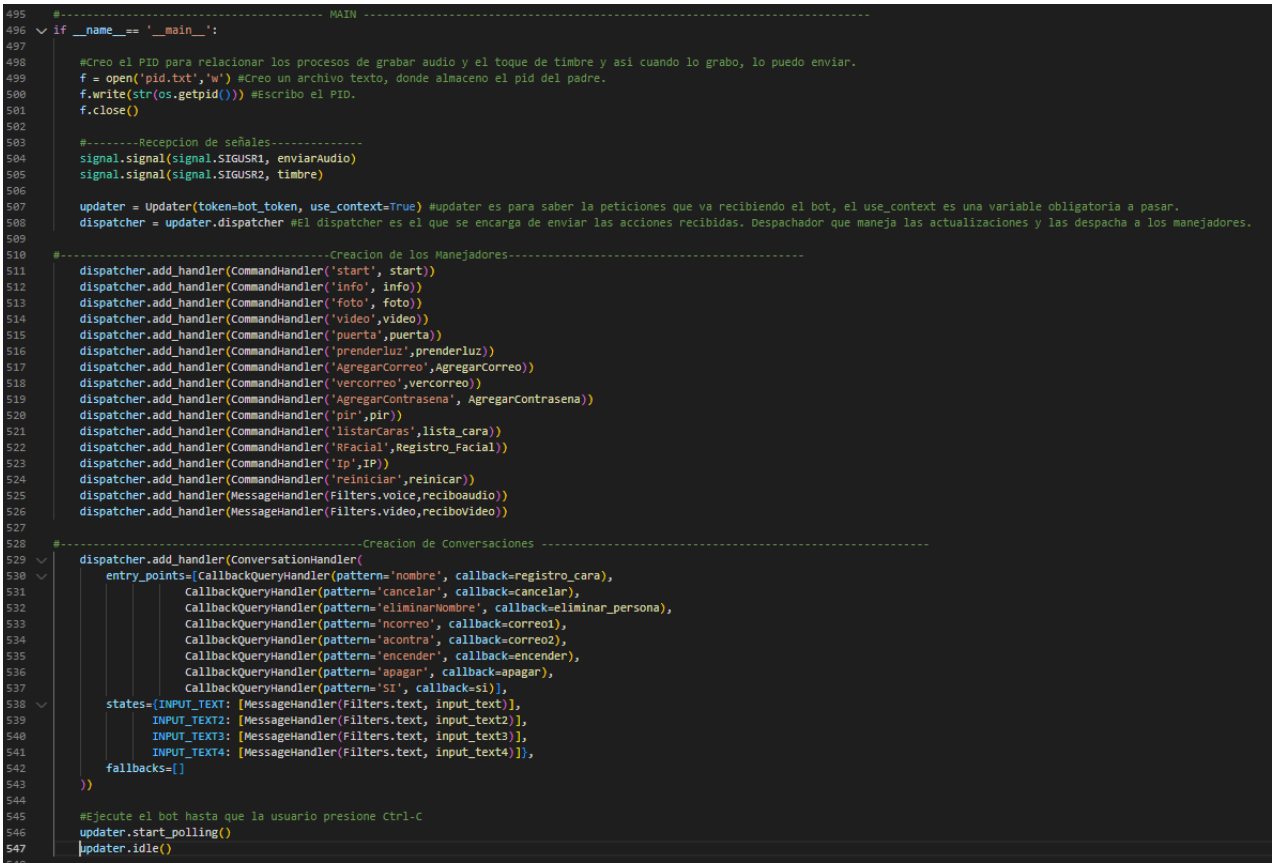
*Figura 17 - archivo record.py (grabar audio)*

En este archivo lo que hago es abrir un stream de entrada para el micrófono y comenzar a grabar el audio en pequeños fragmentos (chunks) que se almacenan en una lista llamada **frames**. La grabación continuará mientras el botón **btn** esté presionado. Cuando se suelta el botón, la grabación se detiene y los datos de audio almacenados en **frames** se escriben en un archivo WAV en el disco. Finalmente, la función envía una señal SIGUSR1 al PID de todo.py para avisarle que hay un audio que se grabó (más adelante se explica que hace la función todo.py al recibir la señales).

En el caso de que se presione el botón para tocar el timbre, lo que hago es enviar envió la señal SIGUSR2 al PID de todo.py para indicar que alguien está tocando el timbre. La espera de 2 segundos es simplemente por si se presiona el botón dos o más veces seguidas.

### 5.1.7 Archivo todo.py

Ahora se mostrará el archivo todo.py, como para seguir lo anterior se muestra lo que hace el __main__ en mi programa, esto se puede ver en la figura 18:



```
#------------------------------------ MAIN ----------------------------------------------------------------------
if __name__== '__main__':

    #Creo el PID para relacionar los procesos de grabar audio y el toque de timbre y asi cuando lo grabo, lo puedo enviar.
    f = open('pid.txt','w') #Creo un archivo texto, donde almaceno el pid del padre.
    f.write(str(os.getpid())) #Escribo el PID.
    f.close()

    #--------Recepcion de señales--------------
    signal.signal(signal.SIGUSR1, enviarAudio)
    signal.signal(signal.SIGUSR2, timbre)

    updater = Updater(token=bot_token, use_context=True) #updater es para saber la peticiones que va recibiendo el bot, el use_context es una variable obligatoria a pasar.
    dispatcher = updater.dispatcher #El dispatcher es el que se encarga de enviar las acciones recibidas. Despachador que maneja las actualizaciones y las despacha a los manejadores.

#---------------------------------------Creacion de los Manejadores-----------------------------------------
    dispatcher.add_handler(CommandHandler('start', start))
    dispatcher.add_handler(CommandHandler('info', info))
    dispatcher.add_handler(CommandHandler('foto', foto))
    dispatcher.add_handler(CommandHandler('video',video))
    dispatcher.add_handler(CommandHandler('puerta',puerta))
    dispatcher.add_handler(CommandHandler('prenderluz',prenderluz))
    dispatcher.add_handler(CommandHandler('AgregarCorreo',AgregarCorreo))
    dispatcher.add_handler(CommandHandler('vercorreo',vercorreo))
    dispatcher.add_handler(CommandHandler('AgregarContrasena', AgregarContrasena))
    dispatcher.add_handler(CommandHandler('pir',pir))
    dispatcher.add_handler(CommandHandler('listarCaras',lista_cara))
    dispatcher.add_handler(CommandHandler('RFacial',Registro_Facial))
    dispatcher.add_handler(CommandHandler('Ip',IP))
    dispatcher.add_handler(CommandHandler('reiniciar',reinicar))
    dispatcher.add_handler(MessageHandler(Filters.voice,reciboaudio))
    dispatcher.add_handler(MessageHandler(Filters.video,reciboVideo))

#-----------------------------------------Creacion de Conversaciones ------------------------------------------------
    dispatcher.add_handler(ConversationHandler(
        entry_points=[CallbackQueryHandler(pattern='nombre', callback=registro_cara),
                      CallbackQueryHandler(pattern='cancelar', callback=cancelar),
                      CallbackQueryHandler(pattern='eliminarNombre', callback=eliminar_persona),
                      CallbackQueryHandler(pattern='ncorreo', callback=correo1),
                      CallbackQueryHandler(pattern='acontra', callback=correo2),
                      CallbackQueryHandler(pattern='encender', callback=encender),
                      CallbackQueryHandler(pattern='apagar', callback=apagar),
                      CallbackQueryHandler(pattern='SI', callback=si)],
        states={INPUT_TEXT: [MessageHandler(Filters.text, input_text)],
                INPUT_TEXT2: [MessageHandler(Filters.text, input_text2)],
                INPUT_TEXT3: [MessageHandler(Filters.text, input_text3)],
                INPUT_TEXT4: [MessageHandler(Filters.text, input_text4)]},
        fallbacks=[]
    ))

    #Ejecute el bot hasta que la usuario presione Ctrl-C
    updater.start_polling()
    updater.idle()
```

*Figura 18 – MAIN de mi programa todo.py*

Mi __main__ es el primer módulo de Python especificado por el usuario que empieza a ejecutarse. Es “de máximo nivel” porque importa todos los demás módulos que necesita el programa.

En la línea 499 tenemos **f = open('pid.txt','w'):** Esta línea crea un archivo llamado 'pid.txt' en modo de escritura ('w') para almacenar el PID del proceso padre.

En la línea 500 tenemos **f.write(str(os.getpid())):** Esta línea escribe el PID del proceso padre en el archivo 'pid.txt'. Como se mencionó en la sección anterior, esto me va a servir para que en mi archivo main.py pueda abrir este archivo 'pid.txt' y leer el PID del programa todo.py.

En la línea 504 y 505 tenemos a **signal.signal(signal.SIGUSR1, enviarAudio) y signal.signal(signal.SIGUSR2, timbre)** : Estas dos líneas establecen la señal de llamada **SIGUSR1** y **SIGUSR2** respectivamente para llamar a las funciones **enviarAudio()** y **timbre()** cuando se recibe la señal (Se explicara más adelante que hace cada función).

**updater = Updater(token=bot_token, use_context=True)**: Esta línea crea un objeto **Updater** que se utiliza para gestionar las actualizaciones de un bot de Telegram, que se autentica con un token proporcionado a través de la variable **bot_token**, **este token se obtiene cuando se crea el bot, es privado y no se debe compartir con nadie.**

**dispatcher = updater.dispatcher** : Esta línea crea un objeto **dispatcher** que es responsable de despachar los comandos recibidos al bot.

**dispatcher.add_handler()**: Esta sección agrega diferentes manejadores para los diferentes comandos que el bot puede recibir.

**dispatcher.add_handler(ConversationHandler())**: Esta sección agrega manejadores de conversación que son responsables de manejar las interacciones de varios pasos con el usuario, que se activan al recibir ciertos comandos.

**updater.start_polling()**: La función start_polling() inicia la escucha de nuevas actualizaciones de Telegram y las envía al despachador (dispatcher) del bot. Este método utiliza la API de Telegram para conectarse al servidor y recibir las actualizaciones de los usuarios.

**updater.idle():** la función idle() mantiene la ejecución del bot en espera indefinidamente, mientras sigue escuchando y procesando las actualizaciones que llegan de Telegram. En otras palabras, esta función bloquea la ejecución del bot hasta que se interrumpe manualmente la ejecución del programa.

Es importante tener en cuenta que, aunque el bot esté en espera, siempre está disponible para procesar y responder a las solicitudes de los usuarios.

### 5.1.8 Comienzo para usar el Bot, Función Start e info

Cuando se abre el Bot en Telegram, se puede ver como en la figura 19, y al apretar INICIAR, automáticamente se envía al chat del Bot el comando **/start**, como se ve en la siguiente figura 20:


*Figura 19 - Primera vista*

**Cuando se aprete INICIAR**

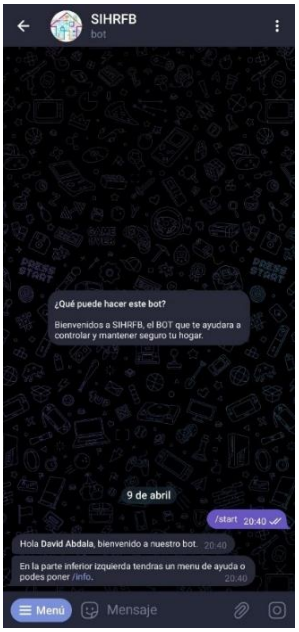



*Figura 20 - comando /start*

Ahora se explica lo que se ve en la figura 21, cuando se envía el comando /start a mi Bot de Telegram, la función **start()** (línea 26) es un manejador que se encarga de responder al comando /start.

```
25  #---------------------------------------Funcion /start del BOT, lo primero que se ejecuta al inicar el bot---------------------------------
26  def start(update, context):
27      global chat_id
28      chat_id = update.message.chat_id
29      first_name = update.message.from_user.first_name
30      last_name = update.message.from_user.last_name
31      msg = f'Hola *{first_name} {last_name}*, bienvenido a nuestro bot. '
32
33      context.bot.sendMessage(chat_id=chat_id, text=msg, parse_mode = "Markdown")
34      context.bot.sendMessage(chat_id, 'En la parte inferior izquierda tendras un menu de ayuda o podes poner /info.')
35
36  #---------------------------------------Funcion /info del BOT, informacion extra y para saber que hace cada comando---------------------------
37  def info(update, context):
38      chat_id = update.message.chat_id
39      msg = "/start: Sirve para inicializar el BOT (una vez o cada vez que se reinicie el sistema).\n\n"
40      msg+= "/foto: Saca una foto en tiempo real desde la camara y lo sube al BOT.\n\n"
41      msg+= "/video: Graba un video de 5 segundos en tiempo real y lo sube al BOT.\n\n"
42      msg+= "/puerta: Abrir puerta, 5 segundos de activacion.\n\n"
43      msg+= "/prenderluz: Activa el rele para prender la luz.\n\n"
44      msg+= "/pir: Activa el rele para prender la luz.\n\n"
45      msg+= "/Ip: Ver direccion IP de la Raspberry PI.\n\n"
46      msg+= "/listarCaras: Listar caras registradas para el Reconocimiento Facial.\n\n"
47      msg+= "/RFacial: Activar Camara para el ingreso por Registro Facial.\n\n"
48      msg+= "/AgregarCorreo: Agregar correo.\n\n"
49      msg+= "/AgregarContrasena: Agregar contraseña de correo.\n\n"
50      msg+= "/verCorreo: ver correo y contraseña guardada.\n\n"
51      msg+= "/reiniciar: Reinicia el sistema, enviar nuevamente /start despues de que se reinicie.\n\n"
52      msg+= "Cargar cara para Registro Facial:\n<i>1)</i> Grabe un video de por lo menos 10 segundos y espere que se guarde.\n\n"
53      msg+= "<i>2)</i> Al abrir la botonera ingresar el Nombre a registrar o puede cancelar la operacion.\n\n"
54      msg+= "<i>3)</i> Una vez registrado el nombre, espere mientras SIHRFB registra la cara, le debe llegar un mensaje de 'Proceso Terminado'.\n"
55
56      context.bot.sendMessage(chat_id=chat_id, text=msg, parse_mode="html")
```

*Figura 21 – funciones start e info*

Esta función toma dos argumentos, el objeto update y el objeto context, que son proporcionados por el dispatcher del bot. El objeto **update** contiene información sobre el mensaje de Telegram que activó esta función, mientras que el objeto **context** proporciona acceso a las funcionalidades del bot.

Dentro de la función, se utiliza la variable global chat_id para almacenar el **ID** del chat de Telegram en el que se recibió el mensaje (El usuario que envió el comando). Luego, se extraen los nombres del usuario que envió el mensaje, utilizando los atributos first_name y last_name del objeto User en update.message.from_user. Después de eso, se construye un mensaje de bienvenida personalizado para el usuario, que incluye su nombre y apellido, utilizando un formato de texto enriquecido (Markdown) que permite aplicar estilos y formatos al texto.

Finalmente, se envían dos mensajes de texto al usuario utilizando el método **sendMessage()** del objeto bot, que se encuentra en el objeto context. El primer mensaje envía el mensaje personalizado de bienvenida, y el segundo mensaje indica al usuario cómo obtener ayuda en el bot, proporcionando opciones para acceder al menú de ayuda con el comando /info.

Cuando el Usuario envía el comando /info, se ejecuta la función **info()** (línea 37), al recibir el comando se despliega una lista de comandos disponibles que el bot puede procesar, junto con una breve descripción de cada uno de ellos (figura 23).

Por otra parte, en el mismo bot en la esquina inferior izquierda hay un botón de menú, el cual te muestra las opciones para poner automáticamente /start o /info, este se puede ver en la figura 22:


*Figura 22 - Botón Menú*

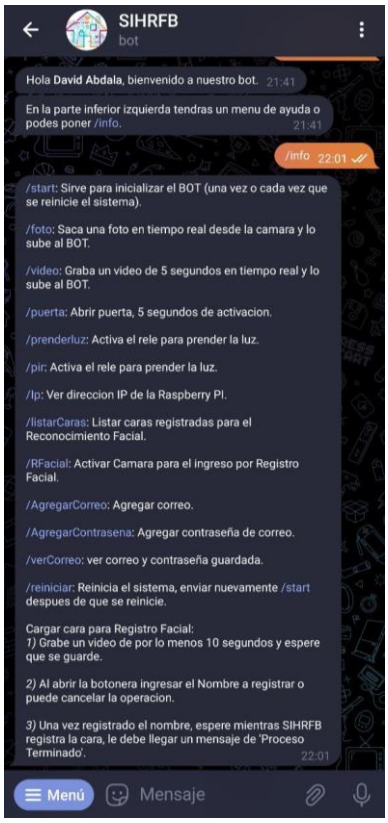



*Figura 23 – Comando /info*

### 5.1.9 Función para enviar audio a Telegram

Al recibir la señal de llamada **SIGUSR1**, lo que hace es llamar a la función **enviarAudio()**, explicándose a continuación:

```
112  #---------------------------------------Funcion para enviar una nota de voz al BOT-----------------------------------------
113  def enviarAudio(nro, framw):
114      global chat_id
115      dispatcher.bot.sendChatAction(chat_id, 'record_audio') #al usar el dispatcher, no es necesario usar context. Avisa que esta grabando un Audio.
116      command = 'sox voice.wav voice.ogg' #Comando para transformar de wav a ogg, usando sox.
117      call([command], shell=True)
118      dispatcher.bot.sendVoice(chat_id, voice = open('voice.ogg', 'rb'))
119
```

*Figura 24 – Función enviarAudio*

La función comienza enviando un mensaje de que se está grabando un audio (figura 25) y luego llama a un comando de terminal que utiliza el programa **"sox"** para convertir el archivo de audio de formato WAV a formato OGG (Esto es porque el Bot para enviarlo como nota de voz es necesario que sea un archivo .ogg). Luego, utiliza la función "sendVoice" del bot de Telegram para enviar el archivo de audio convertido al chat_id correspondiente.


*Figura 25– Mensaje de aviso que se esta grabando un mensaje de voz*

### 5.1.10 Función para recibir un audio desde Telegram

Esta función me sirve para controlar las notas de voz enviada por el Usuario al Bot de Telegram.

```
#-------------------------------------Funcion para reproducir una nota de voz recibida desde el BOT-----------------------------------------
def reciboaudio(update, context):
    file = context.bot.getFile(update.message.voice.file_id) #Al recibir una nota de voz, la descargo y la guardo.
    file.download('voice.m4a')
    os.system('/usr/bin/cvlc --play-and-exit /home/david/facial/voice.m4a 2>/dev/null') #Ejecuto VLC y reproduzco el archivo enviado en la raspberry.

```

*Figura 26 – Función reciboaudio*

Esta función se encarga de reproducir una nota de voz recibida desde el bot de Telegram. La función descarga el archivo de audio y lo guarda en la ruta '/home/david/facial/voice.m4a'. Luego, ejecuta el reproductor VLC en la Raspberry Pi y reproduce el archivo de audio descargado en el parlante externo de mi Raspberry Pi.

Esto está pensado para tener una comunicación por audio desde el Telegram y la Raspberry, pudiéndose así comunicar desde cualquier parte si uno no se encuentra en la propia casa, tal y como se muestra en la siguiente figura:

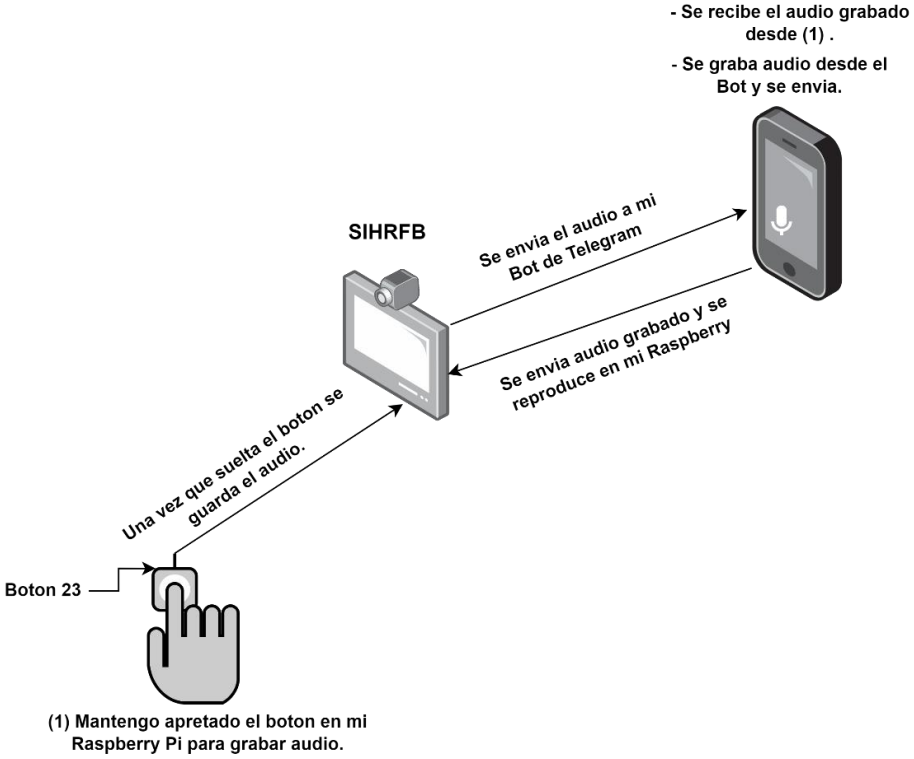



*Figura 27 – Funcionamiento de recibir y enviar audio*

### 5.1.11 Función del envió del Correo

Esta función tiene como objetivo enviar un correo electrónico con una foto adjunta al correo electrónico del propietario de la casa cuando alguien toca el timbre.

Primero, se abren dos archivos de texto ("correo.txt" y "correo2.txt", ver figura 28) que contienen la dirección de correo electrónico del propietario y su contraseña, respectivamente. A continuación, se crea un objeto **MIMEMultipart** para definir el formato del mensaje de correo electrónico y se establecen los campos "Subject", "From" y "To".

Se crea un mensaje en HTML que incluye la hora y fecha en la que se tomó la foto y se adjunta la imagen capturada en el mensaje (figuras 17 y 18). El contenido del mensaje se define como **MIMEText** y la imagen se define como **MIMEImage**.

Por último, se crea una conexión segura con el servidor SMTP de Gmail y se autentica con las credenciales proporcionadas. Luego se envía el mensaje de correo electrónico utilizando la función **sendmail()** de SMTP, que toma el remitente, destinatario y el mensaje como parámetros.

```
def correos():
    p = open('correo.txt','r')
    email_address = str(p.readline())

    p = open('correo2.txt','r')
    email_password = str(p.readline())
#----------------------------------------Crear el mensaje-----------------------------------------------------
    mensaje = MIMEMultipart("alternative") #Estandar
    mensaje["Subject"] = "TIMBRE CASA"
    mensaje["From"] = email_address
    mensaje["To"] = email_address
    d = time.strftime("%H:%M %p del %d-%m-%Y")
#Ponemos el formato en html
    html = f"""
    <html>
    <body>
        Hola,<br>
        Alguien está tocando el timbre a las %s, te adjunto foto.<br>
        ¡Qué tengas un lindo día!
    </body>
    </html>
    """ %(d) #Lo que hago aca es pasar dia y hora que se tomo la foto
# las tres comillas me dejan escribir un string largo, <br> es salto de linea y poner entre <b> </b> se pone en negrita
# el contenido del mensaje como HTML
    parte_html = MIMEText(html, "html")

#-------------------------------------Abro la imagen y la cargo-------------------------------------------------
    f = open("foto.png", "rb")
    image = f.read()
    f.close()
    parte_imagen = MIMEImage(image)

#-------------------------------------agregar los contenidos al mensaje-----------------------------------------
    mensaje.attach(parte_html)
    mensaje.attach(parte_imagen)
#-----------------------------Crear la conexion y enviar el mensaje-------------------------------------------------
    context = ssl.create_default_context()
    with smtplib.SMTP_SSL("smtp.gmail.com", 465, context=context) as server:
        server.login(email_address,email_password) #Inicio de sension
        print("Inicio de sesion")
        server.sendmail(email_address, email_address, mensaje.as_string()) #Enviar email, mensaje.as_string() envia el mensaje como string
        print("Mensaje enviado")
```

*Figura 28 – Función enviar correo*

### 5.1.12 Función Agregar correo y Agregar Contraseña

Esta función lo que me permite es agregar un correo (solo Gmail) para recibir las notificaciones de cuando alguien está tocando timbre.

```
339  #----------------------------------------Funciona para guardar email ----------------------------------------
340  def AgregarCorreo(update, context):
341      button1 = InlineKeyboardButton(text='Agregar Correo',callback_data='ncorreo') # Llama a la funcion correo1
342      button2 = InlineKeyboardButton(text='Cancelar',callback_data='cancelar')    #Llama a la funcion cancelar (linea 161)
343
344      update.message.reply_text(text='Eliga una Opcion:', reply_markup=InlineKeyboardMarkup([
345          [button1, button2]
346          ]))
347
348  def correo1(update, context):
349      query = update.callback_query
350      query.answer()
351      query.edit_message_text(text="Escriba el correo (gmail) a agregar: ")
352      return INPUT_TEXT3
353
354  def input_text3(update, context):
355      global chat_id
356      correo = update.message.text
357      dispatcher.bot.sendMessage(chat_id, text=f'Correo: {correo} Guardado.')
358      archivo = open ("/home/david/facial/correo.txt", "w") #Creo y abro el archivo como solo escritura.
359      archivo.write(correo) #Escribo el gmail.
360      archivo.close
361      return ConversationHandler.END
```

*Figura 29 – Función agregar correo*

Sin embargo, acá usamos manejadores de conversaciones que se explicara a continuación como actúa:

1. Crea dos botones en el chat del usuario para que elija una opción: "Agregar Correo" y "Cancelar".

2. Cuando el usuario hace clic en "Agregar Correo", se llama a la función "correo1".

3. La función "correo1" edita el mensaje anterior para pedirle al usuario que escriba el correo electrónico que desea agregar.

4. Cuando el usuario escribe y envía el correo electrónico, se llama a la función "input_text3" (espera una entrada de texto).

5. La función "input_text3" toma el correo electrónico ingresado y lo guarda en un archivo de texto llamado "correo.txt".

6. Finalmente, la función envía un mensaje de confirmación al usuario y termina la conversación.

7. Si el Usuario hace clic en "Cancelar", se llama a la función "cancelar" como se muestra en la siguiente figura.

```python
#---------------------------------------Funcion para cancelar lo de agregar nombre-----------------------------------------
def cancelar(update, context):
    query = update.callback_query
    query.answer()
    query.edit_message_text(text="\u274C Operacion Cancelada. \u274C")
    return ConversationHandler.END
```

*Figura 30 – Función cancelar*

8. Responde a la consulta del usuario con un mensaje de texto que indica que la operación ha sido cancelada.

9. Finaliza la conversación utilizando la constante ConversationHandler.END, lo que permite que el usuario vuelva a interactuar con el bot.

A continuación, se muestra un diagrama de flujo de cómo sería una conversación:

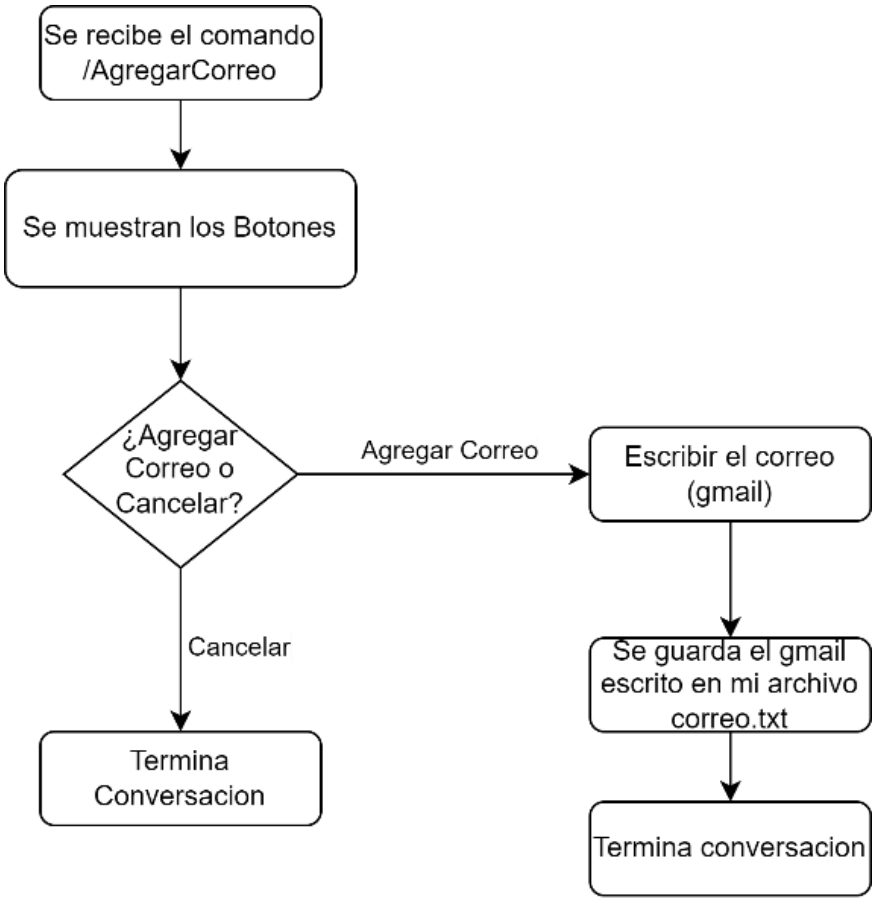



*Figura 31 – Diagrama de flujo del funcionamiento para agregar correo*

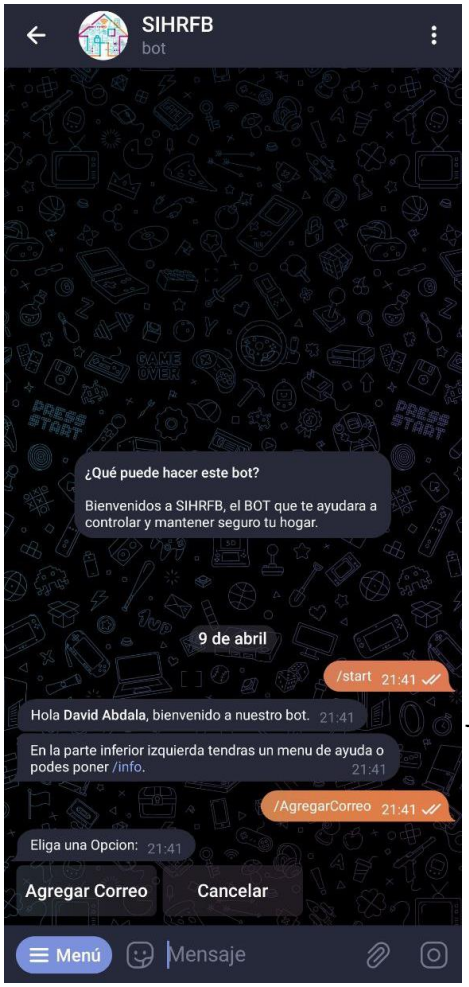



*Figura 32 - Botones a Elegir*

Al apretar el botón **Agregar Correo**, se escribe el Gmail y se guarda.


*Figura 33 – Agregando Correo*

Al apretar el botón **Cancelar**, me cancela la operación y termina la conversación.

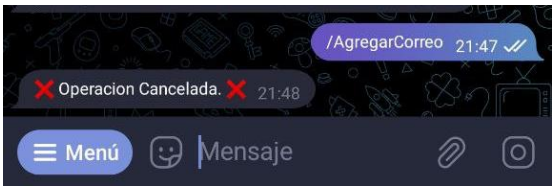



*Figura 34 – Cancelar agregar un correo*

Ahora mostramos la función AgregarContrasena():

```
363   #-----------------------------------------Funciona para guardar contraseña -------------------
364 def AgregarContrasena(update, context):
365     button1 = InlineKeyboardButton(text='Agregar Contraseña',callback_data='acontra')
366     button2 = InlineKeyboardButton(text='Cancelar',callback_data='cancelar')
367 
368     update.message.reply_text(text='Eliga una Opcion:', reply_markup=InlineKeyboardMarkup([
369         [button1, button2]
370         ]))
371 
372 def correo2(update, context):
373     query = update.callback_query
374     query.answer()
375     query.edit_message_text(text="Escriba la contraseña que te genero en 'contraseña de aplicaciones': ")
376     return INPUT_TEXT4
377 
378 def input_text4(update, context):
379     global chat_id
380     correo2 = update.message.text
381     dispatcher.bot.sendMessage(chat_id, text=f'Contraseña: {correo2} Guardada.')
382     archivo = open ("correo2.txt", "w")
383     archivo.write(correo2)
384     archivo.close
385     return ConversationHandler.END
386 
```

*Figura 35 – Función agregar la contraseña*

Esta función lo que me permite es agregar la **CONTRASEÑA DE LA APLICACIÓN PARA EL DISPOSITIVO** (se explica en anexo l como obtener esta contraseña), a continuación, se explica que hace:

1. Crea dos botones en el chat del usuario para que elija una opción: “Agregar Contraseña o “Cancelar” la operación.

2. Cuando el usuario hace clic en "Agregar Contraseña", se llama a la función "correo2".

3. La función "correo2" edita el mensaje anterior para pedirle al usuario que escriba la contraseña que le fue generada en "contraseña de aplicaciones".

4. Cuando el usuario escribe y envía la contraseña, se llama a la función "input_text4" (espera una entrada de texto).

5. La función "input_text4" toma la contraseña ingresada y la guarda en un archivo de texto llamado "correo2.txt”.

6. Finalmente, la función envía un mensaje de confirmación al usuario y termina la conversación.

7. La función “Cancelar” es la misma que se explicó para /agregarcorreo.

El diagrama de flujo para este comando es igual al que fue descripto en mi Figura 31, lo único que cambia es que se guarda en mi archivo ‘correo2.txt’.

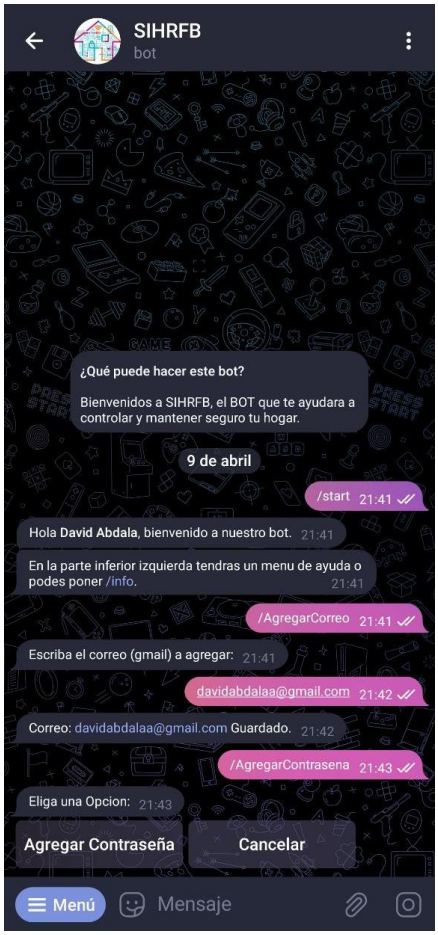



*Figura 36 - Botones a elegir*

Al apretar el botón **Agregar Contraseña**, se escribe la contraseña y se guarda.

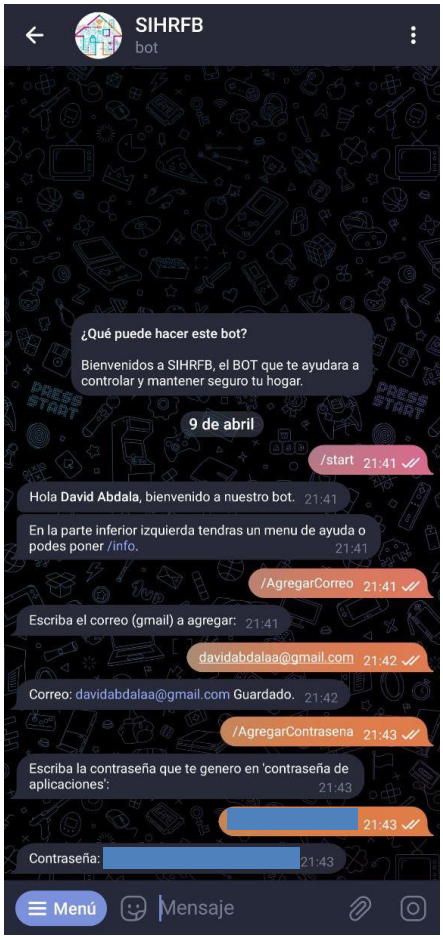



*Figura 37 – Agregando Contraseña*

Si se aprieta el botón ‘Cancelar’, el mensaje que se muestra es igual al que aparece en mi Figura 30.

### 5.1.13 Función para ver el correo agregado

Para saber si el correo y la contraseña agregada son correctas, lo que podemos hacer es enviar el comando /vercorreo, y nos muestra el correo y contraseña guardada en mis archivos ‘correo.txt’ y ‘correo2.txt’.

```
450  #------------------------------------ VISUALIZADOR DE ARCHIVOS ---------------------
451  def vercorreo(update, context):
452      global chat_id
453      archivo=open("/home/david/facial/correo.txt")
454      dispatcher.bot.sendMessage(chat_id, text=f'Correo guardado: {archivo.read()}')
455      archivo1=open("/home/david/facial/correo2.txt")
456      dispatcher.bot.sendMessage(chat_id, text=f'Contraseña guardada: {archivo1.read()}')
457      archivo.close
458      archivo1.close
```

*Figura 38 – Función para ver correo y contraseña guardada*

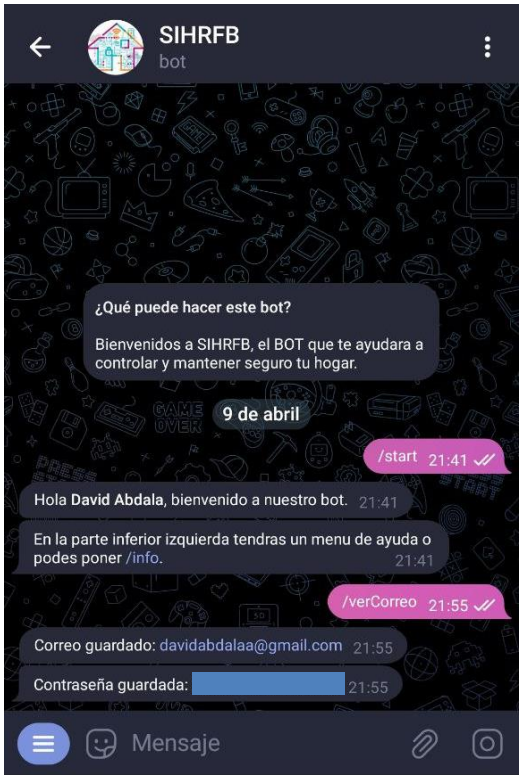



*Figura 39 – Muestra lo guardado en mis archivos*

### 5.1.14 Función notificación por presionar botón timbre

Al recibir la señal de llamada **SIGUSR2**, lo que hace es llamar a la función **timbre()**, explicándose a continuación:

```
320  #-----------------------------------------Avisa cuando alguien toca el timbre, reproduce el sonido de timbre y envi
321  def timbre(update, context):
322      global chat_id
323      camera = PiCamera()
324      camera.rotation = 180
325      camera.resolution = (1280, 720)
326      dispatcher.bot.sendMessage(chat_id, '\U0001f6ce\uFE0F Alguien esta tocando el timbre!\nSacando Foto.')
327      os.system('/usr/bin/cvlc --play-and-exit /home/david/facial/Doorbell.wav 2>/dev/null') #Ejecuta el audio grabado
328      dispatcher.bot.sendChatAction(chat_id, 'upload_photo')
329      sleep(2) #Tiempo de calentamiento de la cámara
330      camera.capture('foto.png', format='png', use_video_port=False) #Capturando la foto
331      camera.close()
332      photofile = open('foto.png', 'rb') #Abriendo archivo creado y enviandolo
333      dispatcher.bot.sendPhoto(chat_id, photo=photofile)
334
335      enviarcorreo = mp.Process(target= correos) #Creo un subproceso para enviar el correo al email.
336      enviarcorreo.start()
337      enviarcorreo.join()
```

*Figura 40 – Función notificación timbre*

Esta función envía un mensaje de texto al chat indicando que alguien está tocando el timbre y se procede a reproducir el sonido de timbre grabado previamente mediante el programa **'cvlc'** (Este sonido se escucha por un parlante externo conectado adentro de la casa, como un timbre normal). A continuación, se espera 2 segundos para que la cámara se caliente antes de capturar una foto, que se envía al chat como un archivo adjunto. Después, se crea un subproceso que llama a mi función **'correos'** de mi archivo **correo.py** para enviar un correo electrónico con la foto capturada.

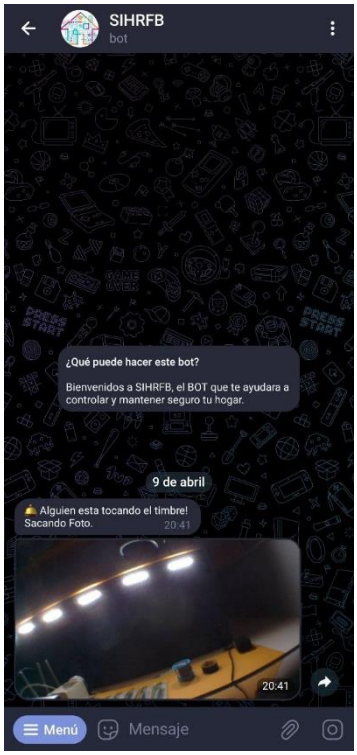



*Figura 41 – Notificación a Telegram*

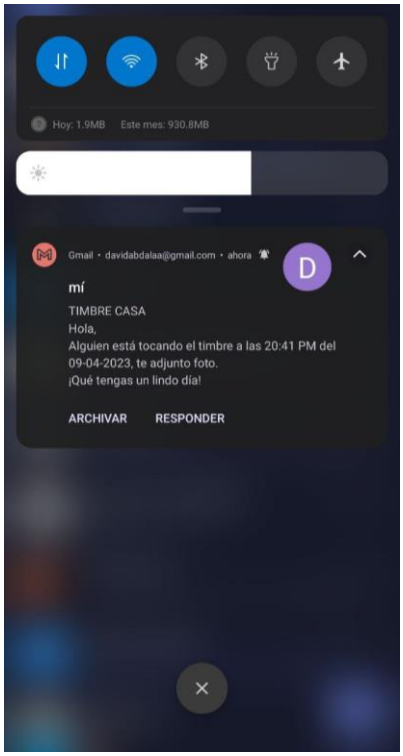



*Figura 42 – Notificación al correo*

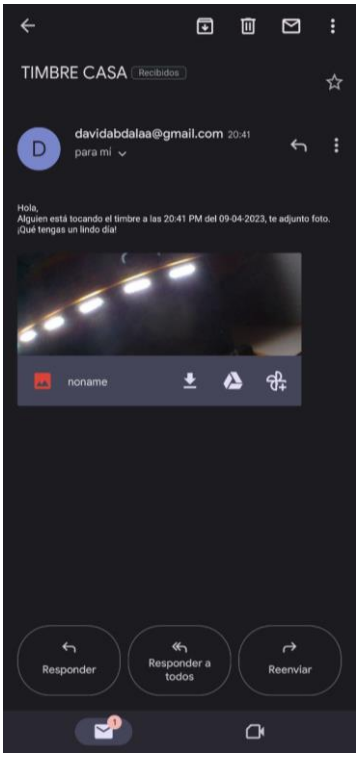



*Figura 43 – Contenido del correo enviado*

### 5.1.15 Función Foto

Esta función permite capturar una foto en tiempo real desde la cámara del dispositivo Raspberry Pi y subirla al bot, entonces, cuando el Usuario envía el comando /foto, se instancia un objeto de la clase PiCamera para poder interactuar con la cámara. Se establece la rotación y la resolución de la imagen.

```
62  #---------------------------------------Funcion para la captura y
63  def foto(update, context):
64      chat_id = update.message.chat_id
65      camera = PiCamera()
66      camera.rotation = 180
67      camera.resolution = (1280, 720)
68      context.bot.sendMessage(chat_id, 'Sacando Foto')
69      context.bot.sendChatAction(chat_id, 'upload_photo')
70      #Tiempo de calentamiento de la cámara
71      sleep(2)
72      #Capturando la foto
73      camera.capture('foto.png', format='png', use_video_port=False)
74      camera.close()
75      #Abriendo archivo creado y enviandolo
76      photofile = open('foto.png', 'rb')
77      context.bot.sendPhoto(chat_id, photo=photofile)
78
```

*Figura 43 – Función foto*

Luego, se envía un mensaje al usuario indicando que se está capturando la foto y se simula la acción de cargar una foto utilizando la función **sendChatAction** (Figura 44). A continuación, se espera 2 segundos para que la cámara se caliente antes de capturar la foto (esto es si o si necesario).


*Figura 44 – Aviso de envío de foto*

Se utiliza la función capture de la instancia de PiCamera para capturar la foto y se guarda en un archivo llamado **'foto.png'**. Luego se cierra el objeto de la cámara para liberar recursos y finalmente, se abre el archivo **'foto.png'** y se la envía al usuario utilizando la función **sendPhoto.**

### 5.1.16 Función Video

Esta función permite la grabación de un video utilizando una cámara Raspberry Pi. Cuando el Usuario envía el comando /video, se instancia un objeto de la clase PiCamera para poder interactuar con la cámara. Se establece la rotación y la resolución de la imagen.

```python
#----------------------------------------Funcion para la grabacion y suba de video desde la camara Pi----------------
def video(update, context):
    camera = PiCamera()
    camera.rotation = 180
    camera.resolution = (1280, 720)
    chat_id = update.message.chat_id
    context.bot.sendMessage(chat_id, 'Espere unos Segundos mientras se sube el Video')
    context.bot.sendChatAction(chat_id, 'record_video') #Le dice al usuario que algo está sucediendo del lado del bot
    #Creo el archivo y empiezo la grabacion
    filename = "./video_22"
    camera.start_recording(filename + ".h264")
    camera.wait_recording(5)
    camera.stop_recording()
    camera.close() #método para liberar los recursos de la cámara
    #Transformo el formato .h264 a .mp4 llamando por consola
    command = "MP4Box -add " + filename + '.h264' + " " + filename + '.mp4'
    call([command], shell=True)
    #Envio el video
    context.bot.sendChatAction(chat_id, 'upload_video')
    context.bot.sendVideo(chat_id, video = open(filename + '.mp4', 'rb'))
    context.bot.sendMessage(chat_id, 'video enviado')
    #Elimino los archvios creados asi no tengo problemas de espacio
    command = "rm " + filename + '.h264'
    call([command], shell=True)
    command = "rm " + filename + '.mp4'
    call([command], shell=True)
```

*Figura 45 – Función video*

Se explicará lo que pasa desde la línea 81 a la 88:

- Se establece un nombre de archivo para el video.
- Se inicia la grabación del video en formato H264.
- Se espera 5 segundos mientras se graba el video.
- Se detiene la grabación del video.
- Se cierra el objeto de la cámara para liberar recursos.
- Se utiliza el comando MP4Box para convertir el video de formato H264 a formato MP4.
- Se llama al comando MP4Box a través de la consola.

Una vez hecho esto, se sube el video al bot de Telegram y se borra el video creado, esto se hace así para no tener archivos que me ocupen espacio en mi disco

### 5.1.17 Función puerta y prender la luz

Esta función permite controlar el estado de un relé conectado a un solenoide para poder abrir una puerta. Lo que hace es: Cuando se llama a la función a través del bot de Telegram, se activa el relé (abre la puerta), se espera 5 segundos para que la puerta se abra, se desactiva el relé (cierra la puerta), y se envían mensajes al chat indicando el estado del relé (encendido o apagado) y, por lo tanto, el estado de la puerta (abierta o cerrada).

```
387  #-------------------------------------- Funcion para abrir la puerta con el rele ----
388  def puerta(update, context):
389      global chat_id
390      relay = gpiozero.OutputDevice(RELAY_PIN, active_high=True, initial_value=False)
391      relay.on() # Rele on
392      dispatcher.bot.sendMessage(chat_id, "Rele de puerta en estado: Encendido")
393      sleep(5)
394      relay.off() # rele off
395      dispatcher.bot.sendMessage(chat_id, "Rele de puerta en estado: Apagado")
396
```

*Figura 46 – Función para abrir puerta*

Se explicará lo que pasa desde la línea 388 a la 395:

- Se crea un objeto de la clase OutputDevice de la librería gpiozero para controlar el relé, especificando el pin donde está conectado y los valores de activación e inicialización.
- Se activa el relé (se cierra el circuito) para abrir la puerta.
- Se envía un mensaje al chat indicando que el relé está encendido (la puerta está abierta).
- Se espera 5 segundos (tiempo suficiente para que la puerta se abra).
- Se desactiva el relé (se abre el circuito) para cerrar la puerta.

Se envía un mensaje al chat indicando que el relé está apagado (la puerta está cerrada).

La función **/prenderluz** es análoga a **/puerta**, utiliza el mismo criterio, cambiando algunas variables de texto nada más.

```
459  #-------------------------------------- Prender LUZ ------------------------
460  def prenderluz(update, context):
461      global chat_id
462      luz = gpiozero.OutputDevice(LUZ_PIN, active_high=True, initial_value=False)
463      luz.on() # Rele on
464      dispatcher.bot.sendMessage(chat_id, "Luz encendida")
465      sleep(5)
466      luz.off() # rele off
467      dispatcher.bot.sendMessage(chat_id, "Luz apagada")
468
```

*Figura 47 – Función para prender la luz*

### 5.1.18 Función para Sensores PIR

La detección de movimiento por sensores PIR se activa mediante el comando /pir y su funcionamiento se puede ver en la siguiente figura.

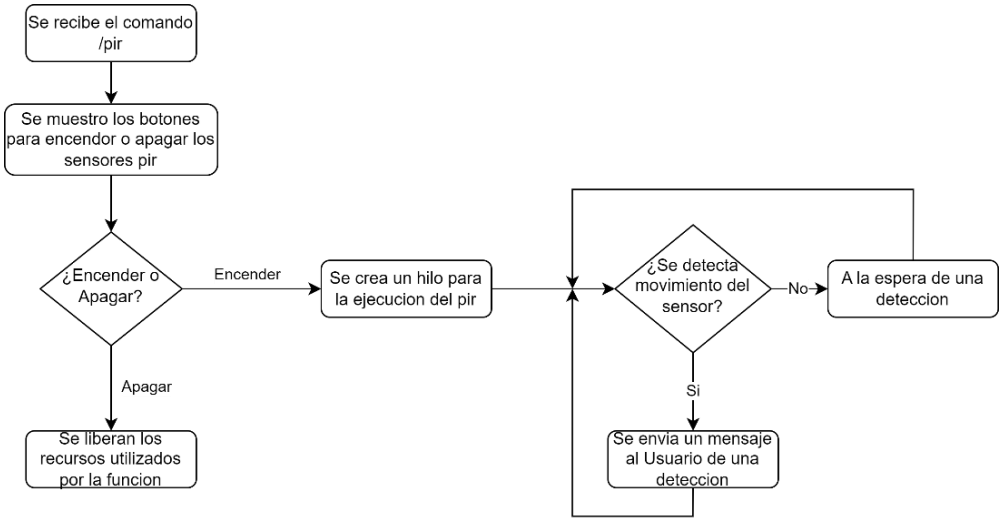



*Figura 48 – Diagrama de flujo del funcionamiento para los sensores PIR*

Para poder recibir datos de los sensores PIR, los pines GPIO correspondientes a cada sensor tienen que ser configurados como entradas:

```
397  #-------------------------------------- Funcion para encender o apagar el sensor PIR--------------------------------
398  def pir(update, context):
399      button1 = InlineKeyboardButton(text='Encender',callback_data='encender') #Al apretar aca hago un call con el dato
400      button2 = InlineKeyboardButton(text='Apagar',callback_data='apagar')
401
402      update.message.reply_text(text='¿Encender o Apagar sensor de movimiento?.', reply_markup=InlineKeyboardMarkup([
403          [button1, button2]
404          ])
405      )
406
407  def encender(update, context):
408      query = update.callback_query
409      global estado
410      if estado == 1:
411          query.edit_message_text(text="El sensor PIR ya esta encendido!")
412          return ConversationHandler.END
413      elif estado == 0:
414          estado = 1
415          pir_start(estado)
416          query.edit_message_text(text="Sensor de movimiento encendido")
417          return ConversationHandler.END
418
419  def apagar(update, context):
420      query = update.callback_query
421      global estado
422      if estado == 0:
423          query.edit_message_text(text="El sensor PIR ya esta apagado")
424          return ConversationHandler.END
425      elif estado == 1:
426          estado = 0
427          pir_start(estado)
428          query.edit_message_text(text="Sensor de movimiento apagado")
429          return ConversationHandler.END
430
431  def pir_start(valor): #Arranco para empezar a usar el PIR
432      valor=int(valor)
433      global t
434      if valor == 1:
435          t = threading.Thread(target=pir_running)
436          t.start()
437      elif valor == 0:
438          t.do_run = False
439          #print("se apago") era para verificar si llegaba a este punto
440
441  def pir_running(): #Funcion para correr el PIR, al usar hilos no me traba los demas programas
442      global chat_id
443      global t
444      pir = MotionSensor(4)
445      t = threading.currentThread()
446      while getattr(t, "do_run", True):
447          pir.wait_for_motion()
448          dispatcher.bot.sendMessage(chat_id, "Se detecta presencia en puerta de entrada!")
449          pir.wait_for_no_motion()
```

*Figura 49 – Función del sensor PIR*

Estas funciones están relacionadas con el manejo de un sensor PIR (sensor de movimiento) conectado al pin 4 de una Raspberry Pi. La función **pir** muestra un mensaje al usuario con dos opciones para encender o apagar el sensor PIR. Esta función utiliza botones de teclado inline para que el usuario seleccione una opción y se utiliza el objeto **update** para enviar un mensaje de respuesta al usuario.

Las funciones **encender** y **apagar** se llaman cuando el usuario selecciona la opción de encender o apagar el sensor en el teclado inline. Ambas funciones verifican si el estado actual del sensor PIR es el mismo que se desea cambiar (encender o apagar) y si es necesario, llama a la función **pir_start** que se encarga de iniciar o detener la detección de movimiento a través del sensor PIR.

La función **pir_start** recibe un parámetro **valor** que indica si se debe encender o apagar el sensor PIR. Si se debe encender, se crea un hilo **t** que ejecuta la función **pir_running**. Si se debe apagar, se cambia el valor de la variable **do_run** en el hilo **t** a **False**, lo que detendrá la ejecución de la función **pir_running**.

La función **pir_running** es la que realmente detecta el movimiento a través del sensor PIR. Se crea un objeto **pir** de la clase **MotionSensor** que se inicializa con el pin 4. La función luego espera a que se detecte un movimiento utilizando el método **wait_for_motion()** del objeto **pir**. Cuando se detecta un movimiento, envía un mensaje de texto a través del objeto **dispatcher.bot** indicando que se ha detectado movimiento en la puerta de entrada. Luego, espera a que se detenga el movimiento utilizando el método **wait_for_no_motion()** del objeto **pir**. Esta función se ejecutará continuamente mientras la variable **do_run** en el hilo **t** sea verdadera.

Se muestra a continuación un diagrama de flujo de la verificación de los sensores, de acuerdo a los botones apretados.

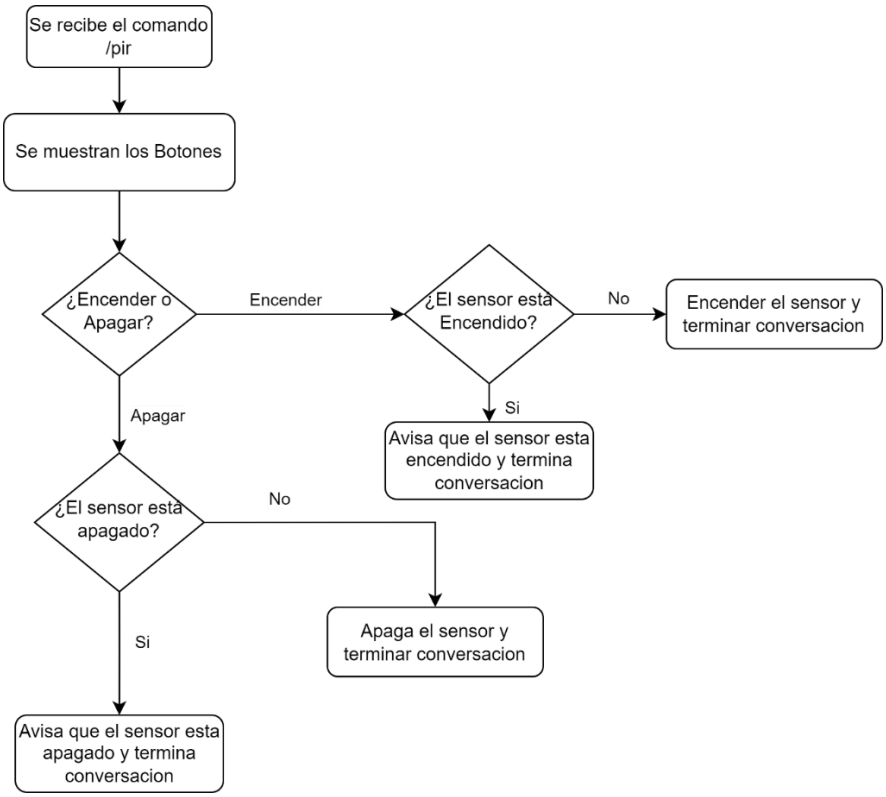



*Figura 50 – Diagrama de flujo de los botones para el comando /pir*

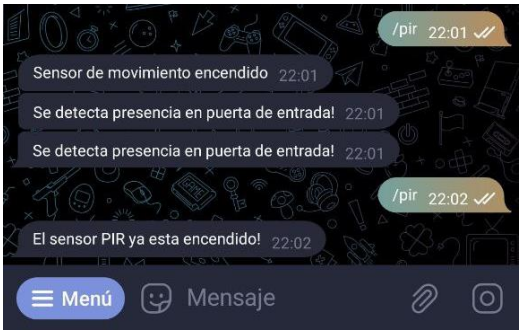
Funcionamiento en Bot de Telegram:


Se envía el comando /pir y se aprieta Encender.

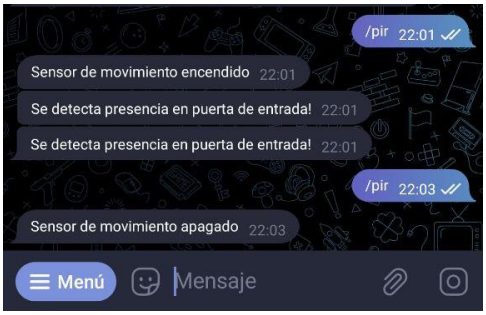


Se envía el comando /pir y se aprieta Encender.

Avisa que el sensor ya está encendido.

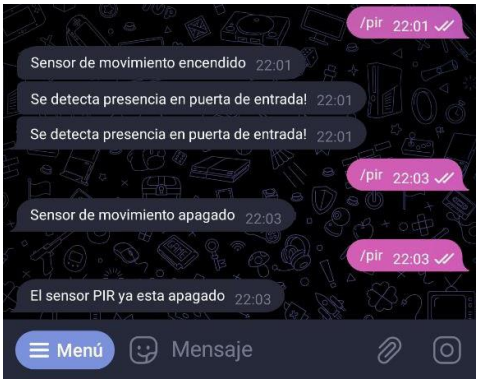



Se aprieta Apagar y avisa que apagó el sensor.




Una vez apagado, se envía el comando /pir y se aprieta Apagar.


Avisa que el sensor ya está apagado.



### 5.1.19 Función Reiniciar

Esta función es utilizada para reiniciar el sistema de la Raspberry Pi, en caso de que la misma no esté funcionando correctamente.

```
475   #------------------------------------ Funcion para reiniciar el sistema----------------------------------------------
476   def reinicar(update, context):
477       global chat_id
478       button1 = InlineKeyboardButton(text='SI',callback_data='si') #Al apretar aca hago un call con el dato "si" y me llama a
479       button2 = InlineKeyboardButton(text='NO',callback_data='cancelar')
480
481       update.message.reply_text(text='¿ESTA SEGURO QUE QUIERE REINICIAR EL SISTEMA?.', reply_markup=InlineKeyboardMarkup([
482           [button1, button2]
483           ])
484       )
485
486   def si(update, context):
487       query = update.callback_query
488       query.answer()
489       query.edit_message_tex(text="Reiniciando Sistema, espere unos minutos y mande nuevamente el comando /start")
490       sleep(3)
491       command = "sudo reboot now"
492       call([command], shell=True)
493
```

*Figura 51 – Función reiniciar*

Primero se utiliza la función **reiniciar()** para preguntar al usuario si desea reiniciar el sistema, proporcionando botones de "SI" y "NO". Si el usuario presiona "SI", se llama a la función **si()** a través de un **CallbackQueryHandler**. Dentro de esta función, se envía un mensaje que indica que se está reiniciando el sistema y se espera 3 segundos antes de ejecutar el comando **sudo reboot now**, que reinicia el sistema. Es importante destacar que el usuario debe enviar nuevamente el comando **/start** después de unos minutos para volver a interactuar con el bot.

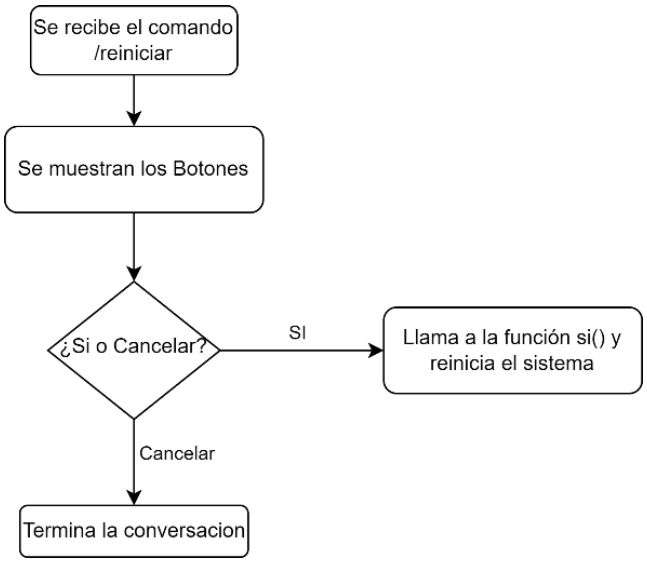



*Figura 52 – Diagrama de flujo al enviar el comando /reiniciar*

### 5.1.20 Reconocimiento Facial

Se retoma desde el archivo “todo.py”, ya que el código que se muestra a continuación se encuentra en el mismo programa, por lo que se pasa a explicar cuál fue la solución que se dio para este caso.

El principal obstáculo que se presentó en este caso fue la limitada capacidad de RAM y procesamiento de la Raspberry utilizada, lo cual impedía llevar a cabo el reconocimiento facial a través de la cámara. Por lo tanto, no era posible encender la cámara, localizar una cara y **registrarla**. Ante esta dificultad, se tomó la decisión de recurrir a una alternativa, la cual consistía en grabar un video de al menos 10 segundos de duración, en el cual se mostrará claramente el rostro y que contará con una iluminación adecuada (como se muestra en la figura 53). Una vez que se obtuvo el video, se procede a subirlo al Bot de Telegram. Al recibir el video, el Bot se encarga de procesarlo y detectar el rostro en cuestión, generando un registro que consta de aproximadamente 30 imágenes de la cara. Esta metodología se implementó con el objetivo de lograr una mayor precisión y confiabilidad en el reconocimiento de la cara.

*Figura 53 – Video Frontal de la cara*

### 5.1.21 Función al recibir un video desde Telegram con la cara a registrar

Esta función “reciboVideo()” se encarga de guardar un video recibido por Telegram, como se ve en la figura.

```
156  #---------------------------------------Funcion para recibir un Video desde el BOT y utilizarlo para el reconocimie
157  def reciboVideo(update, context):
158      #Con esta funcion en caso de recibir video de telegram lo guardo
159      file = context.bot.getFile(update.message.video.file_id)
160      file.download('video.mp4')
161      button1 = InlineKeyboardButton(text='Agregar Nombre',callback_data='nombre') #Al apretar aca hago un call con el
162      button2 = InlineKeyboardButton(text='Cancelar',callback_data='cancelar')
163
164      update.message.reply_text(text='Agregue el Nombre o cancele la operación.', reply_markup=InlineKeyboardMarkup([
165          [button1, button2]
166          ])
167      )
168
```

*Figura 54 – Función cuando recibo el video*

La función utiliza el objeto "context" para obtener el archivo de video asociado al mensaje recibido. Luego, utiliza el método "download" del objeto "file" para descargar el archivo de video en el directorio local con el nombre "video.mp4". Luego crea un Inline en el Bot con dos opciones: Agregar Nombre o Cancelar.

En la siguiente imagen se muestra un flujograma al momento de recibir un video desde el Bot de Telegram:

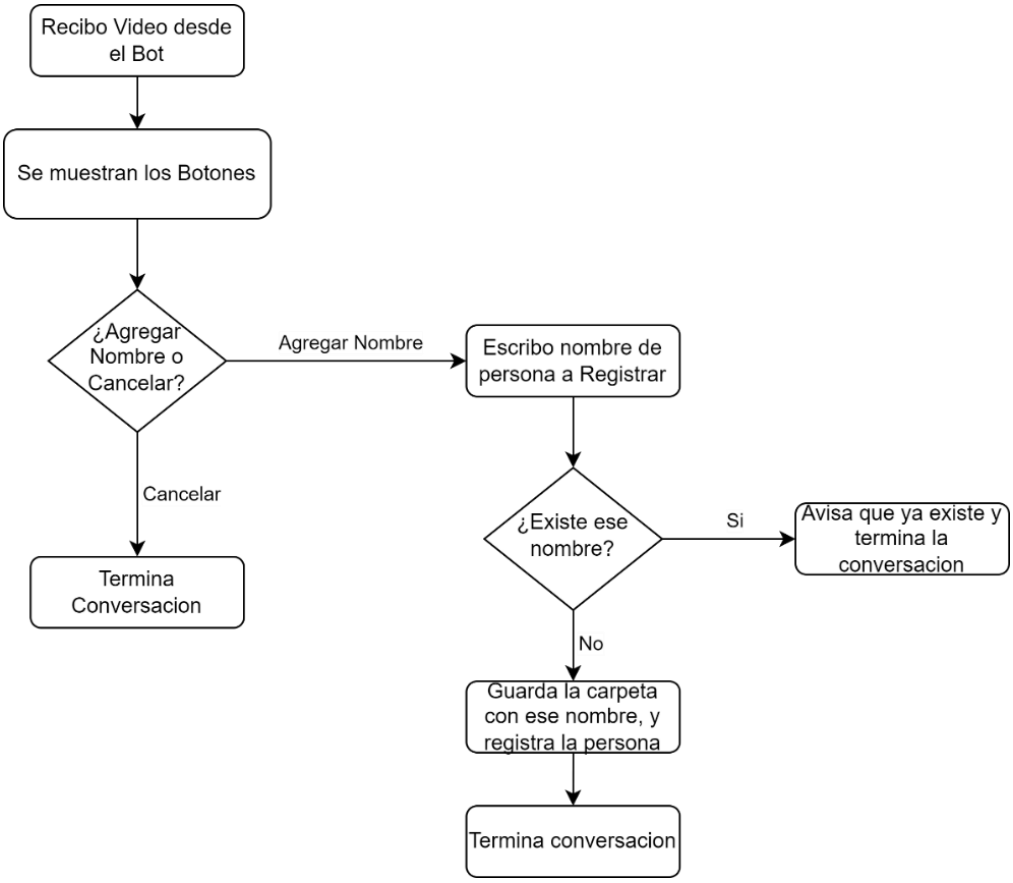



*Figura 55 – Diagrama de flujo del proceso al recibir un video frontal de la cara*

Al apretar agregar nombre, automáticamente me llama a la función “registro_cara()” (Figura 4), el cual espera una entrada de texto para guardar el nombre. Cuando se escribe el nombre a guardar, puede suceder dos situaciones: La primera situación es que el nombre a registrar ya se encuentre la carpeta, por lo tanto, el programa avisa que este nombre ya existe y le pide que vuelva a grabar o se puede enviar nuevamente el video recibido para registrarlo con otro nombre. La segunda situación es que se crea la nueva carpeta con el nombre ingresado y pasa a la función “ReconocerCara()”.

```python
#---------------------------------------Funciones para registrar el nombre y crear la carpeta-----------------------------------------
def registro_cara(update, context):
    query = update.callback_query
    query.answer()
    query.edit_message_text(text="Escriba el nombre a guardar por favor: ")
    return INPUT_TEXT

def input_text(update, context):
    nombre = update.message.text
    dataPath = '/home/david/facial/Data'
    personaPath = dataPath + '/' + nombre

    if os.path.exists(personaPath):
        dispatcher.bot.sendMessage(chat_id, '\u203C\uFE0F Ya existe esta carpeta con este nombre, volver a grabar y registrar otro Nombre.')
        command = "rm " + 'video.mp4'
        call([command], shell=True)
        return
    else:
        dispatcher.bot.sendMessage(chat_id, text=f'\u2705 Carpeta creada con el nombre de: {nombre}.')
        os.makedirs(personaPath)

    ReconocerCara(nombre)
    return ConversationHandler.END
```

*Figura 56 – Función de registro de cara*

**Función ReconocerCara(nombre):** Este código se encarga de detectar caras en un video y almacenarlas en una carpeta con el nombre de la persona correspondiente. Además, luego se utiliza esa carpeta para entrenar un modelo de reconocimiento facial y almacenarlo para su uso posterior. En la siguiente figura se muestra el código en cuestión:

```python
#---------------------------------------Funciones  para registrar la cara y entrenarla-----------------------------------------
def ReconocerCara(nombre):
    global chat_id
    dataPath = '/home/david/facial/Data'
    personaPath = dataPath + '/' + nombre
    dispatcher.bot.sendMessage(chat_id, '\u26A0\uFE0F Espere mientras se registra la Cara \u231A.')
    cap = cv2.VideoCapture('video.mp4') #Lee el video recibido desde telegram.
    faceClassif = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')#Cargo el clasificador
    count = 0
    while True:
        ret, frame = cap.read() #lee los fotogramas de una cámara y ret comprueba el retorno en cada fotograma.
        if ret == False:
            break

        frame = imutils.resize(frame, width=640) #Se redimensiona la imagen, esto lo hago para redimensionar el tamaño de los fotogramas del video de entrada.
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY) #Función cvtColor() para escalar en gris la imagen. Frame se convierte en gris.
        auxFrame = frame.copy()
        faces = faceClassif.detectMultiScale(gray,1.3,5)

        for(x,y,w,h) in faces:
            #cv2.rectangle(frame, (x,y),(x+w,y+h), (0,255,0),2) #dibuja una forma rectangular en la imagen, para despues poder extraer esa parte de la imagen.
            rostro = auxFrame[y:y+h, x:x+w]
            rostro = cv2.resize(rostro, (150,150), interpolation=cv2.INTER_CUBIC) #Se redimensionan las imagenes para que todas posean el mismo tamaño.INTER_CUBIC se usa para hacer zoom.
            cv2.imwrite(personaPath + '/rostro_{}.jpg'.format(count), rostro) #cv2.imwrite() para guardar una imagen. El primer argumento es el nombre del archivo, el segundo argumento es la imagen que desea guardar.
            count = count + 1 #contador creciente para que muestre el número de fotogramas creados

        k = cv2.waitKey(1)
        if k == 27 or count >= 30:
            break
    dispatcher.bot.sendMessage(chat_id, '\u2705 Proceso de registro de cara terminado!')
    command = "rm " + 'video.mp4'
    call([command], shell=True)
    cap.release() #libera recurso de software y hardware de la camara, porque si no salta un error en donde dice: Dispositivo o recurso ocupado.
    entrenamiento(dataPath)
    cv2.destroyAllWindows()
```

*Figura 57 – Función reconocer cara y registrarla*

La función recibe como parámetro el nombre de la persona a la que se le va a registrar su rostro. En primer lugar, se establece la ruta del directorio de datos donde se guardarán las imágenes. Luego, se lee el video recibido desde Telegram y se define un clasificador de HaarCascade para la detección de rostros (este ya viene predefinido para lo que es rostro, siendo el archivo XML proporcionado por el directorio haarcascade).

El programa recorre cada fotograma del video y lo redimensiona a un tamaño de 640x480 píxeles. A continuación, convierte la imagen en escala de grises para facilitar la detección de rostros y detecta la posición de los mismos en el fotograma. Si se encuentra un rostro, se recorta la imagen y se almacena en la carpeta correspondiente con un nombre de archivo "rostro_**i**.jpg", donde **i** es un contador creciente que representa la cantidad de fotogramas creados.

El bucle principal del programa se detiene cuando se ha procesado el número máximo de fotogramas (30 en este caso). Después de la detección de todos los rostros, se elimina el video original y se llama a la función "entrenamiento(dataPath)" para entrenar un modelo de reconocimiento facial con las imágenes guardadas, como se puede observar en la siguiente figura:

```python
def entrenamiento(dataPath):
    global chat_id
    dispatcher.bot.sendMessage(chat_id, '\u26A0\uFE0F Espere mientras se Entrenan las caras.')
    personaList = os.listdir(dataPath) #Lista directorios
    #Creamos dos Array, esto es porque  la computadora debe saber a quien corresponde cada cara
    #por lo que le asignamos etiquetas, asi se puede diferenciar de una persona de otra.
    labels = []
    facesData = []
    label = 0

    for nameDir in personaList: #Con este for especificamos la ruta en donde se van a leer las imagenes.
        personaPath = dataPath + '/' + nameDir
        dispatcher.bot.sendMessage(chat_id, text=f'Leyendo las imagenes de: {nameDir}')

        for fileName in os.listdir(personaPath): # #En este for lo que hacemos es leer cada uno de los rostros.
            labels.append(label) #Ahora almacenamos los rostros con sus respectivas etiquetas.#En labels añadiremos lo que seria la etiqueta.
            facesData.append(cv2.imread(personaPath + '/' + fileName,0)) #en facesData añadiremos a cada una de las imagenes a escala de grises.
            #cv2.imread() El primer argumento es el nombre de la imagen, El segundo argumento es una bandera que especifica la forma en que se debe leer la imagen.
            #El segundo argumento en 0 (cero) especifica que la imagen se leerá en modo de escala de grises.

        label = label + 1 #Ahora incrementamos el valor label para que asigne valores, 0 a la primera carpeta, 1 a la segunda y asi sucesivamente.

    face_recognizer = cv2.face.LBPHFaceRecognizer_create() #Método para entrenar el reconocedor
    dispatcher.bot.sendMessage(chat_id, 'Entrenando\u203C\uFE0F')
    face_recognizer.train(facesData, np.array(labels)) #Entrenando el reconocedor de rostros
    face_recognizer.write('modeloLBPHFace.xml') #Almacenando el modelo obtenido
    dispatcher.bot.sendMessage(chat_id, text=f"\U0001f601 Modelo almacenado! Listo para poder ingresar por Registro Facial las siguientes personas:\n {personaList}")
```

*Figura 58 – Función de entrenamiento cara*

La función "entrenamiento(dataPath)" se encarga de leer todas las imágenes de rostros almacenadas en las carpetas correspondientes, asignar etiquetas a cada imagen y crear dos matrices (una para las etiquetas y otra para los datos de los rostros en escala de grises). Luego, se utiliza el método de reconocimiento LBPH (Local Binary Patterns Histograms) para entrenar un modelo de reconocimiento facial y se guarda en un archivo XML llamado "modeloLBPHFace.xml". Finalmente, se envía un mensaje a Telegram para indicar que el proceso ha terminado y que el modelo está listo para su uso para el ingreso por Registro Facial.

En resumen, se muestra una imagen modelo del procedimiento que hace el programa:

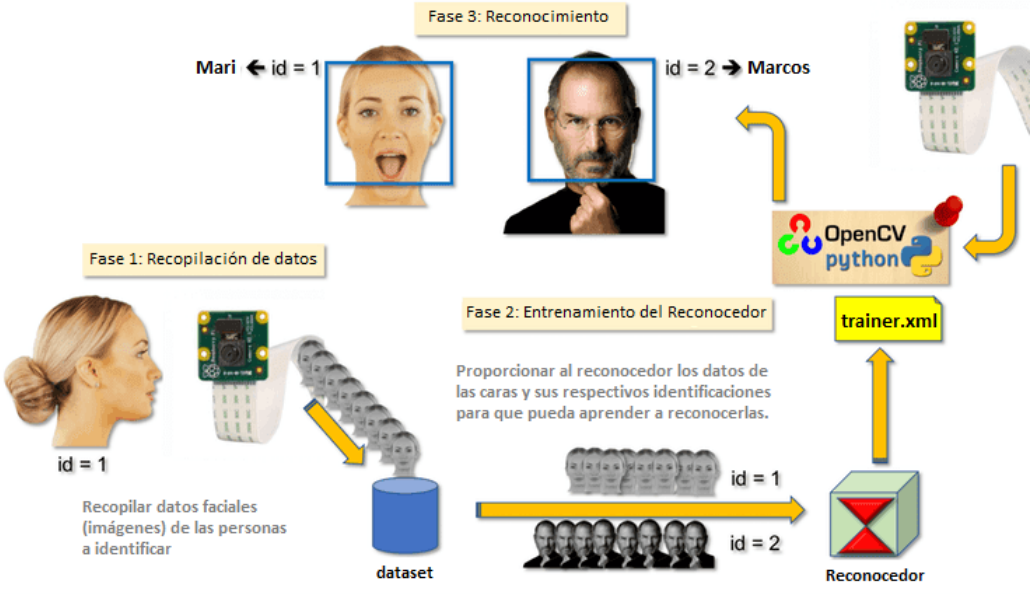



*Figura 59 – Procedimiento Reconocimiento Facial*

Funcionamiento en Bot de Telegram:

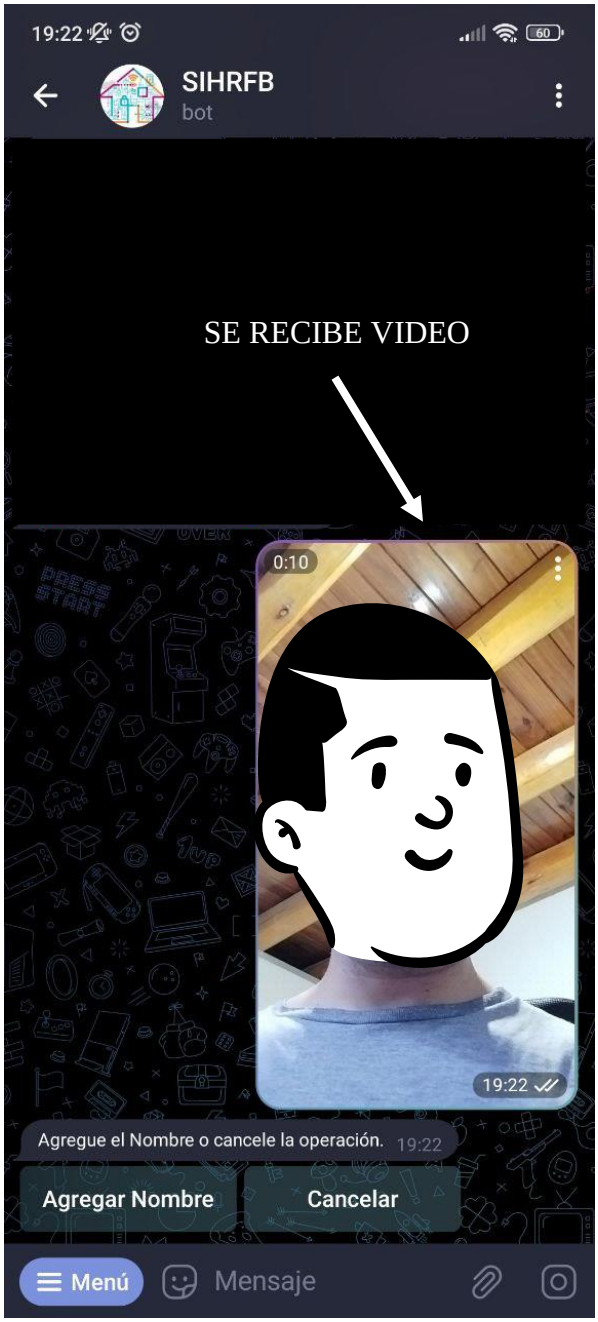



*Figura 60: Al recibir un Video, el bot detecta automáticamente que es para registrar una cara, por lo que extiende una botonera para consultar la operación que desea realizar el Usuario.*


*Figura 61: Cuando el Usuario aprieta el botón "Agregar Nombre", el mensaje se modifica indicándole que debe escribir el nombre de la persona a registrar, una vez hecho esto, el sistema muestra el proceso por mensajes del registro facial. Cuando termina, le avisa que personas puede ingresar aportándole la lista.*

### 5.1.22 Función del comando para iniciar el reconocimiento facial - /RFacial

Esta parte del programa es una implementación de un sistema de reconocimiento facial que utiliza una cámara para capturar imágenes de las caras de las personas y compararlas con las imágenes almacenadas en la carpeta "Data". En la siguiente figura se observa el código que se encarga de manejar este comando:

```
#---------------------------------------Funcion para ejectuar el reconocimiento facial----------------
def Registro_Facial(update, context):
    global chat_id
    dataPath = '/home/david/facial/Data' #Carpeta donde se almacenan los rostros, que seria en Data
    imagePaths = os.listdir(dataPath)
    face_recognizer = cv2.face.LBPHFaceRecognizer_create()
    face_recognizer.read('modeloLBPHFace.xml')
    cap = cv2.VideoCapture(0)
    count = 0
    faceClassif = cv2.CascadeClassifier(cv2.data.haarcascades+'haarcascade_frontalface_default.xml')
    dispatcher.bot.sendMessage(chat_id, 'Sistema de Reconocimiento Facial listo. Acerquese a la camara')
    while True:
        ret,frame = cap.read()
        frame = cv2.rotate(frame, cv2.ROTATE_180)
        if ret == False: break
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        auxFrame = gray.copy()

        faces = faceClassif.detectMultiScale(gray,1.3,5)

        for (x,y,w,h) in faces:
            rostro = auxFrame[y:y+h,x:x+w]
            rostro = cv2.resize(rostro,(150,150),interpolation= cv2.INTER_CUBIC)
            result = face_recognizer.predict(rostro)

            if result[1] < 80:
                nombre = '{}'.format(imagePaths[result[0]])
                dispatcher.bot.sendMessage(chat_id, f'Persona Reconococida: {nombre}')
                relay = gpiozero.OutputDevice(RELAY_PIN, active_high=False, initial_value=False)
                relay.on() # Activa el RELE una vez detectada la cara
                dispatcher.bot.sendMessage(chat_id, "Rele de puerta en estado: Encendido")
                sleep(5) # Espera de 5 segundos
                relay.off() # Apaga el RELE
                dispatcher.bot.sendMessage(chat_id, "Rele de puerta en estado: Apagado")
                cap.release()
                cv2.destroyAllWindows()
            else:
                dispatcher.bot.sendMessage(chat_id, 'No se reconoce la cara, intentelo de nuevo!')
                count = count + 1
                if count == 15:
                    dispatcher.bot.sendMessage(chat_id, 'Se acabaron los intentos sistema apagado.')
                    cap.release()
                    cv2.destroyAllWindows()
    #dispatcher.bot.sendMessage(chat_id, 'APAGADO')
    cap.release()
```

*Figura 62 – Función Registro facial*

A continuación, se establece la ruta de la carpeta donde se almacenan los rostros y se guardan en una lista llamada "imagePaths". Luego se crea un objeto "face_recognizer" utilizando el algoritmo LBPH para el reconocimiento facial y se lee el archivo XML del modelo previamente entrenado.

Luego se inicializa la cámara con "cap = cv2.VideoCapture(0)" y se establece un bucle while infinito que se encarga de capturar continuamente imágenes de la cámara.

Dentro del bucle, se lee el siguiente marco con "ret,frame = cap.read()" y se gira la imagen en 180 grados utilizando "frame = cv2.rotate(frame, cv2.ROTATE_180)".

A continuación, se convierte la imagen a escala de grises utilizando "gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)" y se crea una copia de la imagen en escala de grises con "auxFrame = gray.copy()".

Luego, se detectan las caras en la imagen utilizando un clasificador de Haar con "faces = faceClassif.detectMultiScale(gray,1.3,5)" y se recorren todas las caras detectadas en un bucle for.

Para cada cara, se recorta la región de interés utilizando "rostro = auxFrame[y:y+h,x:x+w]" y se redimensiona a un tamaño de 150x150 píxeles utilizando "rostro = cv2.resize(rostro,(150,150),interpolation= cv2.INTER_CUBIC)".

A continuación, se utiliza el modelo de reconocimiento facial previamente entrenado para predecir la identidad de la persona en la imagen con "result = face_recognizer.predict(rostro)". Si la distancia de la predicción es menor que 80, se asume que la persona ha sido reconocida correctamente y activando un relé y enviándole un mensaje de confirmación al usuario.

Si la distancia de la predicción es mayor que 80, se asume que la persona no ha sido reconocida y se envía un mensaje de error. El bucle while continuará hasta que el usuario sea reconocido o hasta que se hayan realizado 15 intentos fallidos, momento en el cual se apagará el sistema y se liberará la cámara.

Funcionamiento en Bot de Telegram:

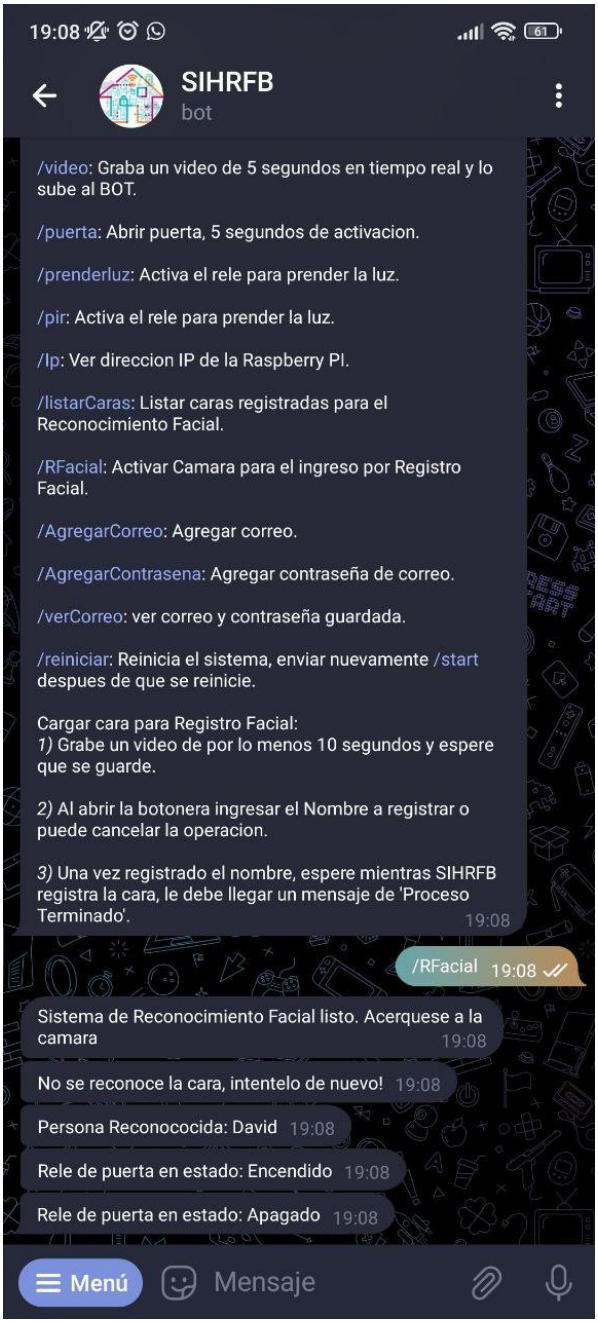



*Figura 63: Reconocimiento facial activado, mostrando el proceso de que cuando reconoce la cara, dice el nombre de la persona que reconoció y habilita la apertura del relé para poder ingresar a la propiedad o donde se desee.*

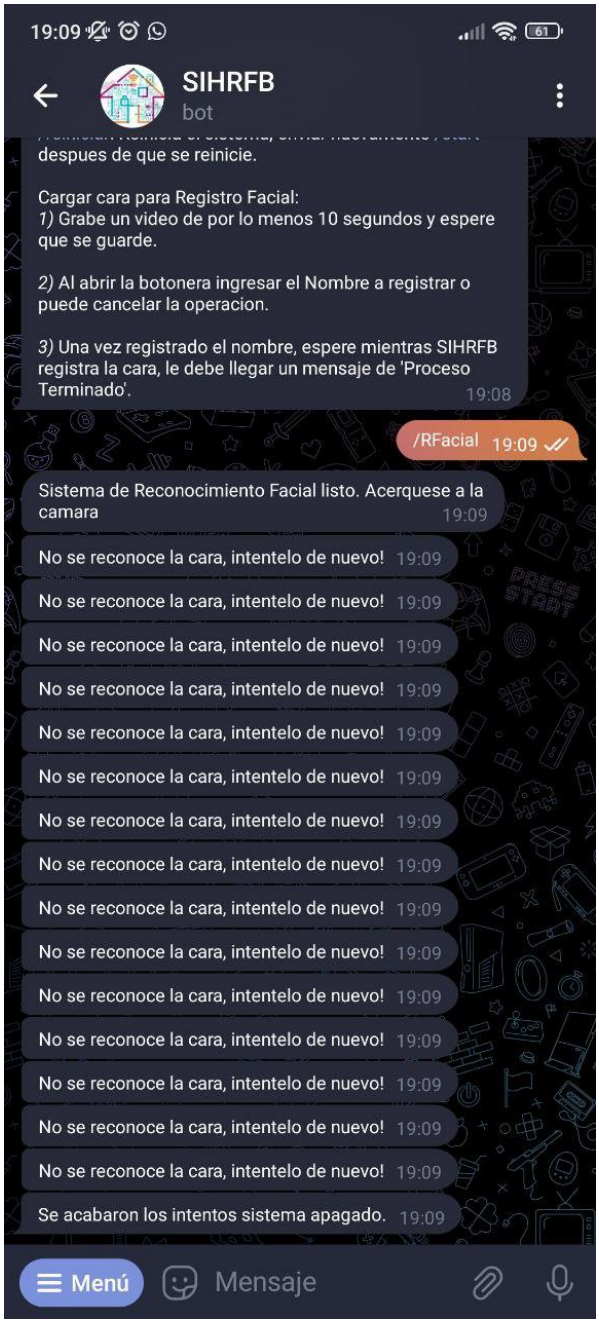



*Figura 64: Reconocimiento facial activado, mostrando el proceso de que cuando no se reconoce la cara, el cual después de 15 intentos fallidos el sistema se da por vencido y apaga el sistema para poder entrar por Reconocimiento Facial.*

### 5.1.23 Función para listar las caras guardadas para el reconocimiento facial

Esta parte del programa, lo que hace es listar las caras registradas para el ingreso por registro facial. A continuación, se muestra un flujograma al recibir este comando:

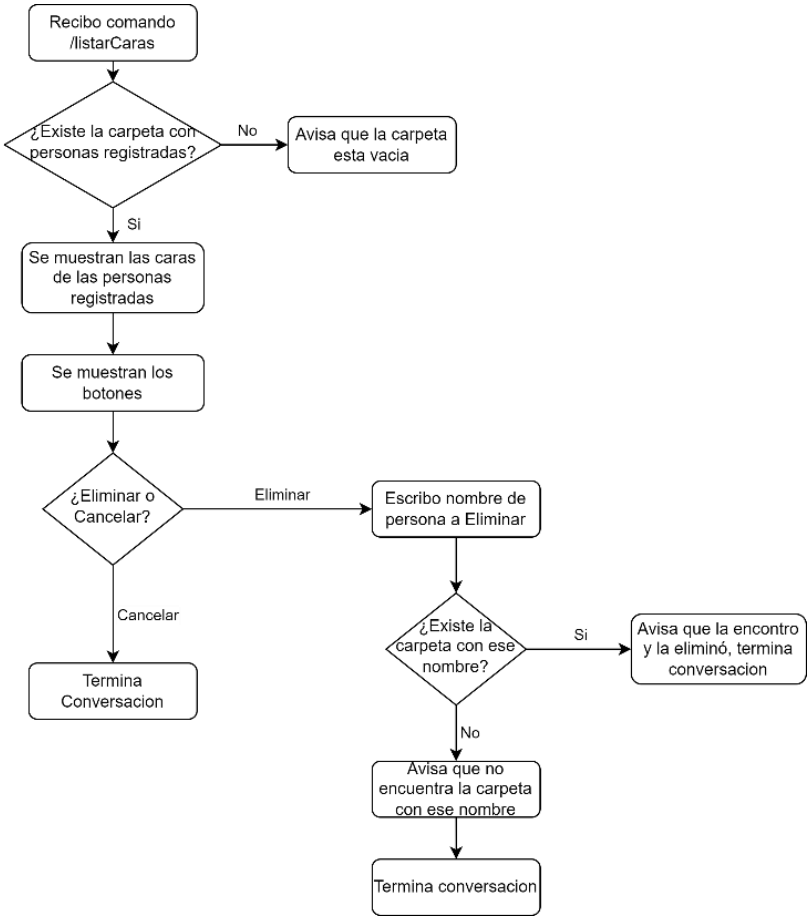



*Figura 65 – Diagrama flujo al enviar el comando de listar caras*

En la siguiente figura se muestra el código del programa:

```
#---------------------------------------Funcion listar caras registradas-----------------------------------------
def lista_cara(update,context):
    dataPath = '/home/david/facial/Data'
    personaList = os.listdir(dataPath)
    chat_id = update.message.chat_id
    if os.listdir(dataPath):
        context.bot.sendMessage(chat_id, text=f'Listando personas: \n {personaList}')
        button1 = InlineKeyboardButton(text='Si',callback_data='eliminarNombre')
        button2 = InlineKeyboardButton(text='No',callback_data='cancelar')

        update.message.reply_text(text='¿Quiere eliminar alguna persona?', reply_markup=InlineKeyboardMarkup([
            [button1, button2]])
        )
    else:
        context.bot.sendMessage(chat_id, text="No se encuentran personas registradas por el momento.")

#-------------------------------------------------Funciones para eliminar caras registradas-------------------------------------
def eliminar_persona(update, context):
    query = update.callback_query
    query.answer()
    query.edit_message_text(text="Escriba el nombre de la persona a eliminar:")
    return INPUT_TEXT2

def input_text2(update, context):
    nombre = update.message.text
    dataPath = '/home/david/facial/Data'
    personaPath = dataPath + '/' + nombre
    if os.path.exists(personaPath):
        dispatcher.bot.sendMessage(chat_id, text=f'Carpeta con el Nombre {nombre} econtrado y eliminado.\u2705')
        command = "rm -r " + personaPath
        call([command], shell=True)
    else:
        dispatcher.bot.sendMessage(chat_id, text=f'No se encuentra el nombre de: <ins>{nombre}</ins>.\u203C\uFE0F', parse_mode="HTML")

    return ConversationHandler.END
```

*Figura 66 – funciones listar cara y eliminar persona*

Lo que hace esta parte del programa es que al recibir el comando “/listarCaras”, obtiene una lista de los nombres de archivos en el directorio “dataPath” (Estos nombres de archivos representan las personas registradas en el sistema de reconocimiento facial). Se hace la comprobación para saber si el directorio donde se encuentra registrados los nombres se encuentra vacío o no, en caso de que se encuentre vacío, envía un mensaje al bot avisando de que no hay nombres registrados, caso contrario lista las personas registradas.

A continuación, crea dos botones interactivos "Sí" y "No" utilizando “InlineKeyboardButton” y los agrega a un objeto “InlineKeyboardMarkup”. Esto permite al usuario seleccionar si desea eliminar una persona registrada o no. En caso de que aprete “Sí” paso a mi función “eliminar_persona()”, (esta función se utiliza para manejar la interacción cuando el usuario selecciona la opción de eliminar una persona registrada)en esta parte edita el mensaje anterior con el texto "Escriba el nombre de la persona a eliminar:". Una vez que obtiene el nombre de la persona ingresado por el usuario, se verifica si existe esa persona en el directorio, en caso de que se encuentre, lo elimina, caso contrario, le envía un mensaje avisándole que no existe tal persona (en ambos casos termina la conversación).

Funcionamiento en Bot de Telegram:

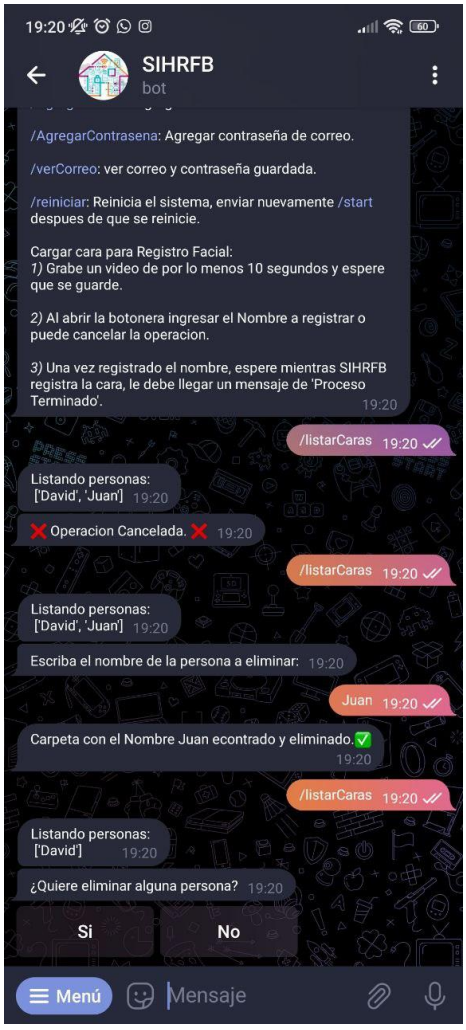



*Figura 67 – Proceso al enviar el comando /listarCaras*

### 5.1.24 Módulo de Alimentación

En el módulo de alimentación se utilizó una fuente de alimentación de 12V – 3A en conjunto con un módulo Step-down/Step-up XL6009 (porque ya los tenía) para realizar el ajuste a la tensión de alimentación de la placa Raspberry Pi y la alimentación a los relés.

*Figura 68 – Fuente de alimentación y Step-down/Step-up XL6009*

### 5.1.25 Armado del prototipo

Para el armado del prototipo se utilizó la aplicación Autodesk Fusion 360, en donde se armó a escala real para poder exportarlo como archivo ".stl" para poder imprimirlo en una impresora 3D. A continuación, se muestra el prototipo en Autodesk Fusion 360 y como quedo después de pasarlo por una impresora 3D.

*Figura 69 – Logo Autodesk Fusion 360*

Prototipo en computadora:


*Figura 70 – Prototipo diseñado, vista en cara frontal*

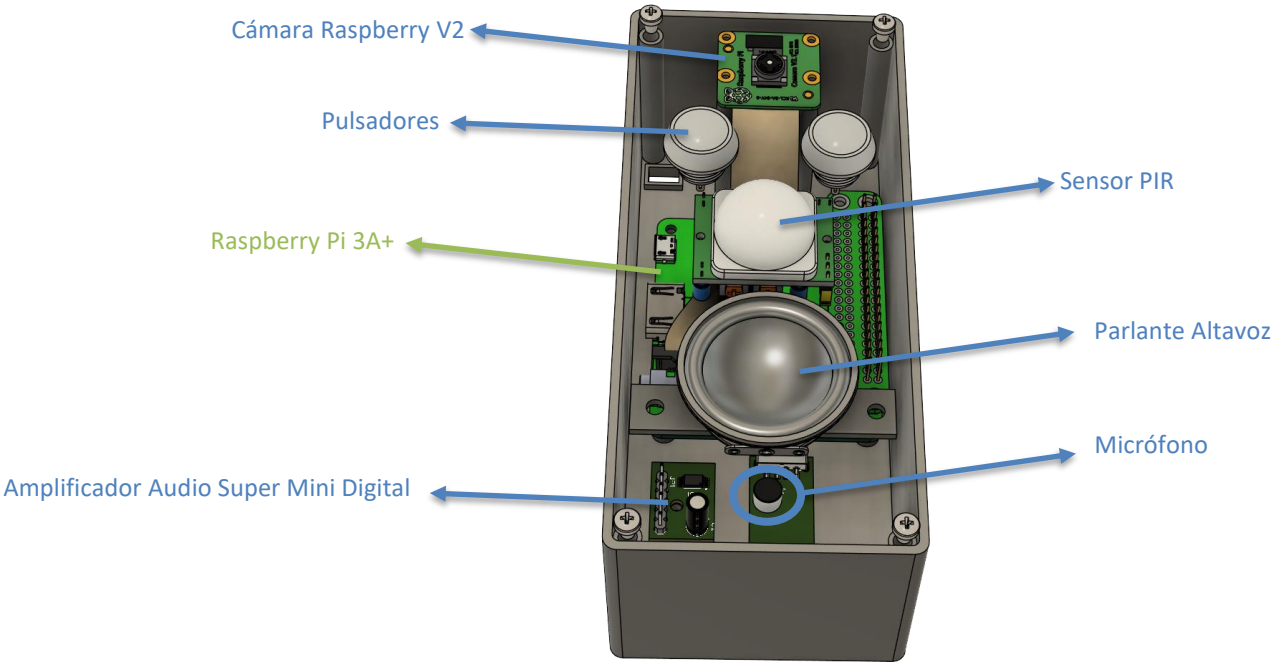



*Figura 71 – Prototipo diseñado, Componentes internos*

Prototipo trasladado a la fabricación:

*Figura 72 – Prototipo armado*

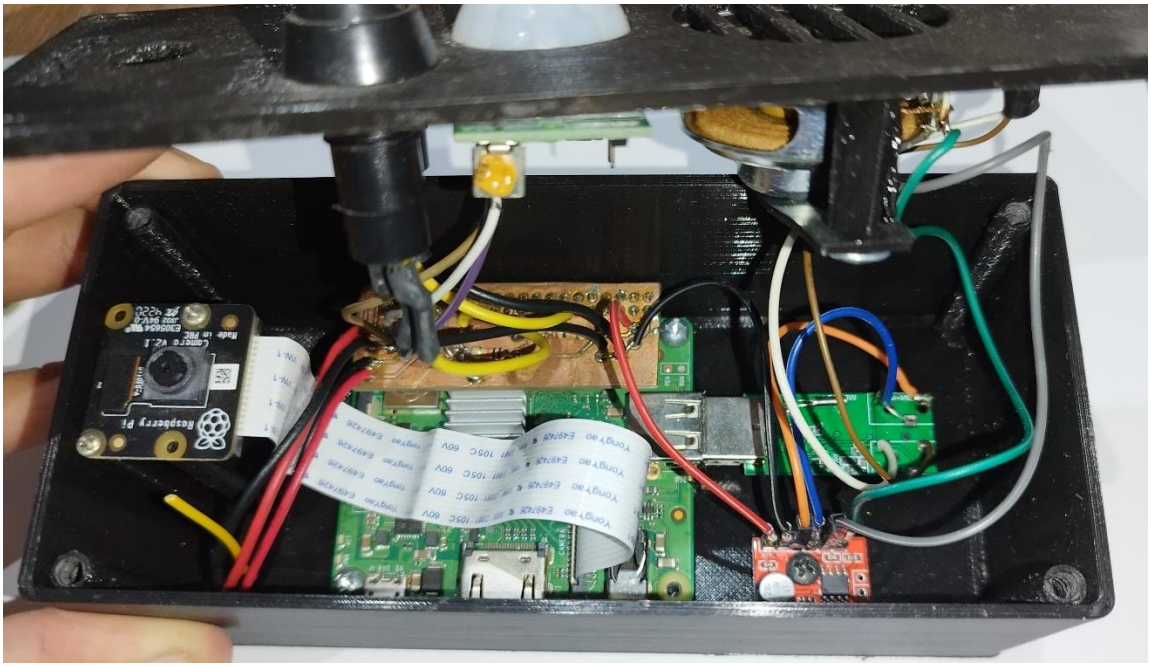


*Figura 73 – Componentes internos y armado*

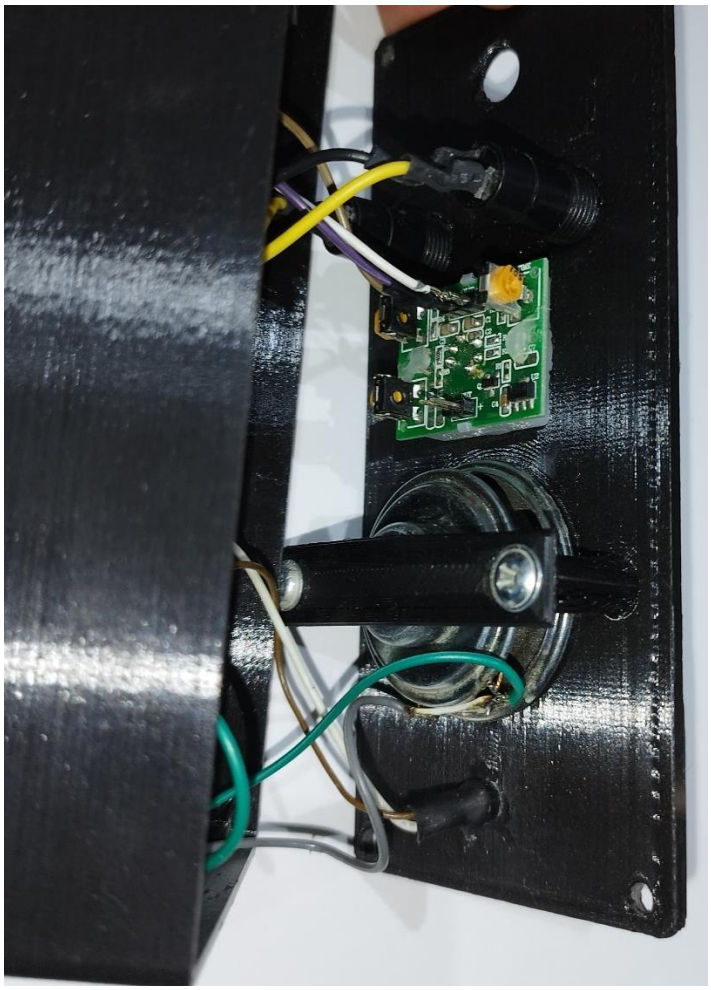


*Figura 74 – Componentes internos y armado*

Prototipo alimentación en computadora:

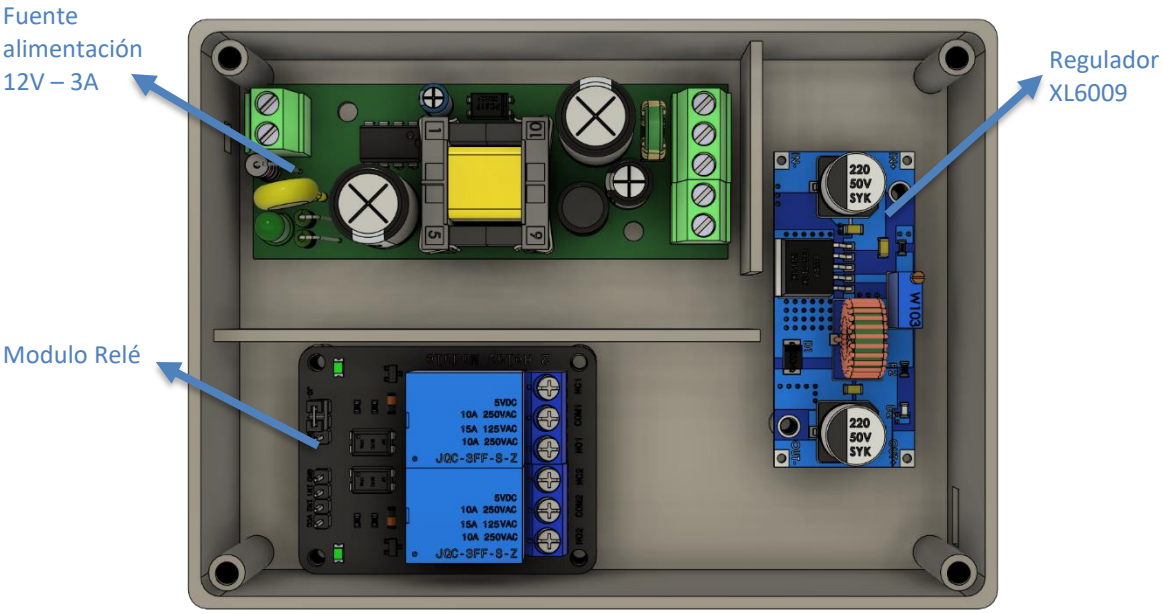



*Figura 75 – Prototipo alimentación diseñado, componentes internos*

Prototipo trasladado a la fabricación:


*Figura 76 – Componentes internos y armado*

**PROTOTIPÓ COMPLETO:**

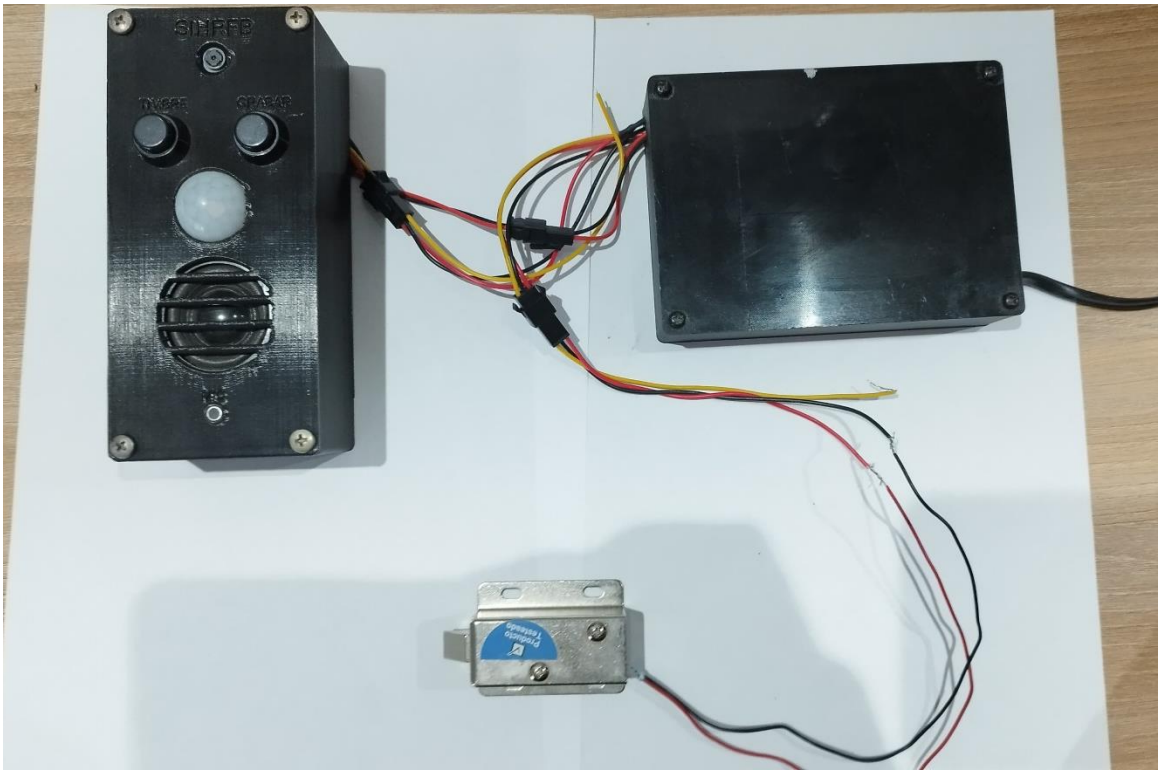



*Figura 77 – Prototipo armado completamente y conectado*

## 5.2 FACTIBILIDAD ECONÓMICA

Para el cálculo de la factibilidad económica, se tuvieron los siguientes supuestos:

- Se considerará un horizonte de evaluación de 5 años.
- El producto estará compuesto por 1 sistema SIHRFB y la fuente de alimentación.
- El precio de venta del producto es de USD.
- Dentro de los costos fijos se consideran los servicios básicos (Luz, Agua, etc.) y alquiler.
- Todo lo referido a marketing, ventas, etc. es realizado por la persona a cargo del proyecto.
- Se cobrará el 100% del precio del producto al realizar la compra.
- El equipo se produce en el mismo mes donde se ejecuta la compra.
- Para el cálculo se toma una tasa de interés del 12%.

### 5.2.1 Flujo de caja para VAN = 0

| Flujo de caja | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Unidades vendidas | | 15 | 25 | 30 | 30 | 30 |
| **Ingreso** | | $ 3.390 | $ 5.650 | $ 6.780 | $ 6.780 | $ 6.780 |
| Precio unitario | | $ 226 | $ 226 | $ 226 | $ 226 | $ 226 |
| **Costos fijos** | | | | | | |
| Total costos fijos | | $ 2.400 | $ 2.400 | $ 2.400 | $ 2.400 | $ 2.400 |
| **Costos variables** | | | | | | |
| Costo de venta del producto | | $ 1.890 | $ 126 | $ 126 | $ 126 | $ 126 |
| Herramientas e instrumentos | $ 40 | $ - | $ - | $ - | $ - | $ - |
| Equipos electrónicos | $ 220 | $ - | $ - | - | $ - | $ - |
| **Costos totales** | | $ 4.290 | $ 2.526 | $ 2.526 | $ 2.526 | $ 2.526 |
| Depreciación herramientas | | $ 81 | $ 81 | $ 81 | $ 81 | $ 81 |
| Utilidad antes de impuestos | | $ -3.381 | $ 643 | $ 1.773 | $ 1.773 | $ 1.773 |
| Impuesto a las ganancias (30%) | | $ -1.014 | $ 193 | $ 532 | $ 532 | $ 532 |
| Utilidad después de impuestos | | $ -2.367 | $ 450 | $ 1.241 | $ 1.241 | $ 1.241 |
| Depreciación herramientas | | $ 81 | $ 81 | $ 81 | $ 81 | $ 81 |
| **Inversión inicial** | | | | | | |
| Herramientas y equipos electrónicos | $ -260 | | | | | |
| Capital de trabajo | $ -1.080 | | | | | |
| Recuperación capital de trabajo | | | | | | $ 1.080 |
| Valor de desecho | | | | | | $ - |
| **Flujo del proyecto** | **$ -1.340** | **$ -2.286** | **$ 531** | **$ 1.322** | **$ 1.322** | **$ 2.402** |

*Figura 78 – Flujo de caja para VAN=0, VAN y TIR*

Para el cálculo de VAN = 0, el precio de venta del producto fue de 226 USD. La cantidad de ventas realizadas para el primer año fue de 15 productos, para el año 2 al 25 y para el año 3 a 5 de 30 productos.

### 5.2.2 Capital de trabajo para VAN = 0

| INGRESO MENSUAL AÑO UNO | | Enero | Febrero | Marzo | Abril | Mayo | Junio | Julio | Agosto | Septiembre | Octubre | Noviembre | Diciembre |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Precio de venta** | | $226,00 | $226,00 | $226,00 | $226,00 | $226,00 | $226,00 | $226,00 | $226,00 | $226,00 | $226,00 | $226,00 | $226,00 |
| Unidades vendidas | | 0 | 3 | 3 | 3 | 1 | 2 | 2 | 1 | 0 | 0 | 0 | 0 |
| **Total de ventas** | | $0,00 | $678,00 | $678,00 | $678,00 | $226,00 | $452,00 | $452,00 | $226,00 | $0,00 | $0,00 | $0,00 | $0,00 |
| **Realización del cobro** | | | | | | | | | | | | | |
| Cobro al momento de la compra | 100% | $0,00 | $678,00 | $678,00 | $678,00 | $226,00 | $452,00 | $452,00 | $226,00 | $0,00 | $0,00 | $0,00 | $0,00 |
| **Ingreso total** | | **0** | **678** | **678** | **678** | **226** | **452** | **452** | **226** | **0** | **0** | **0** | **0** |
| **PRODUCCIÓN** | | | | | | | | | | | | | |
| Stock | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Ventas | | 0 | 3 | 3 | 3 | 1 | 2 | 2 | 1 | 0 | 0 | 0 | 0 |
| Producción Mensual | | 0 | 3 | 3 | 3 | 1 | 2 | 2 | 1 | 0 | 0 | 0 | 0 |
| **EGRESOS** | | | | | | | | | | | | | |
| Costos Fijos | 1 | $200,00 | $200,00 | $200,00 | $200,00 | $200,00 | $200,00 | $200,00 | $200,00 | $200,00 | $200,00 | $200,00 | $200,00 |
| Costos de Fabricación | | $0,00 | $378,00 | $378,00 | $378,00 | $126,00 | $252,00 | $252,00 | $126,00 | $0,00 | $0,00 | $0,00 | $0,00 |
| **Egreso total** | | **$200,00** | **$578,00** | **$578,00** | **$578,00** | **$326,00** | **$452,00** | **$452,00** | **$326,00** | **$200,00** | **$200,00** | **$200,00** | **$200,00** |
| **CAPITAL DE TRABAJO** | | | | | | | | | | | | | |
| Ingresos | | 0 | 678 | 678 | 678 | 226 | 452 | 452 | 226 | 0 | 0 | 0 | 0 |
| Egresos | | $200,00 | $578,00 | $578,00 | $578,00 | $326,00 | $452,00 | $452,00 | $326,00 | $200,00 | $200,00 | $200,00 | $200,00 |
| Acumulado | | -$200,00 | -$100,00 | $0,00 | $100,00 | $0,00 | $0,00 | $0,00 | -$100,00 | -$300,00 | -$500,00 | -$700,00 | -$900,00 |
| **Máximo Acumulado** | | **-$900,00** | | | | | | | | | | | |

| | | |
|---|---|---|
| **CAPITAL DE TRABAJO SIN MARGEN** | | **$900,00** |
| **CAPITAL DE TRABAJO CON MARGEN** | **1,2** | **$1.080,00** |

*Figura 79 – Capital de trabajo para VAN=0*

### 5.2.3 Cálculo para VAN > 0

#### 5.2.3.1 Fujo de caja para VAN > 0

Para el cálculo realizado en VAN>0, el precio de venta del producto fue de 226 USD para cada año. La cantidad de ventas realizadas para el primer año fue de 60 productos, para el año 2 al 5 fue de 70 productos.

| Flujo de caja | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Unidades vendidas | | 60 | 70 | 70 | 70 | 70 |
| **Ingreso** | | $ 13.560 | $ 15.820 | $ 15.820 | $ 15.820 | $ 15.820 |
| Precio unitario | | $ 226 | $ 226 | $ 226 | $ 226 | $ 226 |
| **Costos fijos** | | | | | | |
| Total costos fijos | | $ 2.400 | $ 2.400 | $ 2.400 | $ 2.400 | $ 2.400 |
| **Costos variables** | | | | | | |
| Costo de venta del producto | | $ 7.560 | $ 8.820 | $ 8.820 | $ 8.820 | $ 8.820 |
| Herramientas e instrumentos | $ 40 | $ - | $ - | $ - | $ - | $ - |
| Equipos electrónicos | $ 220 | $ - | $ - | - | $ - | $ - |
| **Costos totales** | | $ 9.960 | $ 11.220 | $ 11.220 | $ 11.220 | $ 11.220 |
| Depreciación herramientas | | $ 81 | $ 81 | $ 81 | $ 81 | $ 81 |
| Utilidad antes de impuestos | | $ 1.119 | $ 2.119 | $ 2.119 | $ 2.119 | $ 2.119 |
| Impuesto a las ganancias (30%) | | $ 336 | $ 636 | $ 636 | $ 636 | $ 636 |
| Utilidad después de impuestos | | $ 783 | $ 1.483 | $ 1.483 | $ 1.483 | $ 1.483 |
| Depreciación herramientas | | $ 81 | $ 81 | $ 81 | $ 81 | $ 81 |
| **Inversión inicial** | | | | | | |
| Herramientas y equipos electrónicos | $ -260 | | | | | |
| Capital de trabajo | $ -240 | | | | | |
| Recuperación capital de trabajo | | | | | | $ 240 |
| Valor de desecho | | | | | | $ - |
| **Flujo del proyecto** | **$ -500** | **$ 864** | **$ 1.564** | **$ 1.564** | **$ 1.564** | **$ 1.804** |

*Figura 80 – Flujo de caja para VAN>0*

#### 5.2.3.2 Capital de trabajo para VAN > 0

| INGRESO MENSUAL AÑO UNO | | Enero | Febrero | Marzo | Abril | Mayo | Junio | Julio | Agosto | Septiembre | Octubre | Noviembre | Diciembre |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Precio de venta** | | $226,00 | $226,00 | $226,00 | $226,00 | $226,00 | $226,00 | $226,00 | $226,00 | $226,00 | $226,00 | $226,00 | $226,00 |
| Unidades vendidas | | 0 | 10 | 10 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 0 |
| **Total de ventas** | | $0,00 | $2.260,00 | $2.260,00 | $1.130,00 | $1.130,00 | $1.130,00 | $1.130,00 | $1.130,00 | $1.130,00 | $1.130,00 | $1.130,00 | $0,00 |
| **Realización del cobro** | | | | | | | | | | | | | |
| Cobro al momento de la compra | 100% | $0,00 | $2.260,00 | $2.260,00 | $1.130,00 | $1.130,00 | $1.130,00 | $1.130,00 | $1.130,00 | $1.130,00 | $1.130,00 | $1.130,00 | $0,00 |
| Ingreso total | | 0 | 2260 | 2260 | 1130 | 1130 | 1130 | 1130 | 1130 | 1130 | 1130 | 1130 | 0 |
| **PRODUCCIÓN** | | | | | | | | | | | | | |
| Stock | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Ventas | | 0 | 10 | 10 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 0 |
| Producción Mensual | | 0 | 10 | 10 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 0 |
| **EGRESOS** | | | | | | | | | | | | | |
| Costos Fijos | 1 | $200,00 | $200,00 | $200,00 | $200,00 | $200,00 | $200,00 | $200,00 | $200,00 | $200,00 | $200,00 | $200,00 | $200,00 |
| Costos de Fabricación | | $0,00 | $1.260,00 | $1.260,00 | $630,00 | $630,00 | $630,00 | $630,00 | $630,00 | $630,00 | $630,00 | $630,00 | $0,00 |
| Egreso total | | $200,00 | $1.460,00 | $1.460,00 | $830,00 | $830,00 | $830,00 | $830,00 | $830,00 | $830,00 | $830,00 | $830,00 | $200,00 |
| **CAPITAL DE TRABAJO** | | | | | | | | | | | | | |
| Ingresos | | 0 | 2260 | 2260 | 1130 | 1130 | 1130 | 1130 | 1130 | 1130 | 1130 | 1130 | 0 |
| Egresos | | $200,00 | $1.460,00 | $1.460,00 | $830,00 | $830,00 | $830,00 | $830,00 | $830,00 | $830,00 | $830,00 | $830,00 | $200,00 |
| Acumulado | | -$200,00 | $600,00 | $1.400,00 | $1.700,00 | $2.000,00 | $2.300,00 | $2.600,00 | $2.900,00 | $3.200,00 | $3.500,00 | $3.800,00 | $3.600,00 |
| **Máximo Acumulado** | | **-$200,00** | | | | | | | | | | | |

| | | |
|---|---|---|
| **CAPITAL DE TRABAJO SIN MARGEN** | | **$200,00** |
| **CAPITAL DE TRABAJO CON MARGEN** | 1,2 | **$240,00** |

*Figura 81 – Capital de trabajo para VAN>0*

#### 5.2.3.3 Aproximación al valor actual neto

| | |
|---|---|
| Tasa de interés | 12% |
| **VAN** | **$ 4.650** |

*Figura 82 – Calculo de VAN>0*

### 5.2.4 Tasa interna de retorno

| | |
|---|---|
| **TIR** | **216%** |

*Figura 83 – Calculo de TIR para VAN>0*

### 5.2.5 Payback o plazo de recuperación

| PRI | | |
|---|---|---|
| PERIODO | FLUJO | ACUMULADO |
| 0 | $ -500 | |
| 1 | $ 864 | $ 864 |
| 2 | $ 1.564 | $ 2.429 |
| 3 | $ 1.564 | $ 3.993 |
| 4 | $ 1.564 | $ 5.558 |
| 5 | $ 1.804 | $ 7.362 |
| | | |
| **PRI** | **0,77 años** | |

*Figura 84 – Playback para VAN>0*

### 5.2.6 Productos y servicios de otros fabricantes

Como se sabe, hay varios productos de similares prestaciones a lo que se plantea para este proyecto, pero ninguno tiene todas las características ya mencionadas integradas en una sola placa. Ya que nuestros principales competidores son internacionales de fabricación china debido a sus precios, se va a tener en cuenta que nuestro producto va a tener una calidad única, por lo cual se estima que el sistema no falle en años, además se va a tener atención personalizada, tanto presencial como virtual y otro plus que se va a tener en cuenta que van a haber técnicos capacitados para cualquier problema que presente este producto.

La competencia hoy en día solamente cuenta con el sistema de timbre inteligente, el cual tiene sensores de presencia y aviso a una aplicación que una empresa propia crea, en nuestro caso vamos a usar una aplicación de uso diario y gratuito.

Hoy en día la información sobre esta solución al problema que se plantea no se encuentra muy bien informada, esto quiere decir, que se pretende hacer publicidad vía redes sociales, televisivos y en radios para conocer el producto y las prestaciones que ofrece, lo cual lo va a hacer único comparado a otros productos.

Se compara algunos productos de otros competidores:

- Xiaomi Smart Doorbell 3, con un costo de 250 dólares, posee características similares, pero sin la utilidad de reconocimiento facial.

*Figura 85 – Producto de la empresa Xiaomi*

- BOTSLAB 2K video dorbell, con un costo de 1100 dólares, este producto es el que actualmente se encuentra en condición completa para competir con todas las características que posee nuestro prototipo.


*Figura 86 – Producto de la empresa Botslab*

Comparando nuestro prototipo con los productos anteriores, podemos considerarlo rentable desde el punto que estamos ofreciendo un equipo por un costo de 226 USD. Esto considerando que el producto puede adaptarse con distintos sensores y especificaciones según lo que requiera el cliente.

# 6 CONCLUSIONES Y ANEXOS

Se puede concluir, a la luz de los resultados, que las tecnologías IoT son capaces de aportar flexibilidad y versatilidad a un sistema de seguridad sencillo a la hora de implementarlo en la práctica.

El hecho de que el prototipo del sistema esté programado íntegramente en Python, demuestra la potencia y relativa facilidad que caracteriza este lenguaje. Librerías tan potentes como OpenCV, usada para proyectos de visión artificial, o scikit-learn, usada para el aprendizaje automático en el área de la inteligencia artificial, se pueden implementar fácilmente en Python. Esta facilidad, en parte, viene de las comunidades activas que hay detrás del lenguaje y de muchas de estas librerías, haciendo que la búsqueda de información sea más rápida y eficiente.

# 7 BIBLIOGRAFÍAS Y REFERENCIAS BIBLIOGRÁFICAS


*[1] The official documentation for Raspberry Pi computers and microcontrollers (01/01/2009). https://www.raspberrypi.com/documentation/*

*[2] python-telegram-bot - Esta biblioteca proporciona una interfaz asincrónica pura de Python para la API de Telegram Bot. (01/03/2015) https://docs.python-telegram-bot.org/en/v21.0.1/*

*[3] Bots: una introducción para desarrolladores. (14/08/2013) https://core.telegram.org/bots*

*[4] Documentación sobre gpiozero. (20/01/2015) https://gpiozero.readthedocs.io/en/stable/index.html*

*[5] Contenido de la documentación de Python. (10/04/2001) https://docs.python.org/es/3/contents.html*

*[6] OpenCV (Open Source Computer Vision) - Face Recognition with OpenCV. (20/12/2019) https://docs.opencv.org/4.2.0/da/d60/tutorial_face_main.html#tutorial_face_eigenfaces*

*[7] Installing OpenCV on the Raspberry Pi. (30/09/2022) https://pimylifeup.com/raspberry-pi-opencv/*

*[8] Creating email and MIME objects from scratch. (19/06/2020) (https://docs.python.org/2/library/email.mime.html#email.mime.image.MIMEImage*

*[9] GeeksforGeeks - OpenCV Tutorial in Python.(01/02/2023) https://www.geeksforgeeks.org/opencv-python-tutorial/?ref=gcse*

*[10] PyAudio Documentation.(24/06/2006)*
*https://people.csail.mit.edu/hubert/pyaudio/docs/#class-pyaudio*