



UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES ARAGÓN

**“Desarrollo e incorporación a un sistema de
comunicación aumentativo y alternativo de un
módulo para control domótico”**

TESIS

Para obtener el título de:

Ingeniero Eléctrico Electrónico

Presenta:

Efraín Castro Carrizal

Asesor de tesis:

Dr. Ismael Díaz Rangel

Ciudad Nezahualcóyotl, Estado de México, 2023





Universidad Nacional
Autónoma de México

Dirección General de Bibliotecas de la UNAM

Biblioteca Central



UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.



AGRADECIMIENTOS

Quiero agradecer primeramente **a Dios**, el autor de vida y de eterna salvación, por la oportunidad de poder concluir esta etapa de mi vida, darme la sabiduría para poder realizar este trabajo y guiarme en cada uno de mis caminos.

A mis padres Juan Francisco Castro y María de los Ángeles Carrizal, que me han apoyado en cada una de las etapas de mi vida en todos los aspectos, me han guiado y ayudado en todos mis proyectos, además de siempre estar cuando los necesito.

A mi hermana Elisa Castro, por animarme cuando me sentía cansado, darme palabras de aliento, y por ser mi amiga incondicional en todo momento.

A mi abuelita Gloria González, por dejarme vivir con ella gran parte de mi carrera y apoyarme económica y físicamente en las tareas de casa para poder enfocarme en mis estudios.

A la Universidad Nacional Autónoma de México, Facultad de Estudios Superiores Aragón, por los profesores y las instalaciones que pusieron a mi disposición para adquirir los conocimientos necesarios para realizar esta tesis.

Al Dr. Ismael Díaz Rangel, por ser mi asesor a lo largo de todo mi proceso de titulación, darme la oportunidad de ser parte del **grupo IDEA** y trabajar en este proyecto, guiándome en cada una de las etapas.

Al Ing. Juan Manuel Hernández Contreras, por su asesoramiento, además de ser un gran profesor en la carrera que, a través de sus clases, despertó mi interés por la programación de microcontroladores y microprocesadores, así como por aprender los lenguajes de programación.

Investigación realizada gracias al Programa UNAM-PAPIIT IT103220.



ÍNDICE

AGRADECIMIENTOS	i
ÍNDICE	ii
Capítulo 1. Introducción	5
1.1 Objetivo general	6
1.2 Objetivos particulares	6
1.3 Actividades	6
1.4 Justificación	7
1.5 Antecedentes	7
1.6 Descripción del capitulado	10
Capítulo 2. Marco teórico	12
2.1 Los SAAC	12
2.1.1 Tipos de SAAC	12
2.1.2 Pacientes que requieren de los SAAC	13
2.2 Raspberry Pi	15
Raspberry Pi 4 Model B	15
2.3 ESP32	17
ESP32-WROOM-32	17
2.4 Códigos infrarrojos	18
2.4.1 Rayos infrarrojos	18
2.4.2 Controles remotos	18
2.4.3 Protocolos	19
2.4.4 Receptor KY-022	19
2.4.5 Transmisor IR383	20
2.4.6 Circuitos de control de señales	20
2.5 Interruptores	21
2.6 Dispositivos de visualización	23
2.6.1 LED	23
2.6.2 Display de segmentos	23
2.6.3 Pantalla LCD	23
2.7 Asistentes virtuales	24



2.8 Python	25
2.9 Arduino IDE	26
2.10 PCB	32
2.10.1 Software para diseño de PCB con EasyEDA	32
2.10.2 Elaboración de PCB casera	33
Capítulo 3. Desarrollo experimental	34
1.1 Módulo IR	35
1.1.1 Diagrama esquemático	35
1.1.2 Diagrama de flujo	36
1.1.3 PCB	40
1.2 Interfaz para módulo IR	41
1.2.1 Mapa de navegación para configuración de control de dispositivos (aparatos electrónicos) mediante infrarrojo	42
1.2.2 Mapa de navegación para operación	54
1.3 Interfaz para asistentes visuales	61
1.3.1 Mapa de navegación para configuración de frases para asistentes virtuales	61
1.3.2 Mapa de navegación para operación	65
Capítulo 4. Pruebas y resultados	67
4.1 Placa de prueba	67
4.2 Creación de interfaz gráfica para comunicación entre receptor IR y la Raspberry	67
4.3 Biblioteca LIRC en Raspberry PI4	71
4.4 Biblioteca IRremote en Arduino	72
4.5 Biblioteca IRLib2 en Arduino	73
4.6 Guardar códigos en la memoria EEPROM en Arduino	73
4.7 Módulo microSD con Arduino	76
4.8 Conexión Serial entre Raspberry y Arduino	77
4.9 Conexión I2C entre Raspberry y Arduino	77
4.10 Biblioteca IRLib2 en ESP32	79
4.11 Biblioteca IRremote en ESP32	80
4.12 Módulo microSD en ESP32	80
4.13 Conexión I2C entre Raspberry y ESP32	81
4.14 Conexión Bluetooth entre Raspberry y ESP32	82



4.15 Conexión USB entre Raspberry y ESP32	84
4.16 Conexión módulo IR	86
4.17 Distancia y ángulo de recepción	88
4.18 Guardado y actualización de un código	89
4.19 Transmisión de una señal	89
4.20 Distancia y ángulo de transmisión con LED IR	90
4.21 Funcionamiento dispositivos	92
4.22 Configuración de nombres de controles y botones editables	95
4.23 Configuración del tamaño de las frases para la comunicación con asistentes virtuales	96
4.24 Prueba de frases para el asistente virtual	97
4.25 Resolución del sistema	98
Conclusiones	102
Referencias	104
Fuentes de imágenes	108
Anexo	111
Anexo A. Código de ESP32	111
Anexo B. Plantillas para incorporación del módulo IR al SAAC	112
B.1 Plantillas editadas ya existentes	112
B.2 Plantillas nuevas	113
Anexo C. Plantillas para incorporación de Asistentes virtuales	114



CAPÍTULO 1. INTRODUCCIÓN

Existen alrededor de 7.9 billones de personas en el mundo (worldometer, s.f.), que están expuestas a cualquier tipo de enfermedades, traumatismos, entre otros padecimientos, dentro de esta población existe aproximadamente un 15% de gente que tiene una discapacidad, según el banco mundial, es un aproximado de 1 000 millones de personas (Grupo Banco Mundial, 2021).

Para poder crear una mejor sociedad, se han realizado diferentes planes y utilizado diversos recursos para poder integrar a este sector de la población en nuestra sociedad, de los cuales, muchos de estos son más sencillos por lo que no necesitan ayuda de una tercera persona para que las personas que sufre la discapacidad puedan desempeñar actividades. Aunque también existe un sector el cual sufre de trastornos motores o neurodegenerativos, esto les impide comunicarse, como ejemplo están las personas con esclerosis múltiple, donde 1 de cada 3 000 personas la padecen; esta enfermedad degenera el sistema motor hasta el punto de no tener control en ningún músculo del cuerpo, incluyendo la boca (Fallas Sanabria, 2010).

Otro ejemplo, son las personas que tienen esclerosis lateral amiotrófica, donde 5 de cada 100 000 personas llegan a padecer de ella; esta enfermedad degenera los sistemas motores, cerebelosos y sensitivos (Porrás Betancourt, Núñez Orozco, Plascencia Álvarez, Quiñones Aguilar, & Sauri Suárez, 2007). Otra enfermedad puede ser la parálisis cerebral, donde 3 de cada 1 000 personas tienen la enfermedad; en está, el cerebro no logró desarrollarse bien y esto tiene como consecuencia problemas motores y del habla (Malagón Valdez, 2007).

Todas estas enfermedades y muchos otros padecimientos, en un estado más avanzado o grave, tienen como consecuencia un déficit en la capacidad de la escritura y del habla, por lo que, si la persona no cuenta con la ayuda de un tercero, esta no podría realizar ningún tipo de actividad debido a la falta de la capacidad para comunicarse, lo cual disminuye su calidad de vida (esto puede presentarse en su última etapa de vida). Como parte de la inclusión y el mejoramiento de la salud de las personas con estos padecimientos, se ha optado por desarrollar dispositivos que permitan la comunicación, ya sea de baja o alta tecnología, y a estos dispositivos se les conoce como Sistemas Aumentativos y Alternativos de la Comunicación (SAAC).

Dentro de la FES Aragón, en el grupo IDEA, se desarrolló un SAAC de costo bajo/moderado (si se llega a comparar con opciones semejantes), este tiene la capacidad de formar y reproducir oraciones a través de una voz sintetizada; por lo que puede ser la forma de comunicarse para las personas con discapacidades de comunicación hablada y de escritura. El SAAC tiene como elemento central una mini computadora Raspberry Pi4 (lo cual permite tener un costo asequible); y este puede ser proyectado en cualquier tipo de pantalla o monitor, donde se muestra una interfaz gráfica que esta conformada de una serie de plantillas, sobre las cuales se muestra el abecedario o pictogramas; y un cursor (recuadro) se desplaza automáticamente sobre las opciones mostradas; el paciente debe presionar un botón o accionar cualquiera de los interruptores que este utilizando cuando el cursor esta sobre el elemento de su interés para realizar la selección, y así poder crear frases que pueden ser verbalizados por medio del sintetizador (Delgadillo Martínez, 2020). Este sistema crea la facilidad de comunicación entre el paciente y la gente de su alrededor, pero aún es necesaria una tercer persona para poder realizar algunas actividades para el paciente con respecto a los aparatos electrónicos a su



alrededor, por tal razón, este trabajo se enfoca a dotar al SAAC existente un módulo para controlar señales infrarrojas, y así, poder tener un control sobre dichos aparatos en sus funciones básicas, además de crear un menú dentro del sistema, para la interacción con asistentes virtuales (si el paciente cuenta con alguno), dotando al sistema la capacidad de reproducción de frases para activar rutinas creadas para este.

1.1 Objetivo general

Dotar a un sistema aumentativo y alternativo de comunicación la capacidad de control doméstico mediante el uso de asistentes virtuales, así como del manejo de aparatos electrónicos mediante señales infrarrojas.

1.2 Objetivos particulares

- Desarrollar un módulo para la lectura y transmisión de señales infrarrojas considerando un costo asequible sin comprometer la fiabilidad.
- Mantener las características de diseño y navegación del SAAC sobre el que se realizará la adición de funcionalidades para el control de aparatos electrónicos.
- Integrar capacidad para el aprendizaje de códigos de controles remotos mediante infrarrojo.
- Crear un módulo para la transmisión de señales infrarrojas de manera multidireccional.
- Incorporar una interfaz para interactuar con los asistentes virtuales: Alexa y Google Assistant.
- Adicionar en el SAAC plantillas orientadas para el manejo de aparatos electrónicos.

1.3 Actividades

- Investigar y comprender qué son los SAAC.
- Indagar las condiciones que condicionan a las personas a utilizar un SAAC.
- Conocer el principio de funcionamiento de controles remotos mediante señales infrarrojas.
- Estudiar los métodos de recepción, aprendizaje y transmisión de códigos infrarrojos para control de aparatos electrónicos.
- Definir e implementar el prototipo de un módulo para la recepción de códigos infrarrojos.
- Hacer un programa en Python para la recepción y almacenamiento de códigos infrarrojos.
- Desarrollo de una interfaz gráfica para la recepción y almacenamiento de múltiples códigos infrarrojos.
- Hacer un programa en Python para la transmisión de códigos infrarrojos.
- Desarrollo de una interfaz gráfica para la transmisión de códigos infrarrojos.
- Probar las distancias alcanzables del infrarrojo y realizar los ajustes necesarios.
- Definir un circuito para la transmisión de señales infrarrojas multidireccionales.
- Desarrollar una interfaz gráfica para agregar comandos para Alexa y Google Assistant.
- Incorporar las interfaces para configuración, transmisión de datos infrarrojos y asistentes virtuales al SAAC.



1.4 Justificación

Una de las metas de la sociedad, es el construir un mundo donde todos tengamos igualdad de oportunidades, no importando la edad, etnia, identidad de género, religión, color, idioma, opinión política, posición económica, discapacidad, o cualquier otra condición de la persona.

Existen personas que padecen de una condición o enfermedad que disminuye sus capacidades motrices, y los hace sufrir de manera física y/o mental; pero no solo ellos se ven afectados, sino que las personas cercanas a ellos pueden ver mermada su calidad de vida; un ejemplo de ello, son las personas que sufren de Esclerosis Lateral Amiotrófica (ELA), las cuales en algún momento de su enfermedad, pierden su capacidad de hablar y escribir, aunque estos cognitivamente se encuentran en buenas condiciones; es decir, están atrapados dentro de un cuerpo disfuncional, lo que conlleva angustia al paciente y también a la persona que debe cuidarlo, debido a la limitada comunicación.

Existen muchos otros padecimientos como el antes mencionado, por lo que trabajar en propuestas de sistemas cuyo objetivo sea mejorar la calidad de vida de estos sectores, es algo que vale la pena el esfuerzo, y más tratándose de un ambiente académico, ya que este tipo de sistemas no son de interés para corporaciones mundiales; además esto ayuda a dotar a los estudiantantes de principios y conocimientos que los forman como mejores individuos para esta sociedad actual.

1.5 Antecedentes

“Se tienen descripciones antes de 1975, sobre personas que no podían realizar una comunicación verbal, pero utilizaban tableros de letras simples, otros con imágenes y también máquinas de escribir para poder expresar algún mensaje.

En su principio, los SAAC estaban limitados para un número reducido de personas con afasia, este es un trastorno causado por lesiones en el cerebro, en las regiones que controlan el lenguaje; estas personas tenían la capacidad de poder señalar símbolos sin la necesidad de ayuda. Fue hasta el año de 1975, cuando se promulgó la Ley de Educación de Todos los Niños Discapacitados autorizada por el Congreso de los Estados Unidos (Ley pública 94-142 o PL 94-142), en donde se estipula que todos los niños entre 5 y 21 años de edad que tengan alguna discapacidad, reciban una educación pública y gratuita, ya que antes de esta, no existía una ley donde se obligara a la educación especial, por lo cual, las personas con discapacidades eran excluidas de la educación general, y debido a ella, estas personas empezaron a recibir la atención debida por parte de educadores expertos en el habla y lenguaje.

El desarrollo de las microcomputadoras fue uno de los factores que influyeron en gran manera en la creación de los SAAC, ya que antes de la existencia de estas, existían los dispositivos aumentativos y alternativos de comunicación (AAC), los cuales eran sistemas eléctricos rústicos, donde muchos de estos eran adaptaciones de máquinas de escribir eléctricas o dispositivos de escaneo electrónico, los cuales no contaban con ningún tipo de salida impresa y mucho menos hablada.

Canon Communicator fue de los primeros dispositivos de comunicación aumentativa a nivel comercial, fue diseñado por la corporación Canon, el cual funcionaba con baterías y se debía



sujetar a a la muñeca de la persona. Era de tamaño pequeño y contaba con un teclado con letras ordenadas alfabéticamente, además de poseer una memoria que almacenaba las frases y palabras que la persona realizó, contaba con la opción de hacer pequeñas impresiones a traves de una impresora de tiras integrada, pero la desventaja era que los usuarios debian contar con conocimientos de ortografía y habilidades motoras finas para poder crear frases presionando los paneles integrados.

Otro sistema fue el AutoCom, que se consideraba el dispositivo AAC portátil mas sofisticado, este contaba con 128 áreas de símbolos que podía programarse y modificarse a la necesidad del usuario, tenía un tamaño de 50x60 cm y pesaba 7.7 kg aproximadamente, contaba con una salida impresa por medio de una pequeña impresora de tiras de papel y además incluía una pantalla LED de 32 caracteres. Este dispositivo proporciono a los usuarios la opción de utilizar imágenes, pero el uso de este, estaba limitado para personas que podían tocar los símbolos con el puntero especial, lo que requería de la habilidad motora fina.

HandiVoice-110 y HandiVoice-120 fueron desarrollados por Phonic Ear en 1978, los cuales fueron los primeros SAAC portátiles a nivel comercial que contaban con salida de voz sintetizada. Este avance impulso el potencial de los SAAC portátiles con salida de voz a expandir su desarrollo por medio de las nuevas tecnologías, para poder adaptar los sistemas a las necesidades y capacidades de los diferentes tipos de usuarios” (C. Vanderheiden, 2002).

La sociedad intenta crear un mundo inclusivo, adaptando a personas con discapacidades a formar parte de las actividades cotidianas, esto a sido posible atraves de las nuevas tecnologías, la gran importancia de esto llego a tal punto que fue expuesto en uno de los eventos mundiales más grandes de tecnología, el Consumer Electronics Shows (CES), donde a partir del 2021 se abrió la puerta a los artículos tecnológicos que ayudar a mejorar la vida cotidiana de las personas con discapacidad, centrados en la accesibilidad para estas personas como teclados baille y aplicaciones que ayudan a los usuarios con discapacidad visual, debido a que, el 15% de la población mundial tiene una discapacidad, lo que significa un gran mercado de billones de dólares y actualmente la mayoría de los dispositivos son bastante costosos (Al-Heeti, 2021).

Además de que los sistemas son costosos, los dispositivos que se utilizan para controlarlos también lo son, en la página Inclusive Technology, podemos encontrar estos dispositivos. El IntegraMouse Plus es un dispositivo que detecta el soplo o succión del usuario, también puede ser una palanca de direcciones controlada por la boca y puede utilizarse como un mouse para una computadora, pero su costo es de aproximadamente un poco más de 2 000 euros (Inclusive Technology Ltd, s.f.). Otro ejemplo es el Quha Zono Mause, que es un mouse controlado por medio de un giroscopio que se activa por medio de un leve movimiento, este tiene un costo de aproximadamente 1 000 euros, por lo que podemos darnos cuenta que son dispositivos poco asequibles, lo que hace difícil la adquisición de estos dispositivos para los usuarios en Latinoamérica (Inclusive Technology Ltd, s.f.).

En la FES Aragón, en el 2016 los profesores del grupo IDEA y el alumno Giovanni Belli comenzaron el desarrollo de un SAAC. El objetivo fue crear un sistema que pueda ser considerado de alta tecnología, pero con la finalidad de tener un costo accesible para los posibles usuarios, ya que, al realizar investigaciones, se identificó un problema de que los SAAC existentes de alta tecnología son sistemas que tienen costos muy elevados.



El sistema debía ser útil para las personas que contarán con dos condiciones, tener una limitada capacidad del habla y además tener limitantes en la motricidad fina. Esta propuesta fue desarrollada en una mini computadora de costo asequible (Raspberry Pi4), el diagrama del sistema se muestra en la figura 1.1, desarrollado en un sistema operativo y con herramientas de desarrollo de uso libre.

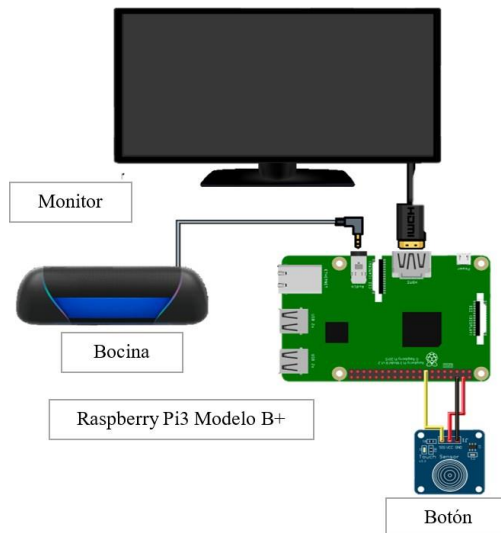


Figura 1.1 Elementos del SAAC.

El SAAC cuenta con una interfaz alfabética y un cursor que se mueve automáticamente sobre dicha interfaz fila por fila. El usuario puede seleccionar la fila que desee al presionar un botón, y una vez seleccionada la fila, el cursor se moverá columna por columna hasta llegar a la letra deseada, una vez seleccionada la letra, el cursor volverá a su movimiento de fila por fila para poder formar la oración o palabra que se desea decir, esta se muestra en la parte inferior de la interfaz, y una vez formada, es posible hacer que la frase creada se escuche de manera verbal (véase figura 1.2). Este SAAC fue presentado como trabajo de titulación (Belli Gómez, 2018).

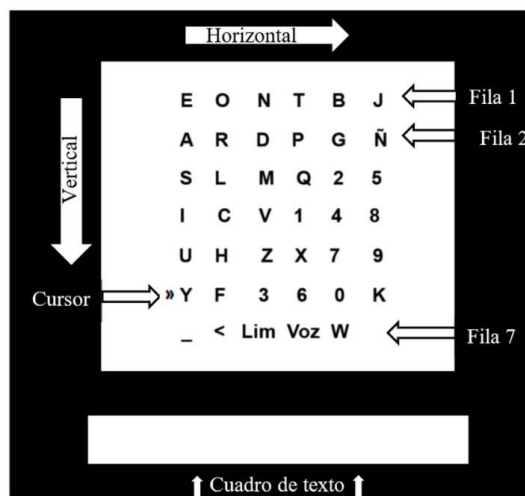


Figura 1.2 SAAC con interfaz alfabética.

Se realizó una incorporación al SAAC, creando una nueva interfaz en base a pictogramas (véase figura 1.3), está esta limitada a un poco más de 400 oraciones predefinidas, pero la ventaja de esta nueva incorporación, es que es posible crear estas frases con una cantidad reducida de “clicks”, por lo que es una manera más rápida de crearlas. Además, se conservó la interfaz alfabética a la cual se le incorporó un predictor de texto, para agilizar la escritura. Estas nuevas funciones fueron presentadas como trabajo de titulación (Delgadillo Martínez, 2020).

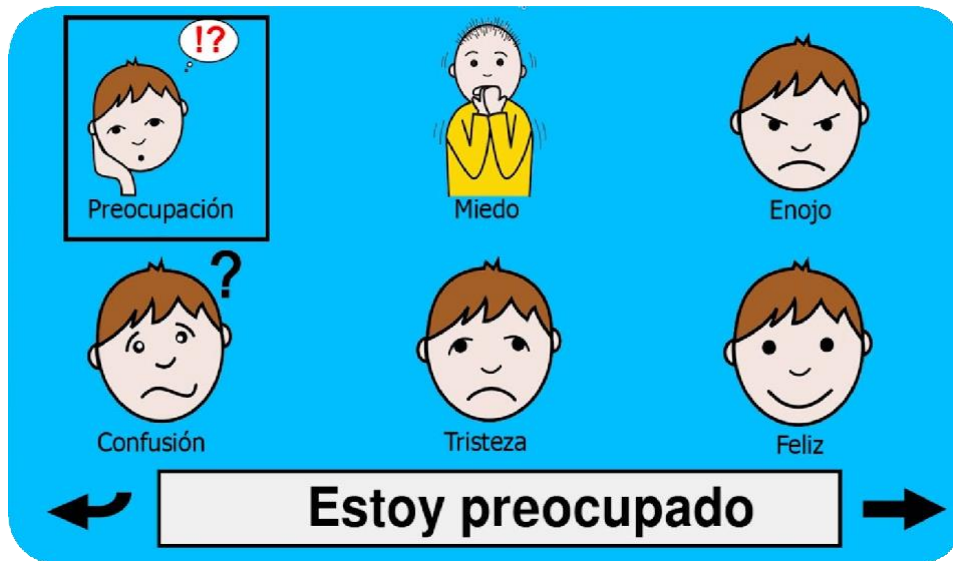


Figura 1.3 SAAC con interfaz pictográfica.

Para poder interactuar con la nueva interfaz es igual que con la alfabética, el cursor se desplaza de manera automática por cada pictograma, los cuales tienen diferentes capas de categorías, donde el usuario va ingresando dentro de las opciones hasta llegar a la opción donde se cree la frase deseada que se muestra en la parte inferior, donde esta es verbalizada por medio de la voz sintetizada.

1.6 Descripción del capitulado

Capítulo 1. Introducción. Se detallan los objetivos del proyecto, las actividades por realizar para cumplirlos, y la justificación del porqué se realizó este trabajo, mostrando un panorama de los trabajos realizados previamente por otras personas.

Capítulo 2. Marco teórico. Se muestra información esencial para la realización y entendimiento del trabajo, se describe la definición de los SAAC, pacientes que requieren uno, tipos de microcontroladores y programación de ellos, definición de señales infrarrojas, y dispositivos para el control y uso de señales infrarrojas.

Capítulo 3. Desarrollo experimental. Se describe la metodología que se siguió para el desarrollo del módulo IR, su integración con el SAAC, y comunicación del SAAC con asistentes virtuales, detallando los procesos, tales como: esquemas de bloques, diagramas de flujo, programación, desarrollo de circuitos en placa.

Capítulo 4. Pruebas y resultados. Se describen las pruebas realizadas para el desarrollo del proyecto, incluyendo prototipos, errores, adaptaciones y cambios que se realizaron para el



desarrollo del módulo IR, la integración de este al SAAC, la comunicación del SAAC con asistentes virtuales, además de pruebas realizadas con el sistema terminado para la corrección de errores visuales, así como la adaptación del SAAC para diferentes resoluciones (480p, 720p, 1080p).

Posteriormente se muestran las conclusiones del trabajo realizado, además de la mención de propuestas para ampliar las funciones del SAAC para versiones posteriores.

Se concluye el trabajo con la presentación de referencias, fuentes de imágenes y un anexo con el código realizado para la ESP32 y las plantillas utilizadas en el SAAC.

CAPÍTULO 2. MARCO TEÓRICO

2.1 Los SAAC

SAAC son las siglas de Sistemas Aumentativos y Alternativos de Comunicación, los cuales son sistemas de formas de expresión distintas al lenguaje hablado, donde el objetivo es aumentar y/o compensar el déficit de lenguaje de personas que padece alguna discapacidad (Pardo Madrid, 2017).

Los SAAC sirven para poder cambiar el lenguaje oral por otros tipos de formatos, como puede ser el lenguaje de signos (ARASAAC - Gobierno de Aragón, s.f.).

2.1.1 Tipos de SAAC

“Los tipos de SAAC son muy variados, estos deben ser elegidos debido a la necesidad de cada persona, dichos sistemas se pueden dividir en:

- Sistemas de comunicación por intercambio de imágenes (PECS): Consiste en el intercambio de una imagen por un objeto o una actividad (véase figura 2.1).



Figura 2.1. *Picture Exchange Communication System (PECS).*

- Lenguaje bimodal: Es la combinación de gestos o signos con las manos que ayudan a complementar el lenguaje oral (véase figura 2.2).

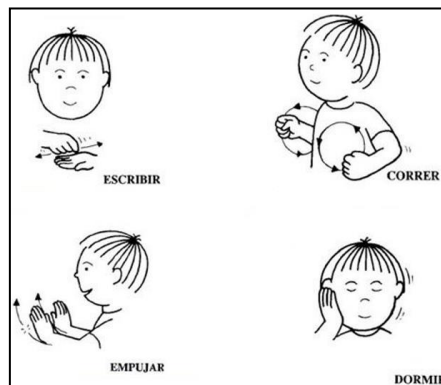


Figura 2.2. *Lenguaje bimodal.*

- Sistema pictográfico: Son imágenes que representan acciones, actividades, personas, emociones, espacios u objetos, estos pueden ir acompañados de palabras (véase figura 2.3).



Figura 2.3. Sistema pictográfico.

- Sistema gráfico: Son dibujos o fotografías de objetos o símbolos internacionales (véase figura 2.4).



Figura 2.4. Sistema gráfico.

- Productos de apoyo para la comunicación: Pueden ser tableros, libros, o aplicaciones para sistemas de computo o celulares, con programas especiales, los cuales se conforman de sistemas gráficos o pictográficos (véase figura 2.5)” (Federación Autismo Madrid, 2021).



Figura 2.5. SAAC electrónico.

2.1.2 Pacientes que requieren de los SAAC

Algunas de las causas que hacen necesario el uso de un SAAC en los pacientes pueden ser:

- Parálisis cerebral: No es considerada una enfermedad, sino un grupo de síndromes, los cuales se manifiestan como problemas motores no evolutivos, pero frecuentemente cambiantes. Son consecuencias de lesiones o malformaciones cerebrales originados en las



primeras etapas del desarrollo, cuando el cerebro está inmaduro. El diagnóstico siempre involucra un déficit motor o presenta otros síntomas como retardo mental, retardo de lenguaje, epilepsia y trastornos sensoriales (Malagón Valdez, 2007).

- Discapacidad intelectual: Es la adquisición lenta e incompleta de las habilidades cognitivas durante el desarrollo humano, lo que implica que la persona pueda tener dificultades para comprender, aprender y recordar cosas nuevas, que se manifiestan durante el desarrollo, y que contribuyen al nivel de inteligencia general, como habilidades cognitivas, motoras, sociales y de lenguaje. El término de discapacidad intelectual hoy en día es más utilizado en lugar de retraso mental (Ke & Liu, 2017).
- Trastornos del espectro autista: Es un conjunto de alteraciones heterogéneas a nivel del neurodesarrollo, inicia en la infancia y permanece durante toda la vida, lo que implica alteraciones en la comunicación social y en los comportamientos, intereses y actividades del individuo (Chaskel & Bonilla, 2017).
- Enfermedades neurológicas:
 - Esclerosis lateral amiotrófica: Es una enfermedad degenerativa de las motoneurona que es más común en adultos. Se caracteriza por una pérdida de la capacidad y actividad motora, debido a la degeneración de neuronas a todos los niveles del sistema motor (superior e inferior). Tiene una incidencia de 1.5 a 2.5 casos por cada 100 000 personas y es más conocida debido al caso del astrofísico Stephen Hawking (Marín Prida, 2009).
 - Esclerosis múltiple: Es una enfermedad autoinmune, crónica e inflamatoria del sistema nervioso central. Es una de las causas más frecuentes de discapacidad neurológica en adultos jóvenes, donde el 50% de las personas que la padecen, no tienen la capacidad de caminar después de los 15 años que inicio la enfermedad (Porrás Betancourt, Núñez Orozco, Plascencia Álvarez, Quiñones Aguilar, & Sauri Suárez, 2007).
- Párkinson: Es una enfermedad caracterizada por temblor en reposo, rigidez, bradicinesia (lentitud motora) y pérdida de los reflejos posturales, es una enfermedad que se presenta frecuentemente en personas de la tercera edad, además de personas que cuenten con historial familiar positiva (C. Scorticati & Micheli, 2006).
- Distrofias musculares: Son enfermedades hereditarias, lenta o rápidamente progresivas que afectan principalmente al músculo estriado, se caracteriza por la debilidad predominante en los músculos. Se presenta de 1 a 3 casos por cada 500 personas, y afecta el movimiento motor, se tiene marcha anormal y trastorno del lenguaje y el habla (Erazo Torricelli, 2004).
- Traumatismos craneoencefálicos: Es una patología médico-quirúrgica, caracterizada por una alteración cerebral secundaria a una lesión traumática en la cabeza, generando un daño estructural del contenido de ésta, tiene una incidencia de 200 casos por cada 100 000 personas, y es causada más frecuentemente por accidentes de tránsito (70%), seguido de hechos violentos y/o caídas. Las consecuencias de esto, son problemas cerebrales que impidan el funcionamiento de la actividad motora o incluso puede producir la muerte (Charry, Cáceres, Salazar, López, & Solano, 2017).



- Afasias: Es un trastorno del lenguaje ocasionado por alguna lesión cerebral, donde la persona no presentaba ningún problema del habla. Se caracteriza por problemas en la emisión del habla y déficit de la comprensión (Vendrell, 2001).
- Pluridiscapacidades de tipologías diversas: Personas con una funcionalidad limitada y procesos de desarrollo lentos, donde presentan la disfunción de dos o más áreas del desarrollo, incluyendo siempre déficit cognitivo (Soro Camats, Basil, & Rosell, 2012).

2.2 Raspberry Pi

Es una mini computadora de bajo costo, de tamaño compacto, y puede ser conectada por medio de HDMI a un monitor o TV, además de necesitar un mouse y teclado externo. Este computador corre con un sistema operativo Linux, con la capacidad para programar en lenguajes como Scratch y Python (MCI Electronics, s.f.), el sistema operativo es Raspberry Pi OS, el cual, anteriormente era conocido como Raspbian, y está basado en el sistema operativo Debian de GNU/Linux (Allan, Imeson, & Scheller, s.f.).

Raspberry Pi 4 Model B

El modelo Raspberry Pi 4 Model B, tiene características que permiten el desarrollo de diferentes proyectos. Cuenta con un chip Soc Broadcom BCM2711, Cortex-A72 de cuatro núcleos (ARM v8) de 64 bits a 1.5 GHz, una memoria SDRAM LPDDR4-3200 de 1, 2, 4 u 8 GB, conexión Wifi 2.4 y 5 GHz IEEE 802.11 ac, y Bluetooth 5.0, BLE Gigabit Ethernet.

Como se observa en la figura 2.6, contiene dos puertos USB 3.0 y dos puertos USB 2.0, dos salidas micro-HDMI compatibles hasta 4K@60Hz, tiene una fuente de alimentación en puerto USB.C de 5 Vcc a 3 A, y un funcionamiento de 0-50°C. Además, tiene integrado un encabezado de GPIO de 40 pines estándar de Raspberry Pi (Raspberry Pi, s.f.).

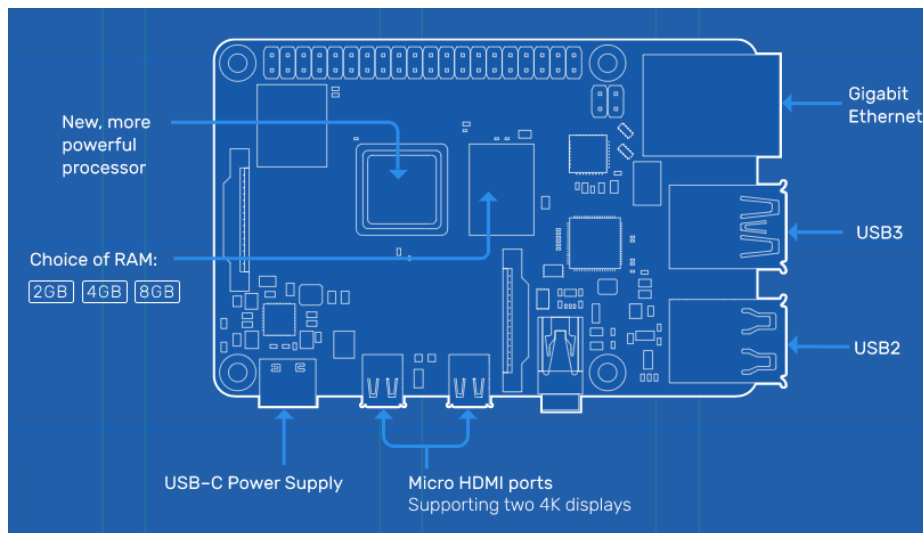


Figura 2.6 Diagrama de Raspberry Pi 4 Model B.

Para el uso de este computador, es necesario instalar el sistema operativo en una memoria SD, una vez descargado el sistema en la memoria, se ejecuta dentro de la Raspberry, todo este



proceso se puede revisar en la documentación de Raspberry Pi que se encuentra en su página oficial¹.

La información sobre el encabezado de GPIO que contiene la Raspberry Pi4 se muestran en la figura 2.7.

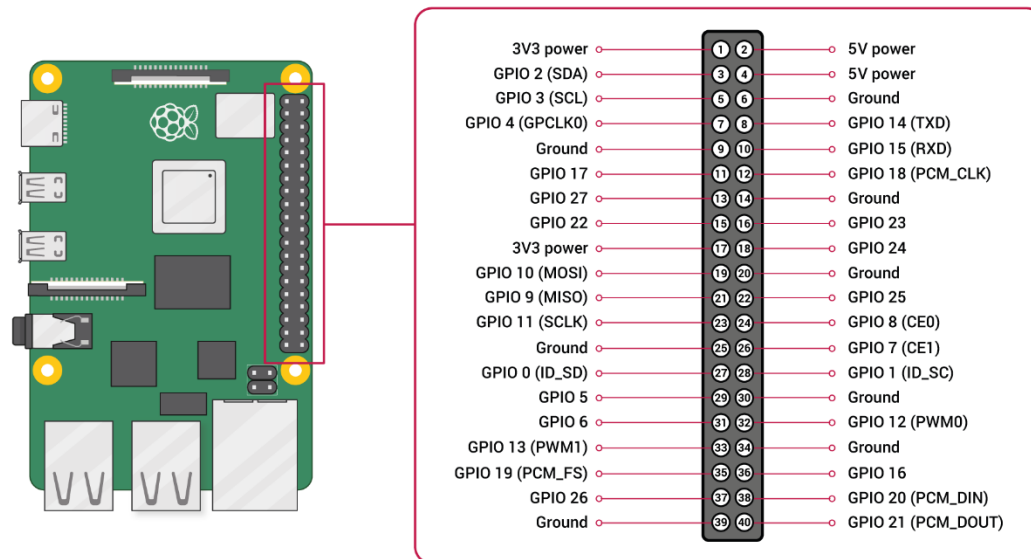


Figura 2.7 Pinout de Raspberry Pi4.

- Alimentación: son los pines que se utilizan para conectar las líneas de alimentación para los circuitos electrónicos. Proveen de energía de 5 V y 3 V (3.3 V limitados a 50 mA de carga), además se tiene los pines de tierra (GND o Ground).
- DNC (Do Not Connect): son los pines que solo existen en algunas versiones, los cuales no tienen ninguna función, se usan como GND por lo general.
- Pines configurables: son los GPIO sin una función en específico, pero estos se pueden programar.
- Pines especiales: son conexiones que están destinados para conexiones o interfaces determinadas como UART, serie TXD y RXD, SDA, SCL, MOSI, MISO, SCLK, CE0, CE1.
- PWM: pueden regular el ancho del pulso, son los GPIO12, GPIO13, GPIO18 y GPIO19.
- SPI: interfaz de comunicación, donde se tienen dos canales para esto:
 - SPI0: MOSI (GPIO10), MISO (GPIO9), SCLK (GPIO11), CE0 (GPIO8), CE1 (GPIO7)
 - SPI1: MOSI (GPIO20); MISO (GPIO19); SCLK (GPIO21); CE0 (GPIO18); CE1 (GPIO17); CE2 (GPIO16)
- I2C: interfaz de comunicación serial entre dos dispositivos digitales. Este bus está compuesto por la señal de datos (GPIO2) y el reloj (GPIO3). Además de EEPROM Data (GPIO0) y EEPROM Clock (GPIO1).
- Serial: interfaz de comunicación con pines TX (GPIO14) y RX (GPIO15).

¹ <https://www.raspberrypi.com/documentation/>



2.3 ESP32

Es un microcontrolador creado por la empresa Espressif Systems, tiene como precedente al ESP8266, es una serie SoC (System on Chip), el cuál integra módulos de bajo costo y bajo consumo energético, está diseñado para dispositivos móviles, dispositivos electrónicos portátiles y aplicaciones IoT (Internet of Things). La familia de ESP32 incluye los chips ESP32-S0WD, ESP32-D0WDQ6, ESP32-D0WD, ESP32-U4WDH, ESP32-PICO, los cuales son integrados en diferentes módulos para aplicaciones distintas (Guerra Carminate, s.f.).

ESP32-WROOM-32

Módulo con chip ESP32-D0WDQ6, tiene doble núcleo de CPU con microcontroladores de bajo consumo, Tensilica Xtensa de 32 bits, frecuencia de reloj ajustable de 80 MHz a 240 MHz.

Posee una potencia de salida de +19.5 dBm, Bluetooth v4.2 BR/EDR y Bluetooth LE (Low Energy) para conexiones con teléfonos inteligentes y conexión Wi-Fi (802.11 b/g/n) HT40 alcanzando hasta 150 Mbits/s. Está conformado por una memoria ROM de 448 KB, y memoria SRAM de 520 KB, con 4MB de memoria flas adicional, incluyendo una antena impresa tipo MIFA para la interfaz de radio.

Puede soportar temperaturas de -40°C a 125°C , y contiene los periféricos que se observan en la figura 2.8 (ESPRESSIF SYSTEMS, s.f.):

- 25 canales GPIO: GPIO 34, 35, 36 y 39 son únicamente de entrada, además de que los GPIO 14, 16, 17, 18, 19, 21, 22 y 23 contienen pull up interno; todos pueden usarse como interrupciones.
- 18 canales de conversión analógico-digital (ADC): GPIO 2, 4, 12, 13, 14, 15, 25, 26, 27, 32, 33, 34, 35, 36 y 39 a 12 bits de resolución.
- 2 convertidores digital-analógico (DAC): GPIO25 y 26.
- 10 GPIO de detección capacitiva: GPIO 0, 2, 4, 12, 13, 14, 15, 27, 32 y 33.
- 3 interfaces UART: GPIO 1, 3, 16, 17, 19 y 22.
- 3 interfaces SPI: GPIO 2, 4, 5, 12, 13, 14, 15, 18, 19, 21, 22 y 23.
- 2 interfaces I2C: GPIO 21 y 22.
- 2 interfaces I2S: GPIO 5, 12, 13, 14, 15, 18, 19 y 23.
- 16 canales de salida PWM: Todos los pines pueden generar PWM excepto de los GPIO 34 al 39 (REDACCIÓN, s.f.).

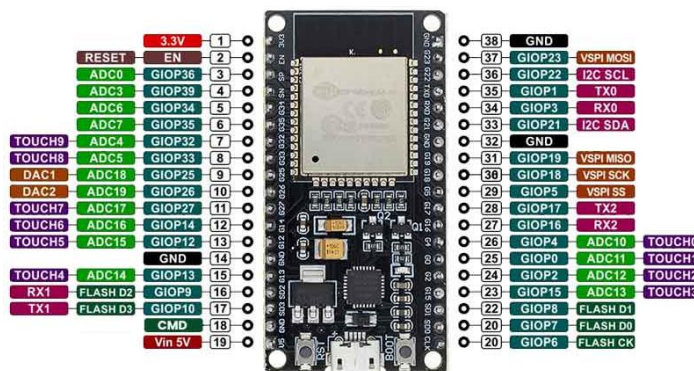


Figura 2.8 Pinout de la placa ESP32.



2.4 Códigos infrarrojos

2.4.1 Rayos infrarrojos

Son una forma de radiación, los cuales forman parte del espectro electromagnético, estos tienen una longitud de onda que va de entre 700 nm y 1000 nm, por lo que están fuera del rango visible del ser humano como se observa en la figura 2.9 (Instituto Tecnológico de Aguascalientes, s.f.).

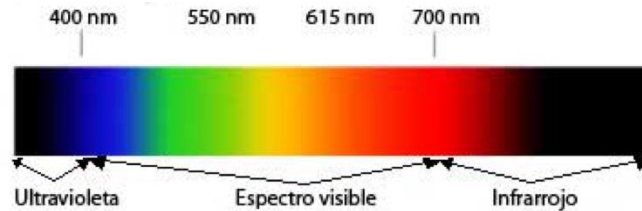


Figura 2.9 Espectro de luz visible.

2.4.2 Controles remotos

Los controles remotos están conformados por una memoria, en la cual se tienen guardados los códigos que serán transmitidos al presionar un botón. Las señales infrarrojas transportan un código de N bits para cada botón, estas señales se encuentran en el rango de los 940 nm, oscilando a una frecuencia de aproximadamente 319.15 THz como se observa en la figura 2.10.

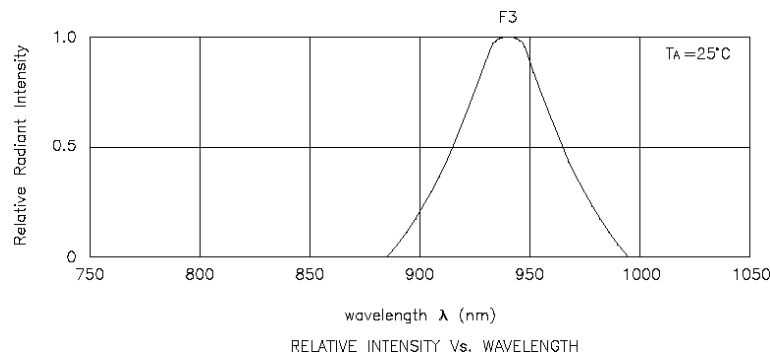


Figura 2.10 Longitud de onda de una señal de un control remoto.

Un ejemplo de una señal producida por un control remoto se muestra en la figura 2.11.

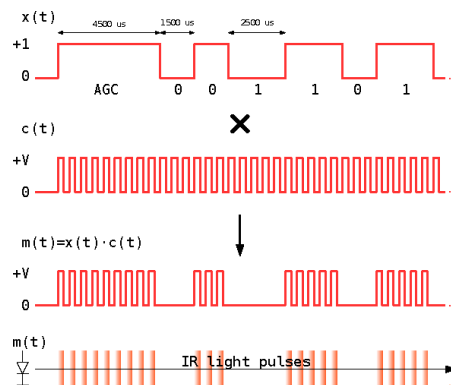


Figura 2.11 Ejemplo de una señal de un control remoto.



Las señales:

- $x(t)$ es la señal moduladora, donde los bits van codificados en la señal de acuerdo con un determinado tipo de modulación, esta es la señal que contiene el código del botón.
- $c(t)$ es la portadora eléctrica, es una señal cuadrada de 38 kHz.
- $m(t)$ es la señal modulada, el cual es el producto de la señal moduladora y de la portadora, esta señal es la que excita al LED IR (Valcarce, s.f.).

2.4.3 Protocolos

Existen una gran variedad de dispositivos que funcionan mediante señales infrarrojas, por lo que las empresas realizaron un protocolo de transmisión único para cada marca, en los que se definen los tiempos de las tramas de la señal moduladora, bits de inicio y de termino.

Los protocolos creados por las marcas pueden ser: NEC, SONY, RC5, RC6, DISH, SHARP, JVC, SANYO, MITSUBISHI, SAMSUNG, LG, WHYNTER, PANASONIC (Sensor IoT, 2017).

La forma más sencilla de obtener cualquier señal infrarroja, es su señal RAW (sin decodificar), ya que esta obtiene la señal con los tiempos exactos de cada trama, incluyendo los bits de inicio y de termino, lo que significa una solución para el trabajo con diferentes aparatos electrónicos, sin importar la compañía y su protocolo.

2.4.4 Receptor KY-022

Es un dispositivo electrónico que nos permite detectar señales infrarrojas que estén dentro de su rango de cobertura y de frecuencia. El módulo KY-022, está construido por un receptor IR TL1838, una resistencia SMD de 1 k Ω y un LED SMD, este trabaja a una frecuencia de 38 kHz, necesita una alimentación de 2.7 a 5 V, una corriente de 0.4 a 1.5 mA y tiene una distancia de recepción de 18 m con un ángulo de $\pm 45^\circ$ (Electrónica J&M Computers & Technologies, s.f.).

En la figura 2.12 podemos observar el módulo y sus componentes.

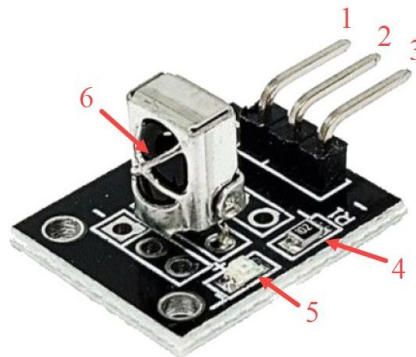


Figura 2.12 Módulo KY-022.

Sus componentes son:

1. Pin de señal de datos: Pin por donde se transmiten los datos recibidos.
2. Pin de voltaje: Pin de alimentación.
3. Pin de tierra.



4. Resistencia: Para limitar la corriente que llega al LED.
5. LED: Dispositivo de visualización, nos indica con un parpadeo si se recibió una señal.
6. TL1838: Receptor infrarrojo.

2.4.5 Transmisor IR383

Los transmisores infrarrojos, son un tipo específico de diodo emisor de luz en un espectro infrarrojo, donde no es visible al ojo humano, pero esta puede ser detectada por un receptor IR.

El transmisor IR383 produce una señal con una longitud de onda de 940 nm, con un ángulo de 12°. Este necesita 1.3 V de polarización directa y un máximo de 1.7 V a 20 mA (Electrónica Steren, s.f.).

Como se observa en la figura 2.13, este transmisor luce prácticamente igual que un LED común y se polariza de la misma forma.

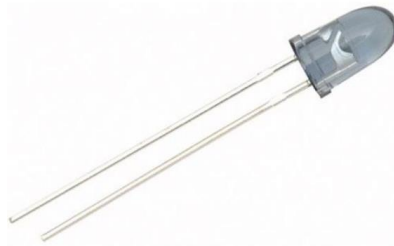


Figura 2.13 Transmisor IR383.

2.4.6 Circuitos de control de señales

Para realizar el envío de una señal infrarroja, es necesario controlar la salida proveniente del microcontrolador, esto se puede realizar mediante la ayuda de un transistor TBJ, ya que se puede configurar su forma de funcionamiento según el circuito en el que se encuentra, donde puede trabajar como un interruptor electrónico cuando opera en zona de corte y de saturación respectivamente, lo que permite (polarizado en la región de saturación) o impide (polarizado en la región de corte) el paso de corriente en el circuito, dando como resultado dos estados lógicos 0 y 1 (Ayala, 2019).

El transistor que se observa en la figura 2.14, es un transistor TBJ 2n2222A TO-18, de tipo NPN, encapsulado de metal, de uso común, baja potencia, capaz de disipar hasta 500 mW, corriente en el colector constante (I_c) de 800 mA, ganancia de corriente >100 hFE y voltaje colector emisor en corte (V_{c-e}) de 50 V (Isaac, s.f.).

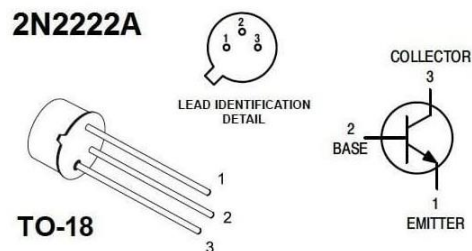


Figura 2.14 Transistor TBJ 2n2222A TO-18.



Una configuración básica de un circuito con un transistor TBJ en modo interruptor, se muestra en la figura 2.15.

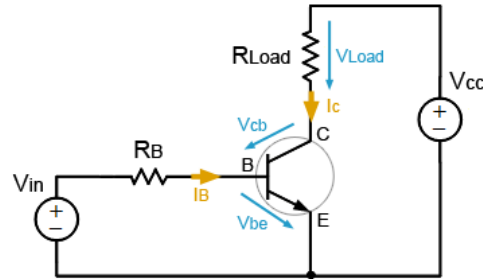


Figura 2.15 Circuito para transistor TBJ como interruptor.

Donde:

V_{cc} : es el voltaje de alimentación

V_{in} : es el voltaje que se usará como interruptor

R_b : es la resistencia de la base

R_{load} : es la resistencia de la carga

Las resistencias junto con el V_{in} , determinarán el funcionamiento del transistor, cambiando su estado de corte o saturación (Llamas, s.f.).

2.5 Interruptores

Son componentes electrónicos que permiten o impiden el paso de la corriente eléctrica cuando estos se pulsan o aprietan, y al ser soltados vuelven a su posición inicial (AREATECNOLOGIA, s.f.).

Existe una gran variedad de interruptores que se adaptan a la necesidad del paciente, ya que no todos los pacientes son capaces de realizar la misma acción, estos tienen diferentes formas de accionarse, en la tabla 2.1 se pueden observar ejemplos de estos (Gonzalez Rus).

Tabla 2.1 Tipos de interruptores.

Modelo	Acción	Descripción	Imagen
Almohadilla o botón	Por presión	El paciente debe presionar el interruptor sin importar con que parte del cuerpo haga esto.	
Interruptor de mercurio	Por posición	El paciente deberá mover o girar el interruptor para que este se active.	



Cylindrical Touch	Por contacto o sensibilidad	No es necesario ejercer presión sobre él, al mínimo contacto o roce se activará.	
Flexo o palanca	De palanca	El paciente deberá jalar la palanca para activarlo.	
Interruptor de pedal	De pedal	El paciente deberá activarlo ejerciendo presión con el pie.	
Soplo-succión	De soplo-succión	El paciente lo activa con la presión aérea (soplo o respiración).	
Interruptor de sonido	Por sonido o vibración	Es un micrófono que se activará con el sonido o ruido.	
Interruptor de luz	Por luz	El interruptor emitirá una luz o rayo infrarrojo que al ser tapado se activará.	
Pulsador Joystick	Joystick	Se acciona cada vez que se mueve el joystick.	

2.6 Dispositivos de visualización

Son aparatos electrónicos que nos permiten como usuario, observar información con respecto al dispositivo utilizado. Existen diferentes tipos que van desde lo más simple, como una luz, a componentes más complejos, como pantallas de alta resolución.

2.6.1 LED

Un diodo emisor de luz, mejor conocido como LED por sus siglas en inglés (Light Emitting Diode), es un dispositivo semiconductor, con una longitud de onda monocromática. El símbolo que representa este dispositivo se observa en la figura 2.16.

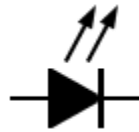


Figura 2.16 Diodo LED.

Al tratarse de un diodo, podemos observar que tiene un ánodo y un cátodo, lo cual nos indica la forma en que se debe realizar la polarización para permitir el paso de la corriente eléctrica, y dependiendo de la frecuencia en la que trabaje, estos emitirán un color de luz diferente (González, s.f.).

2.6.2 Display de segmentos

Son dispositivos electrónicos que nos ayudan a representar números decimales y de cierta forma algunas letras, están compuestos por segmentos de LED individuales o cristales líquidos que se activan por medio de un pin de control para cada uno de sus segmentos, todos se alimentan por un cátodo (el pin de control se alimenta con la fuente) o un ánodo (el pin de control se conecta a tierra) en común según sea el caso (véase figura 2.17). Existen en segmentos de 7, 14 y 16 (Premier Farnell Limite, s.f.).

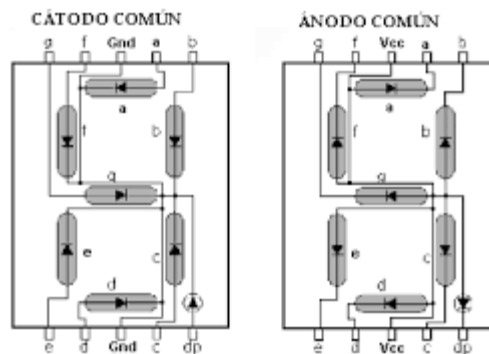


Figura 2.17 Display de 7 segmentos y sus conexiones.

2.6.3 Pantalla LCD

Las pantallas de cristal líquido (por sus siglas en inglés Liquid Crystal Display), son pantallas que cuentan con cientos de miles de píxeles, que son utilizados ampliamente en dispositivos



electrónicos como teléfonos, tabletas, cámaras digitales, pantallas de televisión o monitores de computadoras en sus formas más complejas (véase figura 2.18), pero también existen las pantallas más sencillas, utilizadas estéreos sencillos, proyectos, entre otros, que nos llegan a mostrar mucha menos información (véase figura 2.19).



Figura 2.18 Pantalla LCD.



Figura 2.19 Display LCD de 2x16.

Estos dispositivos están formados por un arreglo bidimensional de píxeles, donde cada uno de ellos puede estar encendido o apagado de manera independiente. Está conformado en sus componentes básicos por celdas de cristal líquido, polarizador, electrodos y placas de vidrio (véase figura 2.20) (Mellado Villaseñor, 2013).

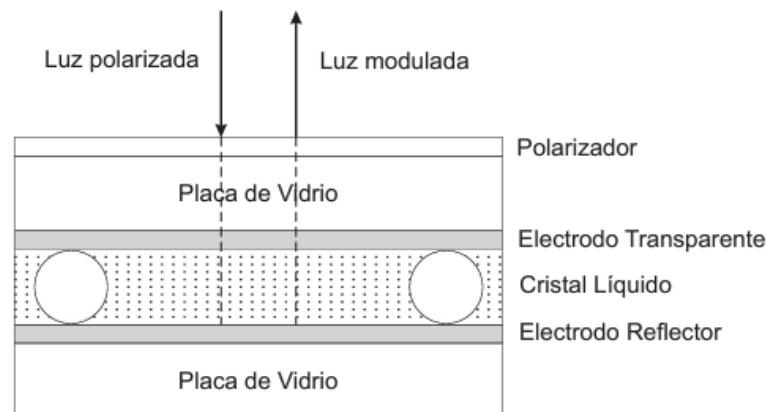


Figura 2.20 Diagrama esquemático de un píxel de una LCD.

2.7 Asistentes virtuales

Los asistentes virtuales inteligentes, son un tipo de software que ayuda al usuario a realizar o automatizar tareas u ofrecer un servicio con la mínima interacción entre hombre-máquina. La comunicación que se da entre ambos es de forma natural, es decir, la persona habla directamente con el asistente y este recibe, procesa, interpreta, realiza la tarea y en ocasiones responde de la misma forma hablada.

Las tareas realizadas están basadas en datos de entrada del usuario, pueden ser la ubicación y autorización para el acceso de cierta información personal como correos, calendarios, cuentas de plataformas de streaming, además de acceso a recursos en línea como el clima, tráfico, noticias, precios de acciones, alarmas, etcétera (Equipo BLOG Grupo Cajamar, 2021).



Hoy en día también existe una gran variedad de dispositivos de uso domótico, los cuales son compatibles con este tipo de asistentes, podemos encontrar dispositivos como focos, ventiladores, contactos eléctricos, detectores de movimiento, humidificadores, vaporizadores, motores para persianas, entre muchos otros, los cuales pueden ser controlados y programados a través de los asistentes.

En la tabla 2.2 se puede observar los cinco asistentes virtuales con mayor impacto en el mercado en una puntuación de -250 a 250 puntos, en la cual se calificó aspectos como la amplitud de uso y el grado de aceptación, entre otros (Mena Roa, 2020).

Tabla 2.2 Asistentes de voz con mayor impacto en el mercado.

Asistente Virtual	Calificación (-250 a 250)
Alexa (Amazon)	110.2
Google Assistant	97.3
Siri (Apple)	38.3
Bixby (Samsung)	25.5
SoundHound	11.0

2.8 Python

Es un lenguaje de programación de alto nivel, fue creado por los años 90 por por Guido Van. Sus características son:

- Lenguaje interpretado o de script: es el lenguaje que se ejecuta utilizando un programa intermedio conocido como intérprete, el cual sustituye la acción de compilar el código a lenguaje máquina (este código compilado es el que una computadora puede comprender y ejecutar directamente). Los lenguajes interpretados son más flexibles y portables. Python es un lenguaje semi interpretado ya que contiene muchas características de los lenguajes compilados, su código fuente se traduce a un pseudo código llamado bytecode la primera vez que se ejecuta, lo que genera archivos con terminación .pyc o .pyo, estos son los que se van a ejecutar en las siguientes ocasiones.
- Tipado dinámico: esta característica nos indica que no es necesario establecer el tipo de dato que vamos a asignar a una variable, por lo que esta va a determinarlo en el momento de ejecución dependiendo de los valores que adquiera.
- Fuertemente tipado: a una variable no se le permite tratar valores distintos fuera del tipo previamente definido, por lo que es necesario convertir de forma explícita dicha variables al nuevo tipo.
- Multiplataforma: está disponible para diversas plataformas como UNIX, Linux, DOS, Windows, Mac OS, etc., nos indica que, si no se usan paquetes específicos para una plataforma, el programa se podrá ejecutar en otro sistema sin tener cambios.
- Orientado a objetos: la ejecución del programa consiste en una serie de interacciones entre objetos y clases, con la finalidad de reducir el código ya que no es necesario repetir ciertas secciones códigos si ya fueron establecidas previamente (González Duque).



2.9 Arduino IDE

Un IDE se define como un entorno de desarrollo integrado (por sus siglas en inglés Integrated Development Environment). Por lo que Arduino IDE es un programa compuesto por un conjunto de herramientas de programación, está empaquetado como aplicación, pero este esta conformado por un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI), además, tiene herramientas necesarias para poder cargar el programa compilado a la memoria flash del hardware (Crespo, 15).

La ventana de Arduino IDE se muestra en la figura 2.21

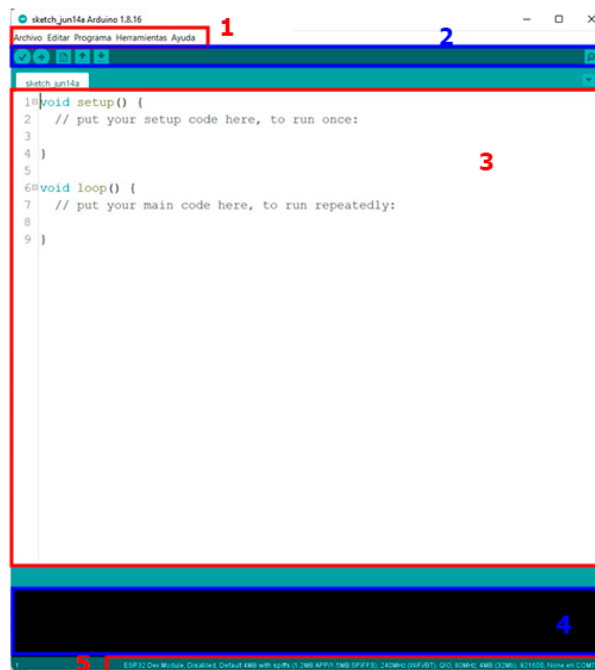


Figura 2.21 Arduino IDE.

Esta se divide en 5 secciones principales:

1. Barra de menús: Se encuentran todos los menús que son:
 - Archivo (véase figura 2.22): En este menú tenemos varias opciones, una de las opciones importantes es el menú de preferencias.

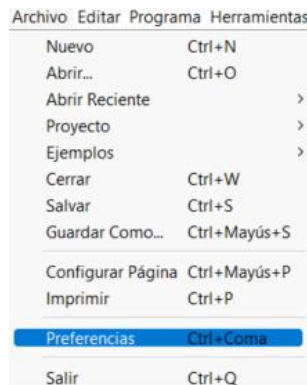




Figura 2.22 Menú Archivo.

En el menú de preferencias podemos editar diferentes opciones del IDE, donde en el botón de “Gestor de tarjetas adicionales” que se muestra en la figura 2.23, se deben agregar los links correspondientes para el uso de placas diferentes a Arduino, como lo son las placas de ESP32.

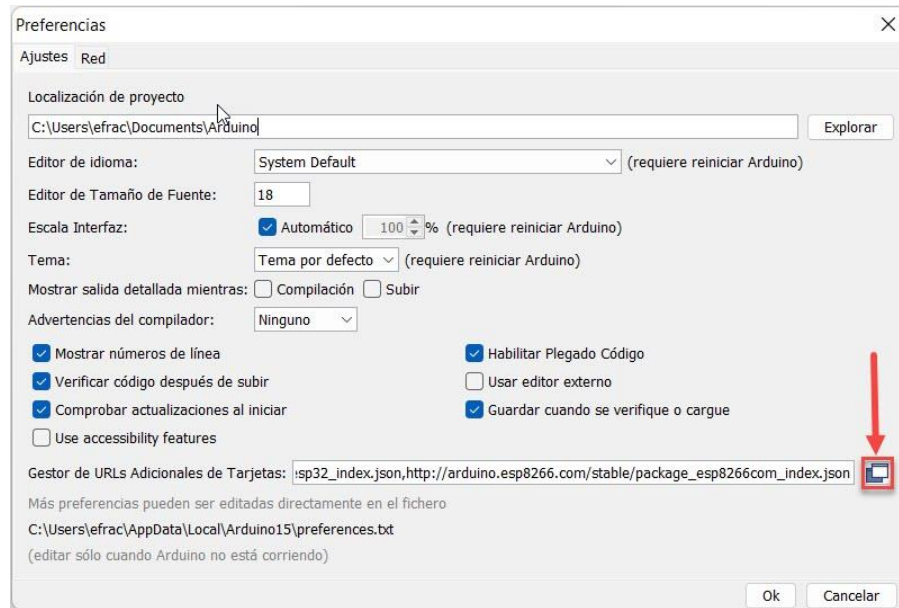


Figura 2.23 Botón de gestor de tarjetas adicionales.

Al seleccionar el botón, se abre una ventana flotante (véase figura 2.24), donde se deben pegar los links correspondientes para la tarjeta ESP32^{1,2} y al finalizar presionar el botón “Ok” en la ventana del Gestor y en la ventana de Preferencias, para guardar los cambios.



Figura 2.24 Gestor de URLs Adicionales de Tarjetas.

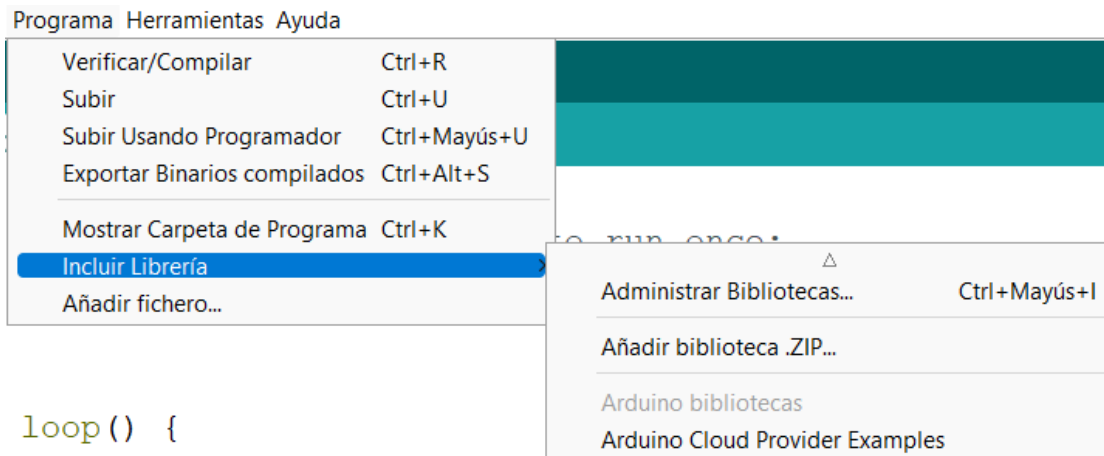
- Editar: En este menú tenemos las opciones para poder editar fácilmente el programa con opciones como pegar, copiar, cortar, buscar, etc.
- Programa: En este menú (véase figura 2.25) encontramos las opciones para cargar los programas a la placa, verificarlos, compilarlos, etc, donde además se encuentra el menú

¹https://dl.espressif.com/dl/package_esp32_index.json

²http://arduino.esp8266.com/stable/package_esp8266com_index.json



de bibliotecas, en el cual se pueden descargar bibliotecas que se encuentren dentro del Arduino IDE o cargar bibliotecas descargadas en una carpeta .zip.



```
loop() {
```

Figura 2.25 Menú Programa> Incluir Librería.

Para agregar bibliotecas incluidas en el IDE, se debe entrar al menú Administrar Bibliotecas y buscar la biblioteca deseada e instalarla.

Se pueden agregar bibliotecas externas que, aunque ya existan dentro del IDE, contengan alguna modificación adicional. Es necesario descargarlas como una carpeta .zip, después se debe abrir el menú Añadir biblioteca ZIP, donde se debe seleccionar el archivo descargado.

Un ejemplo de una biblioteca externa es la biblioteca ESP32-IRremote, la cual existe dentro de las bibliotecas IDE como IRremote, pero la limitación de la biblioteca oficial, es que al utilizarlos en la placa ESP32 solamente permite la recepción de códigos y no la transmisión, por lo que la alternativa es una biblioteca modificada¹, la cual incluye la función del envío de códigos, esta se puede descargar como .zip.

Esta biblioteca debe iniciarse con las instrucciones:

```
IRrecv nombreReceptor(4); //Donde se define el pin donde se va a conectar el receptor
IRsend nombreTransmisor(5); //Donde se define el pin donde se va a conectar el transmisor
```

También podemos hacer uso de ciertas bibliotecas, las cuales no se necesitan instalar y siempre están disponibles, como la biblioteca Serial, en la cual se debe iniciarse para poderla usar, además de indicar a qué velocidad debe trabajar, esto se realiza a través de la función:

```
Serial.begin(115200); //Entre paréntesis la velocidad de transmisión.
```

- Herramientas: Aquí encontramos diferentes herramientas, entre las cuales está el menú placa (véase figura 2.26), donde se selecciona que tipo de placa vamos a usar, además de indicar en que puerto COM vamos a transmitir y la velocidad de transmisión de la placa.

¹ <https://github.com/SensorsIot/Definitive-Guide-to-IR/tree/master/ESP32-IRremote>

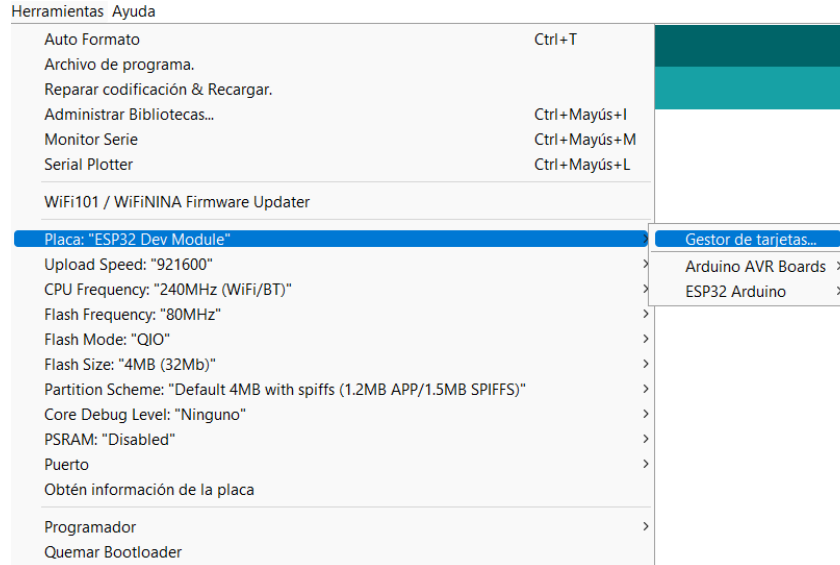


Figura 2.26 Menú Herramientas.

Cuando hemos agregado una biblioteca diferente para otra placa (como se describió en el menú de archivo previamente), es necesario descargar estas bibliotecas, para esto se debe ingresar al menú placa y entrar al menú gestor de tarjetas.

Al abrir el menú, en el buscador debemos poner “esp32”, donde nos aparece la biblioteca correspondiente (véase figura 2.27) y debemos instalarla.

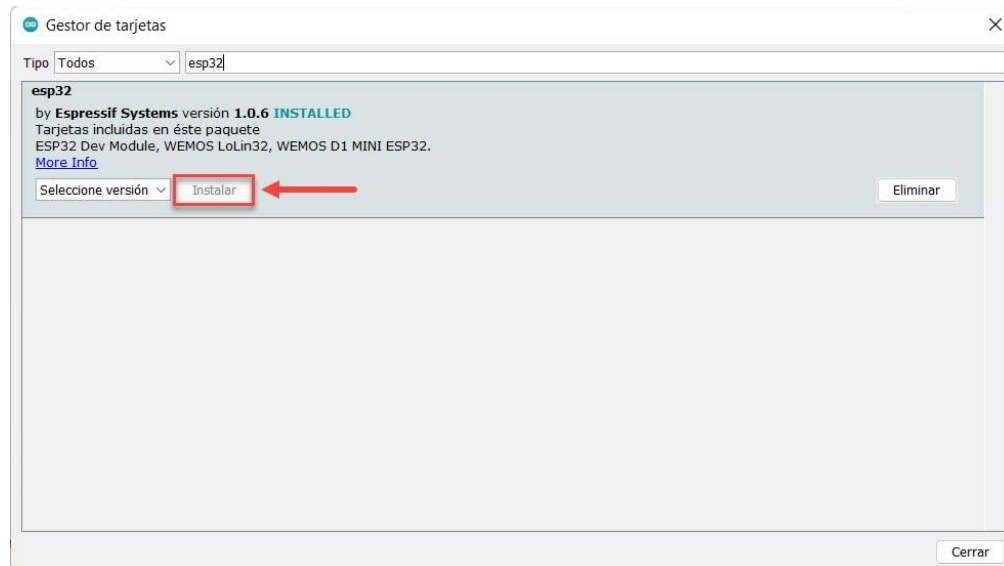


Figura 2.27 Gestor de tarjetas.

Finalmente cerramos todas las ventanas y en el menú Herramientas>Placa>ESP32 Arduino, seleccionamos la placa a usar como se muestra en la figura 2.28 (Lozano, 2021).

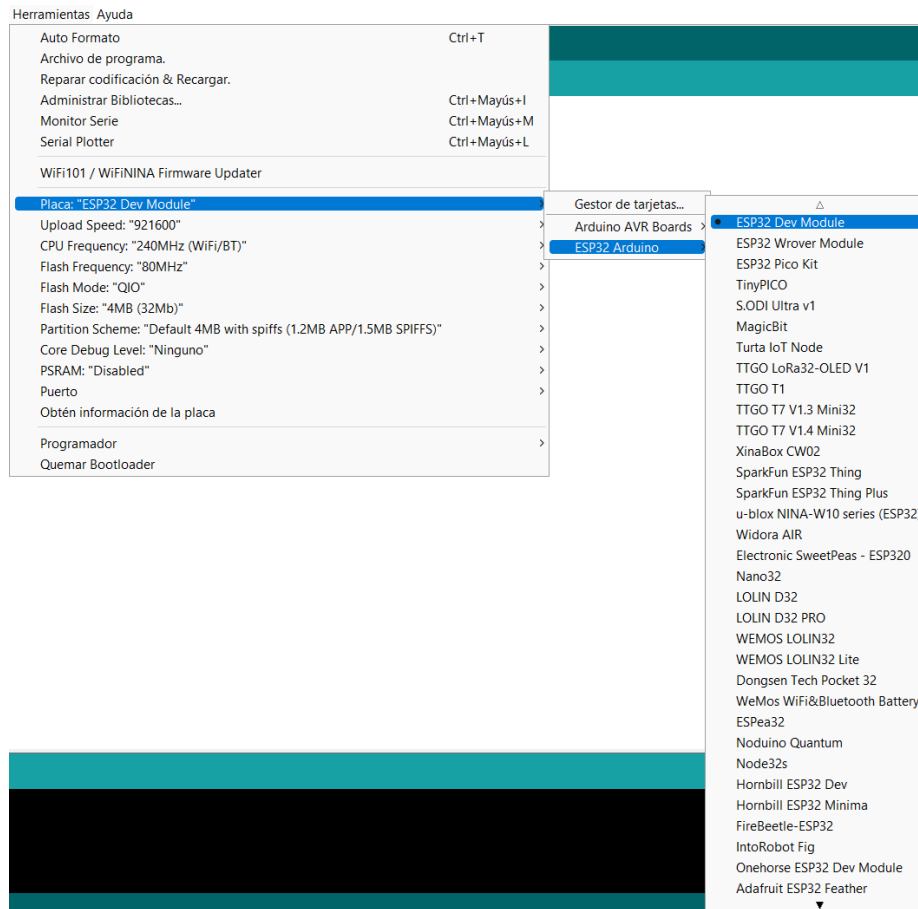


Figura 2.28 Selección de placa.

- Ayuda: Menús donde se proporciona información sobre dudas que se tengan del programa, además de tener la documentación de este.
2. Barra de botones: Se encuentran los botones de:
- Verificar/Compilar: Comprueba la sintaxis del código fuente que se encuentra en el área de edición, si llegará a encontrar algún error lo mostrará en la consola de salida, además de resaltar con color rojo la línea en donde se encuentra el error.
 - Subir: Transfiere el programa que se encuentra en el área de edición a la placa, esta función previamente verifica y compila el código y posteriormente lo graba en el microcontrolador.
 - Nuevo: Permite crear un nuevo programa (sketch).
 - Abrir: Permite cargar un sketch previamente escrito.
 - Guardar: Permite guardar el sketch en un archivo con terminación .ino.
 - Monitor Serie: Permite la comunicación en tiempo real entre la placa y el dispositivo en el cual se programa. El monitor serie se observa en la figura 2.29:

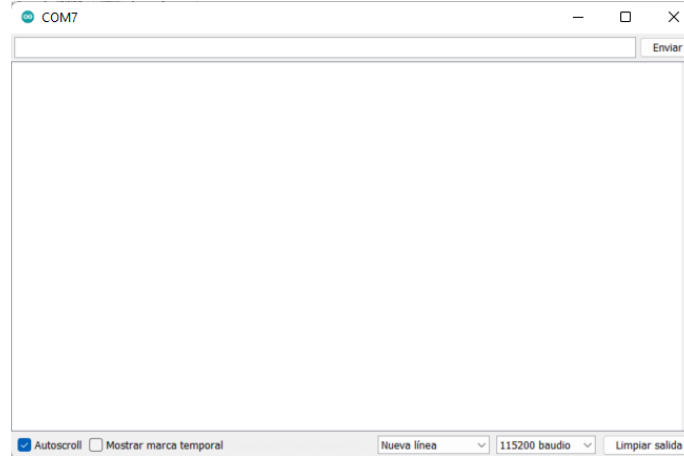


Figura 2.29 Monitor Serie Arduino.

El monitor serie tiene en la parte superior el nombre del puerto por el cual se está conectado, la barra inferior es una línea de entrada donde se pueden ingresar los caracteres que se desean enviar a la placa, presionando el botón de enviar o simplemente presionar el Enter de la computadora.

La parte más amplia es la ventana de salida del monitor serie de la placa, observaremos todo lo que la placa este enviando por este puerto.

En la parte inferior encontramos botones de acción y configuración, los botones de acción sirven para tener un Autoscroll que es un deslizamiento automático de los datos enviados desde la placa al dispositivo, además de Mostrar marca temporal el cual nos indica la hora en la que nos llegaron los datos por el puerto serie.

Los botones de configuración, son donde se configura la manera de mostrar los datos recibidos, en una nueva línea, con retorno de carro, sin ajuste de línea o ambos. También tenemos la velocidad a la que está trabajando la placa, esta es importante verificar, ya que si se encuentra en una velocidad diferente a la establecida en la configuración se recibirán caracteres diferentes; y finalmente tenemos un botón para limpiar la salida del monitor serie.

3. Área de edición: En esta área se debe escribir el código fuente de los programas, el cual debe contener la función `void setup () {}`, y `void loop () {}` forzosamente para poder compilar. La función `void setup ()` es para poder configurar la tarjeta, esta función solamente se ejecutará una sola vez al encenderse. La función `void loop ()` es una función que se estará ejecutando todo el tiempo que este encendida la placa, a menos de que exista alguna interrupción o llamado a otra función. Fuera de estas dos funciones se pueden agregar interrupciones o funciones que serán llamadas después.
4. Consola de salida: Nos indica el estado del código, avisa de errores al compilar, además del estado al compilar y subir el código a una placa, en el cual nos proporciona la información adicional como el espacio usado en la memoria del microcontrolador y memoria disponible.
5. Barra de información: Proporciona la información sobre la placa usada como el modelo que tenemos o el puerto serie utilizado.



2.10 PCB

Las PCB (por sus siglas en inglés Printed Circuit Board), son placas o tarjetas de circuito impreso, que contiene y conecta los componentes electrónicos entre sí, mediante pistas de cobre, para que un circuito funcione según su diseño, sin la necesidad de interconectar dichos componentes con cables expuestos (Peterson, 2020).

2.10.1 Software para diseño de PCB con EasyEDA

Existe una gran variedad de software que nos ayudan a diseñar una placa PCB, para ello se necesita tener un circuito elaborado y probado previamente, un ejemplo de estos programas es EasyEDA.

Este programa contiene herramientas diseñadas para realización de proyectos y producción de sistemas electrónicos, abarca desde la creación de un circuito integrado, hasta el desarrollo de placas PCB, es una herramienta gratuita por lo que no requiere instalación ya que está basado en la nube, por lo que es posible encontrar proyectos ya elaborados dentro del programa (Pérez Esteso, 2016). Es necesario registrarse en la página para poder utilizar, su pantalla principal se puede observar en la figura 2.30.

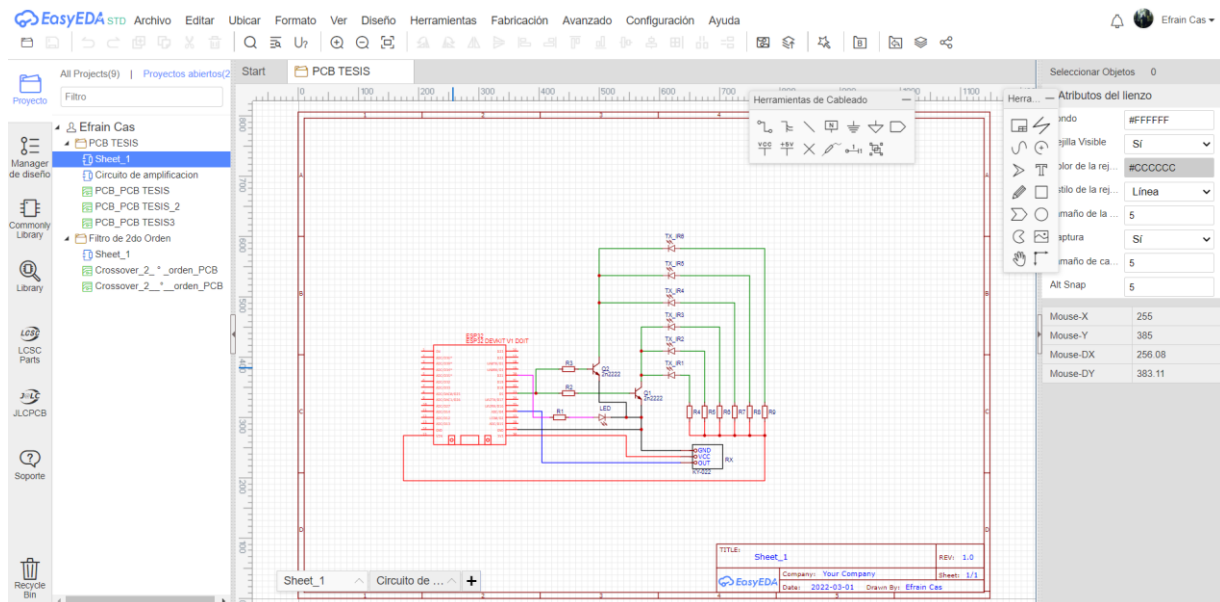


Figura 2.30 Interfaz principal de EasyEDA.

En la parte de la izquierda se encuentran los proyectos existentes y también las bibliotecas de los componentes electrónicos, los cuales vienen con las medidas exactas de los componentes para ser añadidos a la placa.

Para realizar el diseño de la PCB, primero es necesario hacer el esquemático del circuito con todas las conexiones correspondientes.

En la parte de la derecha tenemos un menú donde se pueden editar los nombres de los componentes y de las conexiones. En la ventana flotante tenemos herramientas para hacer las conexiones entre los componentes.



Al terminar el esquemático, es necesario guardar el proyecto, posteriormente en el menú de Diseño>Convertir esquemático a PCB, donde se abrirá un nuevo archivo donde indicaremos las dimensiones de la placa a realizar y aparecerán todos los componentes que agregamos en el esquemático.

En esta nueva ventana se debe acomodar todos los componentes de tal manera que las líneas estén lo menos torcidas posibles, y después se puede utilizar la opción de auto ruteo que se encuentra en el menú Route>Auto route, ahí podemos elegir el tamaño de las pistas y la distancia entre ellas, o se puede hacer el ruteo manualmente.

Al terminar el diseño de la placa se deben exportar las capas como PDF para ser impresas, se recomienda que la capa superior donde vendrán los nombres de los componentes se exporte en forma de espejo, con la finalidad de poder centrarla con la capa inferior (si se desea poner), además ambas se deben exportar en color blanco y negro.

2.10.2 Elaboración de PCB casera

Existen diferentes métodos para realizar una PCB de forma casera, uno de los más comunes es mediante el uso del cloruro férrico.

Para realizar la PCB se deben seguir los siguientes pasos:

1. Imprimir el diseño en papel brillante (papel couché o fotográfico), es importante que se haga en una impresora de tóner ya que, si no se realiza en estas, la tinta se va a correr y no se imprimirá correctamente.
2. Se procede a limpiar y preparar la placa donde se debe pegar el diseño, para esto se necesita una placa fenólica y un par de lijas lo más finas posibles, se debe lijar hasta que la placa quede brillante, posteriormente se necesita limpiar con un trapo húmedo para quitar los residuos.
3. Se debe colocar y fijar el circuito impreso sobre la placa (se puede colocar un poco de cinta en los alrededores para que no se levante ni se mueva), y con la ayuda de una plancha se debe calentar la hoja donde está la impresión para que la tinta se transfiera a la placa. Este procedimiento se debe realizar hasta que el circuito quede totalmente pegado (10 a 15 min).
4. La placa se necesita sumergir en agua para poder quitar todos los residuos de papel y que solamente quede el circuito impreso sobre la placa. Al realizar este procedimiento, posiblemente algunas partes de las pistas podrían levantarse por lo que se pueden corregir con un plumón indeleble.
5. Posteriormente la placa debe ser sumergida en cloruro férrico por completo, es necesario mover el recipiente donde se encuentra hasta lograr ver que el cobre de la placa se disolvió y solo quedan las pistas con la tinta, una vez comprobado esto, se debe limpiar la placa con agua hasta que se quiten toda la tinta.
6. El último paso es realizar con el taladro y una broca de carburo de tungsteno los hoyos correspondientes para los componentes y soldar los componentes en su respectivo lugar (mieletrónicafacil, 2022). Además, se puede agregar una capa superior con los nombres y posiciones de los componentes realizando el paso 3 en la cara correspondiente.

CAPÍTULO 3. DESARROLLO EXPERIMENTAL

En este apartado se describe el desarrollo de las incorporaciones de funcionalidades al SAAC mencionadas en el capítulo 1 de este trabajo. En síntesis, el propósito del presente trabajo es adicionar un módulo para la recepción y envío de códigos de controles remotos, mediante radiación infrarroja, con la finalidad de que el usuario pueda controlar aparatos electrónicos del hogar, lo cual implica que además del módulo, se ha tenido que trabajar en la adecuación del software del SAAC, agregando plantillas y programación para permitir que el sistema pueda “aprender” los códigos de cualquier control remoto y posterior utilización; todo esto, manteniendo un esquema de manejo y estilo igual al del SAAC del que se parte. Otra incorporación (que solo repercute en el software) es la de proporcionar órdenes a los asistentes virtuales de Alexa y Google Assistant.

En la figura 3.1 se muestra el diagrama de bloques general del SAAC con las incorporaciones propuestas.

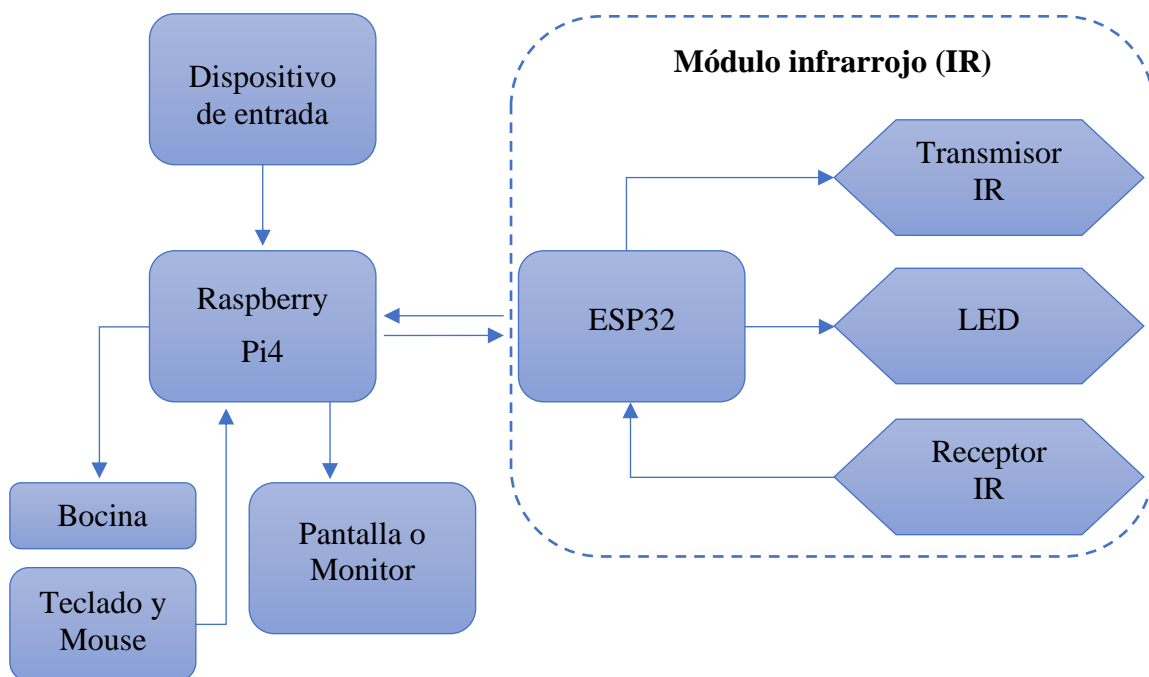


Figura 3.1 Diagrama de bloques general.

Raspberry Pi4: computador principal en el que se encuentra instalado el SAAC, es donde se controlan los elementos de entrada y salida para su funcionamiento.

Dispositivo de entrada: es cualquiera de los módulos de interacción hombre-máquina con los que se controla el SAAC, y que su funcionamiento consiste en el envío de un pulso eléctrico a petición del usuario.

Pantalla o monitor: dispositivo de visualización, en el cual se puede observar el SAAC.



ESP32: microcontrolador responsable de controlar la recepción y decodificación de señales infrarrojas, y transmisión de esta información hacia la Raspberry para su almacenamiento, también se utiliza para la transmisión de señales infrarrojas hacia aparatos electrónicos, la información de estas señales proviene desde la Raspberry, es el componente principal del módulo IR.

Transmisor IR: LED IR que se encarga de la transmisión de códigos provenientes del ESP32.

LED: dispositivo de visualización el cual nos indica en qué momento el módulo IR está en modo aprendizaje, y el lugar a donde debe ser apuntado el control remoto para obtener una lectura correcta.

Receptor IR: dispositivo encargado de recibir las señales infrarrojas y transmitir las al ESP32.

3.1 Módulo IR

En este apartado se describe todos los elementos que conforman al módulo IR, el cual se usa para la recepción y transmisión de códigos por medio de señales infrarrojas, así como la codificación utilizada para lograr esto, recordando que este módulo es independiente del SAAC, pero su funcionamiento depende totalmente de las instrucciones enviadas por él.

Se utilizó una conexión serial USB para la comunicación entre el ESP32 y la Raspberry Pi, utilizando únicamente el cable de alimentación USB para ello, lo que permitió la alimentación del microcontrolador ESP32 y la comunicación entre ambos.

3.1.1 Diagrama esquemático

En la figura 3.2 se muestra el diagrama esquemático del módulo IR.

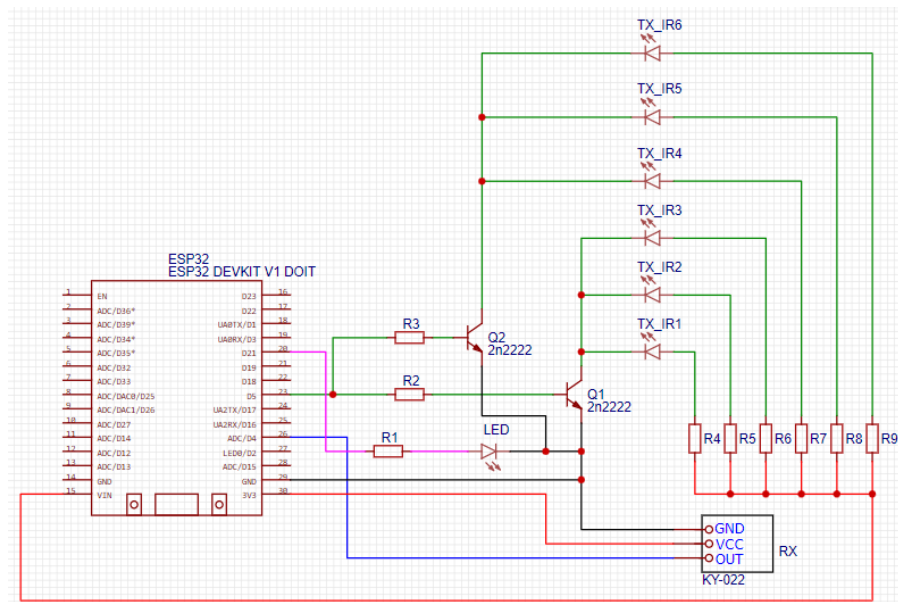


Figura 3.2 Diagrama esquemático del módulo IR.



ESP32: es el microcontrolador en el cual se cargó el código, para este se utilizó la librería IRremote modificada, la cual nos ayuda para el control de las señales infrarrojas, todo esto, programado en el software Arduino IDE.

R1: resistencia de 220Ω , limita la corriente que llega al LED de visualización.

R2 y R3: resistencias de 220Ω , conectadas en el pin 5 del ESP32 hacia la base de los transistores, las cuales limitan las corrientes que llegan a los transistores, y así, controlar el cambio de estado de corte a saturación y viceversa para estos.

R4 – R9: resistencias de 100Ω , conectadas en el pin V_{in} de 5 v del ESP32 hacia los cátodos de los LED IR, estas limitan la corriente que llega a ellos.

LED: dispositivo de visualización de color rojo, el cual nos indica si el sistema se encuentra en modo aprendizaje y hacia donde se debe dirigir el control remoto para su aprendizaje.

Q1 y Q2: transistores 2n2222, los cuales funcionan en modo interruptor para el envío de señales infrarrojas, se utilizaron dos debido al flujo de corriente que demandan los seis LED, dividiendo así el trabajo en dos transistores de bajo costo.

TX_IR1 – TX_IR6: LED IR, modelo IR383, para la transmisión de las señales. Se utilizaron seis LED, ya que la cobertura de un solo LED es de 50° de manera horizontal y 20° de manera vertical, y considerando que el usuario se encontrará en un cuarto con diferentes aparatos electrónicos que estarán posicionados en lugares distintos, se tomó la decisión de utilizar seis transmisores; con los primeros tres (apuntando de manera horizontal), se tiene una cobertura de aproximadamente 180° de manera horizontal, y colocando los otros tres restantes en un ángulo de 45° con respecto a los primeros tres, se obtuvo una amplitud de transmisión de aproximadamente 75° en vertical, logrando una cobertura deseada.

RX: receptor IR, modelo KY-022, el cual está conectado al pin de 3 V para su alimentación, el pin de tierra común y el pin de datos va conectado al pin 4 del ESP32, para la recepción de los códigos de las señales infrarrojas provenientes de controles remotos de los aparatos electrónicos.

3.1.2 Diagrama de flujo

Para controlar la recepción y transmisión de las señales infrarrojas se usó una placa ESP32 WROOM de 32 pines, para la cual se realizó un programa utilizando el software Arduino IDE, donde se instalaron las bibliotecas correspondientes para la utilización de la placa ESP32 y para el control de señales infrarrojas (recepción y transmisión).

A continuación, en la figura 3.3 se muestra el diagrama de flujo que describe el comportamiento del módulo IR para su posterior programación.

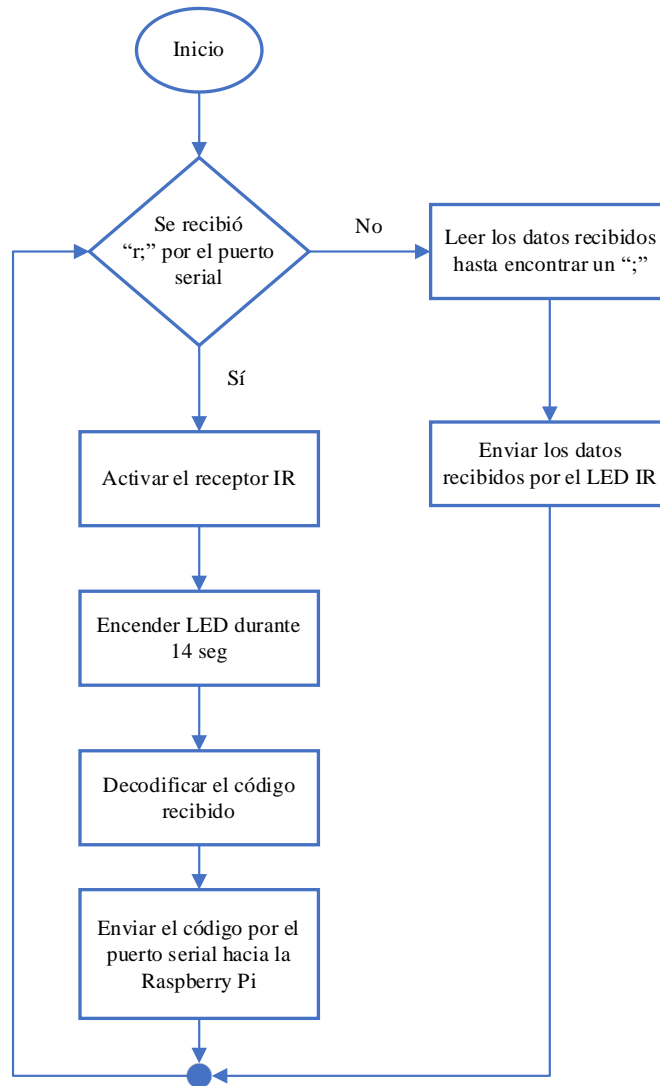


Figura 3.3 Diagrama de flujo del módulo IR.

Una vez definido el diagrama de flujo, se procedió a la programación de este módulo. Como primer paso se incluyó una biblioteca que facilita el trabajo con señales de control remoto por infrarrojo (línea 1), se configuraron los pines correspondientes para la recepción y transmisión las señales, y se crearon variables, entre ellas *decode_results*, para la obtención de los resultados de las señales recibidas.

```
1  #include <IRremote.h>
2
3  IRrecv irRecibo(4);
4  IRsend irEnvio(5);
5  decode_results resultados;
6
7  #define led 21
8  int tamCod=0;
9  String aux="";
10 unsigned int dato;
```



```
11 unsigned int codigo[100];
12 int freq=36;
13 String valor="";
```

Declaradas las variables, se inicializó la comunicación serial a una velocidad de 115200 baudios, se extendió el tamaño en el buffer de recepción del serial a 1024 localidades, ya que, al recibir una gran cantidad de valores provenientes de la Raspberry por el puerto serial, el buffer perdía datos por no tener suficiente espacio para alojarlos, por lo que se decidió expandirlo a su máximo; además se configuró al pin led como salida.

```
void setup()
16 {
17   Serial.begin(115200);
18   Serial.setRxBufferSize(1024);
19   pinMode(led,OUTPUT);
20 }
```

En la función *loop()* se realizan lecturas de lo que se recibe por el puerto serial, donde se hace la comparación que se observa en el diagrama de flujo, esto determinará el modo en el que el módulo va a trabajar (aprendizaje o transmisión).

Esto se logró con las funciones de la biblioteca serial, determinando como punto de parada de la lectura hasta el “;”, y comparándolo con los caracteres recibidos.

```
35 void loop()
36 {
37   while (Serial.available()>0)
38   {
39     String valor=Serial.readStringUntil(';');
40     if (valor== "r")
41     {
51     }
52     else
53     {
74     }
75 }
```

Identificada la instrucción recibida en el puerto serial desde la Raspberry, se realiza la recepción de una señal infrarroja en caso de obtener una “r;”. Para esto, primeramente, se habilita el receptor IR (línea 42), después se enciende un LED por 14 segundos, durante este tiempo el usuario debe apuntar el control remoto hacia el LED (que se encuentra en el módulo IR) y presionar el botón que se desea guardar el código, al realizar esta acción, la información recibida se guardará en un buffer definido por la función correspondiente de la biblioteca IR.

Se revisa una condición (línea 46) en la cual, si es cierta (true) se procede a la lectura y decodificación del código recibido (línea 48), y después de esto se vuelve a activar el receptor IR. En caso contrario, no se realizará ninguna acción y volverá a la función previa.

```
40   if (valor== "r")
41   {
42     irRecibo.enableIRIn();
```




```
43     digitalWrite(led,HIGH);
44     delay(14000);
45     digitalWrite(led,LOW);
46     if (irRecibo.decode(&resultados))
47     {
48         lectura(&resultados);
49         irRecibo.resume();
50     }
51 }
```

Al ingresar a la función *lectura()*, primero se obtiene la cantidad de datos en el buffer de recepción (línea 24), después se ejecuta un ciclo para transmitir hacia la Raspberry cada dato del código recibido seguido de una coma; esto se realiza hasta transmitir toda la información, concluyendo con la cadena “1000;”.

```
22 void lectura(decode_results *resultados)
23 {
24     int cont = resultados->rawlen;
25     for (int i=1; i<cont; i++)
26     {
27         int valor=(resultados->rawbuf[i]*USECPERTICK);
28         Serial.print(valor);
29         Serial.print(",");
30     }
31     Serial.print("1000");
32     Serial.print(";");
33 }
```

En caso de se reciba en el puerto serial un valor diferente a “r;”, significa que el módulo está en modo de transmisión, por lo que se procede a obtener la cantidad de datos recibidos en el puerto (línea 54), y se ingresa a un ciclo en el cual se hace la lectura de cada dato recibido.

En la lectura se realiza una comparación de caracteres, con la finalidad de obtener y separar los valores de cada trama, al encontrar una “;” se guarda el valor obtenido en un array (línea 63), pero para esto, previamente se convierte en una variable tipo int (línea 62), ya que los valores obtenidos por el puerto serial son de tipo string, y se tiene un contador que nos indica el índice donde se van guardando dentro del array (línea 65). Si la comparación no coincide, se concatenan los valores obtenidos (línea 69) para formar el valor de la trama.

Una vez concluido el ciclo, el contador se incrementa en uno para saber el tamaño final del array, y se procede al envío de la señal obtenida (línea 73), en el que se tiene como argumentos el código (array formado), el tamaño del código (tamaño del array), y la frecuencia (variable definida al inicio). Al realizar esta función el receptor automáticamente se deshabilita, por lo que es necesario habilitarlo si se desea usar.

```
52     else
53     {
54         int tamStr=valor.length();
55         tamCod=0;
56         aux="";
57         for(int i=0;i<tamStr;i++)
58         {
```

```
59     char byt=valor[i];
60     if(byt==' ')
61     {
62         dato=aux.toInt();
63         codigo[tamCod]=dato;
64         aux="";
65         tamCod++;
66     }
67     else
68     {
69         aux+=valor[i];
70     }
71 }
72 tamCod++;
73 irEnvio.sendRaw(codigo,tamCod,freq);
74 }
```

Terminada la codificación del diagrama de flujo, se compiló y se procedió a cargarlo a la placa ESP32, posteriormente se procedió a colocar la tarjeta ESP32 en una PCB. El código completo se encuentra en el Anexo A.

3.1.3 PCB

El diseño para la placa del módulo IR se realizó usando la aplicación de EasyEDA. Ya realizado el esquemático, se procedió a crear una PCB, en la cual se acomodaron todos los componentes en una placa de 70 x 70 mm. Además, se configuró el software para que las áreas sin pistas, funcionasen como el común de la placa (tierra), se puede observar el diseño en la figura 3.4.

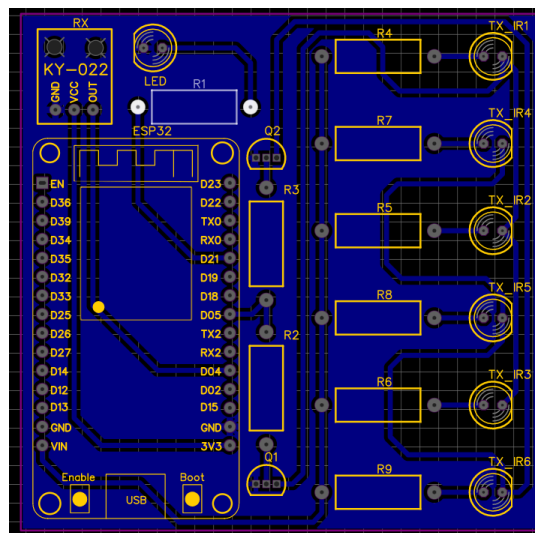


Figura 3.4 Diseño de la placa PCB para el módulo IR.

Concluido el diseño, se procedió a exportar los negativos de ambas caras, como se observa en la figura 3.5, con ellos se realizó el procedimiento correspondiente para generar las pistas en una placa fenólica de manera casera.

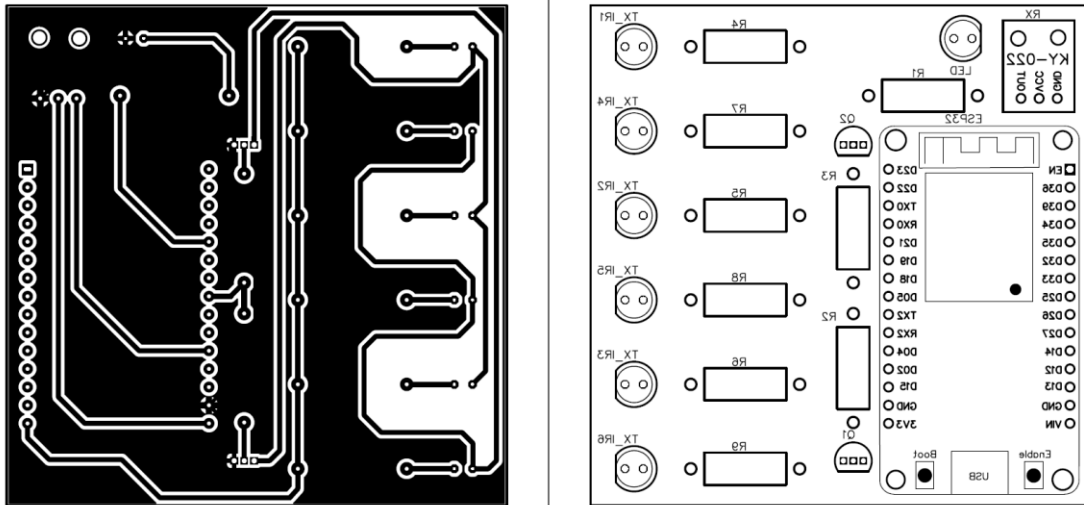


Figura 3.5 Negativos del circuito, parte inferior y superior.

Terminada la placa, se procedió a soldar dispositivos y colocar pines hembra para inserción de componentes, con la finalidad de poder reemplazar fácilmente a los más susceptibles a dañarse o que requieren mayor flexibilidad en su colocación (figura 3.6).

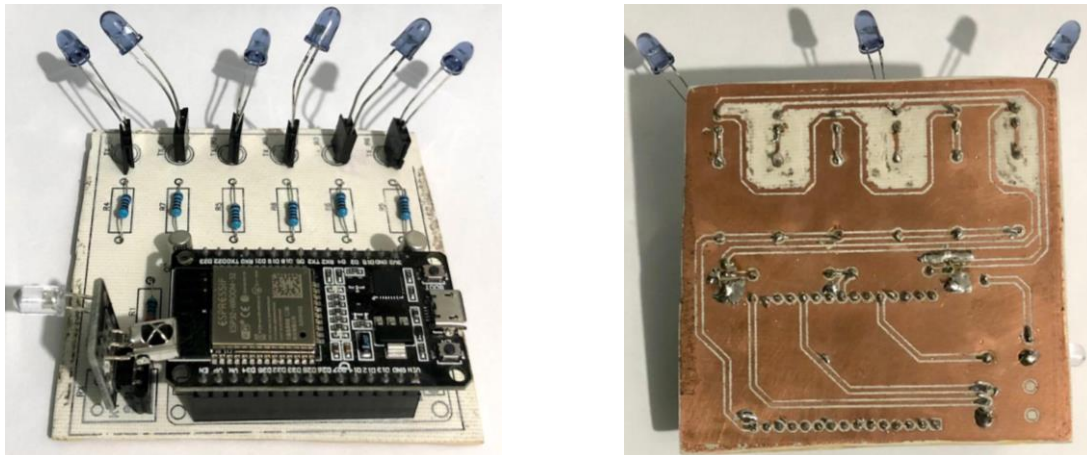


Figura 3.6 PCB de módulo IR.

3.2 Interfaz para módulo IR

Con el módulo IR terminado, se procedió a la incorporación de este al SAAC; en este apartado se explica la codificación desarrollada en Python para su utilización.

Se realizaron plantillas nuevas y se modificaron plantillas ya existentes del SAAC para el nuevo menú de domótica, un ejemplo de estas se observa en la figura 3.7, para conocer el total de plantillas creadas y modificadas véase el Anexo B.

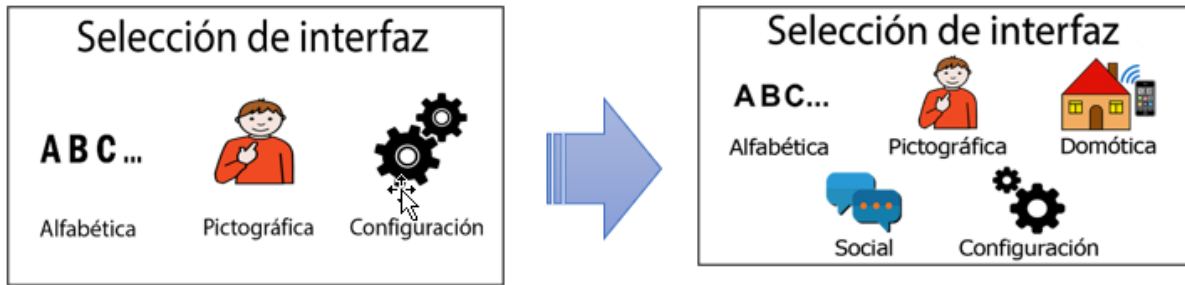


Figura 3.7 Rediseño de plantilla inicial del SAAC.

Se dividió en dos modos esta incorporación, el primero es de configuración, donde se reciben los códigos infrarrojos, y son guardados dentro de la Raspberry; el segundo es de operación, donde los códigos son enviados a la ESP32 para su transmisión. Todos esto se realizó respetando la forma de trabajo del SAAC original.

Se importaron todas las nuevas plantillas dentro del sistema para ser usadas posteriormente, además se creó un nuevo subdirectorio (carpeta) donde se guardan los códigos infrarrojos (línea 349). A continuación, se muestra una porción de código, donde se cargan las nuevas plantillas:

```

348 ##### IMAGENES PARA DOMOTICA #####
349 UbicacionDom="/home/pi/Documents/SistemaAumentativoTouch_DOMOTICA_ESP32Rutinas/codigosIR/"
350 Domotica=pygame.image.load(Ubicacion+"SeleccionInterfazDomotica.png").convert_alpha()
351 Asistentes=pygame.image.load(Ubicacion+"SeleccionAsistente.png").convert_alpha()
352 Dispositivos1=pygame.image.load(Ubicacion+"Dispositivos1.png").convert_alpha()

```

Estas plantillas fueron realizadas respetando el mismo diseño del SAAC, utilizando imágenes del mismo formato y guardando la homogeneidad del sistema, elaboradas bajo una resolución de 797 x 413 pixeles.

3.2.1 Mapa de navegación para configuración de control de dispositivos (aparatos electrónicos) mediante infrarrojo

En este apartado se explica el menú de configuración de “Dispositivos”, que se encuentra dentro del menú “Domótica”, donde se tienen las instrucciones para el aprendizaje de códigos infrarrojos desde la ESP32, los cuales se almacenan en un archivo de texto (.txt) para su posterior lectura.

En la figura 3.8 se muestra el mapa de navegación para este modo.

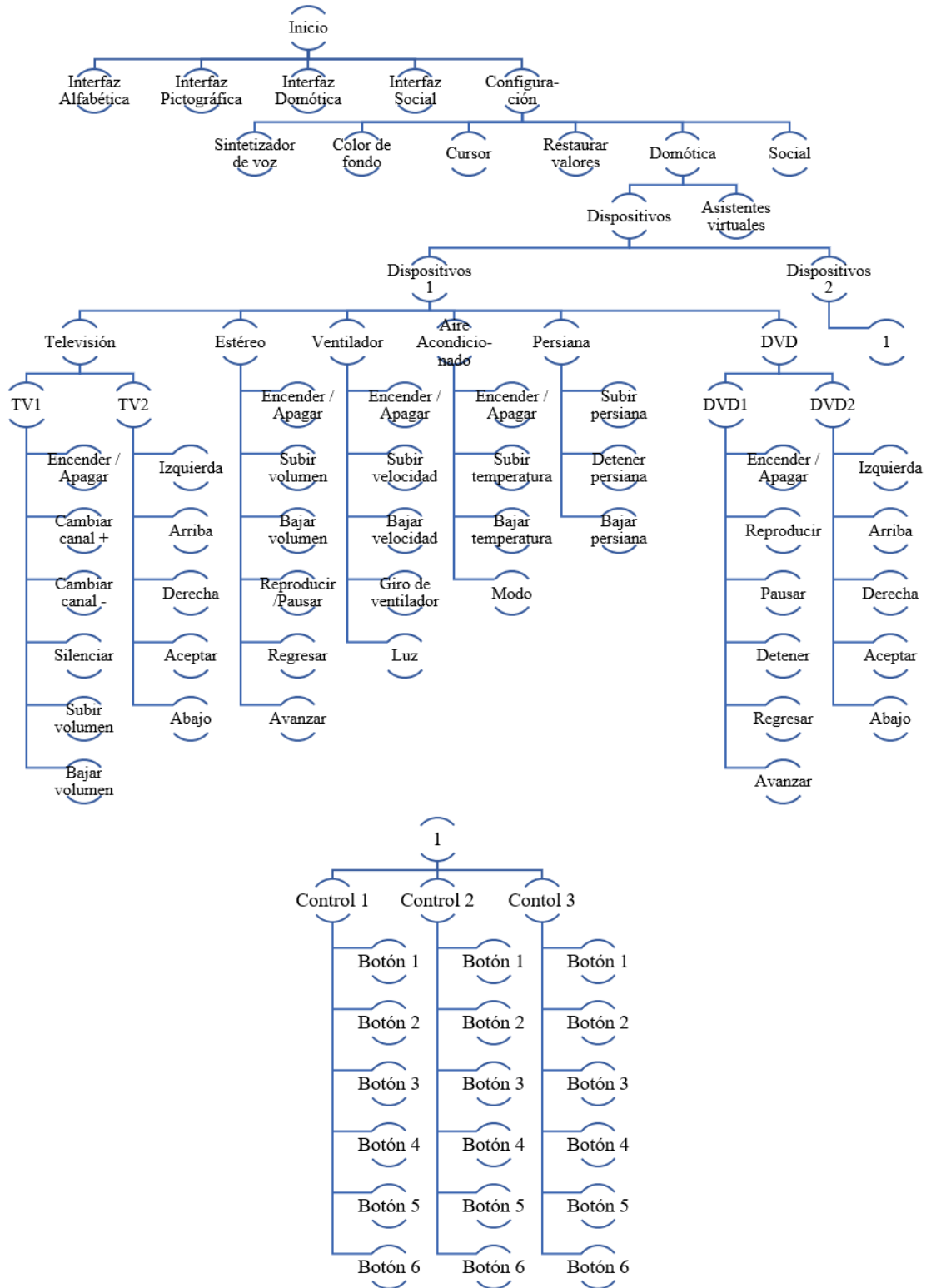


Figura 3.8 Mapa de navegación menú Domótica>Dispositivos>Configuración.



Para la navegación entre los menús se utilizó la función *seleccion(plantilla, cuadro, posx, posy)*, donde se modificó el nombre de cada plantilla y el recorrido en “x” y “y” para las posiciones de los botones, y la función *TiempoEspera(Interfaz_A_Desplegar)*, donde se modificó la función a realizar, recordando que estás dos funciones ya se encontraban definidas dentro del SAAC.

Cada una de las plantillas cuenta con dos o tres iconos de navegación, los cuales nos permiten movernos entre los menús para avanzar, regresar o saltar directamente al inicio (véase figura 3.9).



Figura 3.9 Iconos de navegación.

En el mapa de navegación contamos con seis menús que visualmente no existen dentro del SAAC, los cuales son “Dispositivos 1 y 2, TV 1 y 2, y DVD 1 y 2”, estos corresponden a plantillas que tienen una continuación; es decir, en ellas contamos con los tres iconos de navegación, donde el tercer icono es para desplazarnos a la continuación de la plantilla (véase figura 3.10 y 3.11).

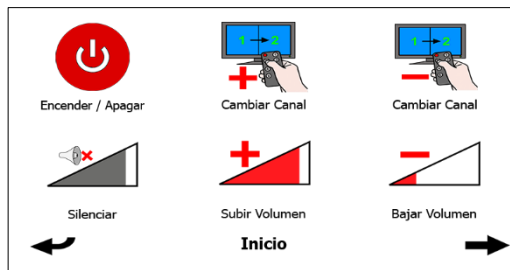


Figura 3.10 Plantilla TV1.

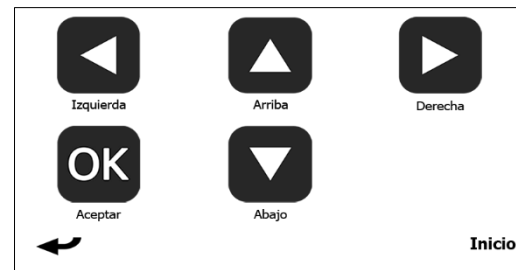


Figura 3.11 Plantilla TV2.

Al seleccionar el menú “Dispositivos”, se realiza una revisión de la conexión entre la Raspberry Pi y la ESP32 (línea 3297), si no encuentra la conexión, se muestra una imagen que indica el error en la conexión (línea 3300) y regresa al menú de configuración domótica. Cabe mencionar que de no existir la conexión lo que se debe realizar es desconectar y conectar el cable de alimentación de la ESP32 y volver a ingresar al menú.

Si la conexión existe, ingresa a la función *ConfDispositivos()* (línea 3299).

```

3289     def ConfiguracionDomotica():
3289         while True:
3290             CargarParametros()
3291             seleccion(ConfDomotica, CuaAlfa, 135, 0)
3292             x=0
3293             while x <= VelocidadCursor :
3294                 if GPIO.input(21) == GPIO.HIGH:
3295                     sonidoSeleccion()
3296                     try:
3297                         with
serial.Serial("/dev/ttyUSB0", 115200, timeout=1) as esp32:
3298                             time.sleep(0.1)
3299                             ConfDispositivos()
3300                     except OSError:
3301                         print("OS")

```



```
3302         screen.fill(ColorDeFondo)
3303         screen.blit(ConexionModulo, (0, 0))
3304         pygame.display.flip()
3305         union = ("espeak "+ Idioma + Tono + " '" +
"No hay conexión con el módulo infrarrojo, revisa tu conexión" + "' --
stdout|aplay")
3306         os.system(union)
3307         time.sleep(3)
3308         ConfiguracionDomotica()
3309         time.sleep(0.01)
3310         x=x+10
3311
3312         seleccion(ConfDomotica,CuaAlfa,400,0)
3313         TiempoEspera(ConfAsistentes)
3314
3315         seleccion(ConfDomotica,CuadritoPeque,0,0)
3316         TiempoEspera(interfazConfiguracion)
3317
3318         seleccion(ConfDomotica,CuadritoPeque,673,0)
3319         TiempoEspera(Selecciondeinterfaz)
```

El menú “Dispositivos” se dividen en dos submenús, el submenú “Dispositivos 1”, tiene seis dispositivos electrónicos predefinidos, con botones predeterminados y el submenú “Dispositivos 2”, tiene tres controles editables, donde se puede configurar el nombre tanto del control como el de los seis botones de cada uno.

En el submenú “Dispositivos 1”, el modo de configuración en los botones de todos los controles es el mismo, donde solamente se diferencia dentro de la programación, en el argumento que se da al llamar cada función.

Al seleccionar un botón, la primera acción es llamar la función *sonidoSeleccion()*, esta reproduce un clic por la bocina, posteriormente se llama a la función *modifCodigo()*, la cual tiene como argumento el archivo de texto del botón que se desea configurar y la plantilla a la que debe regresar el sistema una vez terminado el proceso.

```
4392         def ConfTV():
4393             while True:
4394
4395                 CargarParametros()
4396                 seleccion(TV,Cuadrito,0,0)
4397                 x=0
4398                 while x <= VelocidadCursor :
4399                     if GPIO.input(21) == GPIO.HIGH:
4400                         sonidoSeleccion()
4401                         modifCodigo("onTV.txt","TV1")
4402                         time.sleep(0.01)
4403                         x=x+10
```

Al ingresar a la función *modifCodigo()* se abre el archivo de texto del botón seleccionado y se revisa si este contiene un valor previo guardado o contiene la cadena “0000;” (este valor se



ingresó al crear el archivo, debido a que si en el modo operación el usuario selecciona un botón no configurado y el archivo se encuentra totalmente vacío, el sistema queda en un bucle infinito).

Si el valor es diferente a “0000;” entonces se muestra y escucha un aviso el cual nos indica que existe un código guardado previamente y si se desea modificarlo (líneas 4271 y 4273), en caso de querer modificarlo se procede a la siguiente función (línea 4278), y en caso negativo se regresa a la plantilla del dispositivo al que pertenece ese botón (línea 4287).

Si el valor leído en el archivo es igual a “0000;” entonces procede a la siguiente función (línea 4265 y 4267).

```
4258     def modifCodigo (archivo, plantilla):
4259         abrirDoc = open (UbicacionDom+archivo, "r")
4260         lecturaDeLinea=abrirDoc.readlines ()
4261         abrirDoc.close ()
4262         valor=conversion (lecturaDeLinea)
4263         print (valor)
4264         if valor=="0000;\n":
4265             reciboCodigo (archivo, plantilla)
4266         elif valor=="0000;":
4267             reciboCodigo (archivo, plantilla)
4268         else:
4269             while True:
4270                 CargarParametros ()
4271                 union = ("espeak "+ Idioma + Tono + " '" +
4272 "¿Desea modificar el código existente?" + "' --stdout|aplay")
4273                 os.system (union)
4274                 seleccion (ModificarCod, CuaRepetirFraseInicio, -
405, -32)
4275                 x=0
4276                 while x <= VelocidadCursor :
4277                     if GPIO.input (21) == GPIO.HIGH:
4278                         sonidoSeleccion ()
4279                         reciboCodigo (archivo, plantilla)
4280                         time.sleep (0.01)
4281                         x=x+10
4282                 seleccion (ModificarCod, CuaRepetirFraseInicio, 10, -
32)
4283                 x=0
4284                 while x <= VelocidadCursor :
4285                     if GPIO.input (21) == GPIO.HIGH:
4286                         sonidoSeleccion ()
4287                         plantillaAnteriorDom (plantilla)
4288                         time.sleep (0.01)
4289                         x=x+10
```

La función que se utiliza para el regreso de plantilla se llama *plantillaAnteriorDom()*, donde se hace una comparación de caracteres con el argumento, y al encontrar alguna coincidencia, llama a la función correspondiente como se muestra en el siguiente fragmento de código.

```
4366     def plantillaAnteriorDom (plantilla):
4367         if plantilla=="TV1":
```




```
4368         ConfTV ()
4369     elif plantilla=="TV2":
4370         ConfTV2 ()
4371     elif plantilla=="AireA":
4372         ConfAireAcondicionado ()
```

Una vez llamada la función *reciboCodigo()*, se revisa nuevamente la conexión con la ESP32 (línea 4292) y se envía la cadena “r;” (línea 4295) para indicar al módulo IR que opere en modo aprendizaje. Se muestra la plantilla que nos muestra las instrucciones para guardar el código del botón (línea 4298), además se visualiza un temporizador el cual nos indica el tiempo que tiene el usuario para presionar el botón del control remoto (líneas 4301-4330).

Una vez terminado el temporizador se revisa si se recibió un código por el puerto serial (línea 4331) y se guarda en el archivo de texto correspondiente. En caso de no recibir ningún valor, se muestra y se reproduce la plantilla que nos indica que no se recibió el código y es necesario repetir el proceso nuevamente (línea 4349 y 4351).

```
4291     def reciboCodigo (archivoTXT,plantilla):
4292         with serial.Serial ("/dev/ttyUSB0", 115200,timeout=1) as
esp32:
4293             time.sleep(0.1)
4294             mando="r;"
4295             esp32.write (mando.encode ())
4296             time.sleep(0.1)
4297             screen.fill (ColorDeFondo)
4298             screen.blit (RecibiendoCod, (0, 0))
4299             pygame.display.flip ()
4300             while True:
4301                 contador ("15")
4302                 time.sleep (1)
4303                 contador ("14")
4304                 time.sleep (1)
4305                 contador ("13")
4306                 time.sleep (1)
4307                 contador ("12")
4308                 time.sleep (1)
4309                 contador ("11")
4310                 time.sleep (1)
4311                 contador ("10")
4312                 time.sleep (1)
4313                 contador ("9")
4314                 time.sleep (1)
4315                 contador ("8")
4316                 time.sleep (1)
4317                 contador ("7")
4318                 time.sleep (1)
4319                 contador ("6")
4320                 time.sleep (1)
4321                 contador ("5")
4322                 time.sleep (1)
4323                 contador ("4")
4324                 time.sleep (1)
4325                 contador ("3")
```



```

4326         time.sleep(1)
4327         contador("2")
4328         time.sleep(1)
4329         contador("1")
4330         time.sleep(1)
4331         if esp32.inWaiting()>0:
4332             answer=esp32.readline()
4333             value=(answer.decode("utf-8"))
4334             txt = open(UbicacionDom+archivoTXT, 'w')
4335             txt.writelines(str(value))
4336             print(str(value))
4337             txt.close()
4338             screen.fill(ColorDeFondo)
4339             fuente=pygame.font.Font(None,100)
4340             frase = fuente.render("Código guardado", 1,
(0, 0, 0))
4341             screen.blit(frase, (120,180))
4342             pygame.display.flip()
4343             union = ("espeak "+ Idioma + Tono + "Código
guardado" + "' --stdout|aplay")
4344             os.system(union)
4345             time.sleep(2)
4346             plantillaAnteriorDom(plantilla)
4347         else:
4348             screen.fill(ColorDeFondo)
4349             screen.blit(NoRecibiCod, (0, 0))
4350             pygame.display.flip()
4351             union = ("espeak "+ Idioma + Tono + " '" +
"No se recibió código, favor de intentarlo de nuevo" + "' --
stdout|aplay")
4352             os.system(union)
4353             time.sleep(2)
4354             plantillaAnteriorDom(plantilla)

```

La función *contador()* se utiliza para crear el temporizador visual el cual despliega un recuadro que muestra el valor recibido como argumento.

```

4356         def contador(numero) :
4357             font=pygame.font.Font(None, 30)
4358             cuaConta=pygame.Rect(720,350,40,40)
4359             titulo=font.render(numero,True,(0,0,0))
4360             pygame.draw.rect(screen,(255,255,255),cuaConta)
4361             pygame.draw.rect(screen,(0,0,0),cuaConta,1)
4362             screen.blit(titulo,(cuaConta.x+12,cuaConta.y+10))
4363             pygame.display.flip()

```

En el submenú “Dispositivos 2”, el modo de configuración es similar al “Dispositivos 1”, ya que se llaman a las mismas funciones en momentos distintos debido a la interacción de otras funciones, las cuales permiten al usuario a editar el nombre para los controles y para los botones, para este proceso es necesario tener conectado un teclado externo en la Raspberry.

El recorrido en el menú dispositivos cambia, en este recorrido se llama a la función *vistaNombresControles()*, la cual muestra el nombre asignado a cada control en la interfaz, esto



debido a que esta plantilla no contiene los nombres de los controles (véase figura 3.13) a diferencia de los del menú “Dispositivos 1” (véase figura 3.12).



Figura 3.12 Plantilla Dispositivos 1.

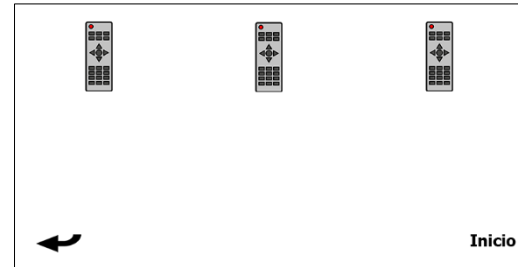


Figura 3.13 Plantilla Dispositivos 2.

En la función *vistaNombresControles()*, se abre el archivo *nombresControles.txt* en el que están los nombres predefinidos para cada control (los cuales pueden ser modificados por el usuario), se leen los nombres de cada control y se muestran en la interfaz con la ayuda de un ciclo en el cual se le asigna a cada uno su posición.

```

3396     def vistaNombresControles():
3397         posicion=70
3398         for i in range(3):
3399             fuente = pygame.font.Font(None,40)
3400             abrirDoc = open(Ubicacion+"nombresControles.txt",
3401 "r")
3401             lecturaDeLinea=abrirDoc.readlines()[i].strip()
3402             abrirDoc.close()
3403             frase= fuente.render(lecturaDeLinea, 1, (Negro))
3404             if i==0:
3405                 screen.blit(frase, (posicion,140))
3406             elif i==1:
3407                 screen.blit(frase, (posicion+260,140))
3408             elif i==2:
3409                 screen.blit(frase, (posicion+530,140))
3410             i+=1
3411             pygame.display.update()

```

Al seleccionar un control para configurar sus botones, primeramente, se llama a la función *cuadroInputNombreDispositivo()*, donde el usuario puede editar el nombre del control o dejar el existente. Además, se creó la función *seleccionAsis()*, que es igual a la función *seleccion()* ya existente, con la diferencia de que se eliminó la instrucción final *pygame.display.flip()*, debido a que esta función actualiza la pantalla y al tener los nombres escritos genera un parpadeo al mover el cursor.

```

3354     def ConfDispositivos2():
3355         while True:
3356             CargarParametros()
3357
3358             seleccionAsis(Dispositivos2,Cuadrado,0,0)
3359             vistaNombresControles()
3360             x=0
3361             while x <= VelocidadCursor :
3362                 if GPIO.input(21) == GPIO.HIGH:

```



```
3363         sonidoSeleccion()  
3364         cuadroInputNombreDispositivo("0",0)  
3365         time.sleep(0.01)  
3366         x=x+10
```

Cuando se manda a llamar la función *cuadroInputNombreDispositivo()*, se despliega una ventana flotante en la cual se muestra el nombre del dispositivo que ya tiene asignado, si se desea modificar se puede borrar (línea 4993-4994) y asignar un nuevo nombre (línea 4995-4996) o simplemente dar enter para confirmar y continuar (línea 7968-4992).

Al modificar el nombre, el usuario debe saber que este está limitado en número de caracteres, los cuales no deben exceder el tamaño del recuadro, ya que si este excede (línea 5004), el recuadro cambiará a color rojo (línea 5014), y no dejará realizar ninguna acción, solamente el usuario podrá borrar un carácter para poder continuar (línea 5012), estos cambios se guardan en el archivo de texto editando la línea correspondiente para el dispositivo seleccionado (es el argumento de la función llamado numLinea).

Posteriormente se ingresa al menú del control deseado, que es el argumento de la función llamado control (líneas 4976, 4984 y 4992).

```
4944     def cuadroInputNombreDispositivo(control,numLinea):  
4945         font = pygame.font.Font(None, 28)  
4946         font2 = pygame.font.Font(None, 32)  
4947         input_box = pygame.Rect(340,200, 100, 40)  
4948         box = pygame.Rect(320,150, 140, 110)  
4949         color_inactive = pygame.Color('red2')  
4950         color_active = pygame.Color('dodgerblue2')  
4951         color = color_active  
4952         active = True  
4953         textFrase = ''  
4954         done = False  
4955  
4956         output = open(Ubicacion+"nombresControles.txt", 'r')  
4957         nombre=output.readlines()[numLinea].strip()  
4958         textFrase=nombre  
4959         output.close()  
4960  
4961         while not done:  
4962             for event in pygame.event.get():  
4963                 if event.type==pygame.QUIT:  
4964                     done=True  
4965                 if event.type == pygame.KEYDOWN:  
4966                     if active:  
4967  
4968                         if event.key == pygame.K_RETURN:  
4969                             if control == "0":  
4970                                 output =  
4971                                 frases=output.readlines()  
4972                                 frases[numLinea]=textFrase+"\n"  
4973                                 output.seek(0)  
4974                                 output.writelines(frases)
```



```
4975         output.close()
4976         ConfControl1()
4977         elif control == "1":
4978             output =
open(Ubicacion+"nombresControles.txt", 'r+')
4979             frases=output.readlines()
4980             frases[numLinea]=textFrase+"\n"
4981             output.seek(0)
4982             output.writelines(frases)
4983             output.close()
4984             ConfControl2()
4985         elif control == "2":
4986             output =
open(Ubicacion+"nombresControles.txt", 'r+')
4987             frases=output.readlines()
4988             frases[numLinea]=textFrase+"\n"
4989             output.seek(0)
4990             output.writelines(frases)
4991             output.close()
4992             ConfControl3()
4993         elif event.key == pygame.K_BACKSPACE:
4994             textFrase = textFrase[:-1]
4995         else:
4996             textFrase += event.unicode
4997
4998         titulo = font2.render("Nombre", True, (0,0,0))
4999         pygame.draw.rect(screen, (234,230,202),box)
5000         pygame.draw.rect(screen, (204,202,202), (320,150,
140, 40))
5001         screen.blit(titulo, (box.x+30,box.y+10))
5002
5003         txt_surface = font.render(textFrase, True, (0,0,0))
5004         width=max(100,txt_surface.get_width()+15)
5005         if width==100:
5006             active=True
5007         else:
5008             active=False
5009             color=color_active if active else color_inactive
5010             if event.type==pygame.KEYDOWN:
5011                 if event.key==pygame.K_BACKSPACE:
5012                     textFrase = textFrase[:-1]
5013                     active=True
5014                     color=color_active
5015             screen.blit(txt_surface, (input_box.x+5,
input_box.y+10))
5016             pygame.draw.rect(screen, color, input_box, 2)
5017             pygame.display.flip()
```

Al entrar al menú del dispositivo seleccionado, se muestra la plantilla de botones, en donde, el recorrido se hace con la estructura del SAAC, pero se agregó la función *vistaNombresBotones()* que nos ayuda a visualizar los nombres asignados a cada uno, y al seleccionarlo se llama a la función *cuadroInputNombreBotones()*.



```
5136     def ConfControl1():
5137         while True:
5138             CargarParametros()
5139             seleccionAsis(Botones,Cuadrito,0,0)
5140             vistaNombresBotones("0")
5141             x=0
5142             while x <= VelocidadCursor :
5143                 if GPIO.input(21) == GPIO.HIGH:
5144                     sonidoSeleccion()
5145             cuadroInputNombreBotones("0",0,"b1Control1.txt","Control1")
5146             time.sleep(0.01)
5147             x=x+10
```

La función *vistaNombresBotones()*, tiene como argumento el número de control al que pertenece este menú, con este, se realiza una comparación en la que se indica que archivo de texto abrir, lee los valores y muestra los nombres correspondientes para cada botón en su posición.

```
5105     def vistaNombresBotones(control):
5106         for i in range(6):
5107             fuente = pygame.font.Font(None,40)
5108             if control == "0":
5109                 output = open(Ubicacion+"botonesC1.txt", 'r')
5110                 nombre=output.readlines()[i].strip()
5111                 output.close()
5112             elif control == "1":
5113                 output = open(Ubicacion+"botonesC2.txt", 'r')
5114                 nombre=output.readlines()[i].strip()
5115                 output.close()
5116             elif control == "2":
5117                 output = open(Ubicacion+"botonesC3.txt", 'r')
5118                 nombre=output.readlines()[i].strip()
5119                 output.close()
5120             frase= fuente.render(nombre, 1, (Negro))
5121             if i==0:
5122                 screen.blit(frase, (80,140))
5123             elif i==1:
5124                 screen.blit(frase, (350,140))
5125             elif i==2:
5126                 screen.blit(frase, (620,140))
5127             elif i==3:
5128                 screen.blit(frase, (80,305))
5129             elif i==4:
5130                 screen.blit(frase, (350,305))
5131             elif i==5:
5132                 screen.blit(frase, (620,305))
5133             i+=1
5134             pygame.display.update()
```

La función *cuadroInputNombres()*, trabaja de la misma forma que la función *cuadroInputNombreDispositivo()*, aunque esta función tiene dos argumentos extras que son archivo y plantilla, que se utilizan como argumentos cuando se llama a la función



modifCodigo(), la cual se llama al momento de terminar de configurar el nombre del botón (línea 5062, 5070, 5078) y que se explicó previamente.

```
5019     def
cuadroInputNombreBotones (control,numLinea,archivo,plantilla):
5020         font = pygame.font.Font (None, 28)
5021         font2 = pygame.font.Font (None, 32)
5022         input_box = pygame.Rect (340,200, 100, 40)
5023         box = pygame.Rect (320,150, 140, 110)
5024         color_inactive = pygame.Color('red2')
5025         color_active = pygame.Color('dodgerblue2')
5026         color = color_active
5027         active = True
5028         textFrase = ''
5029         done = False
5030
5031         if control == "0":
5032             output = open(Ubicacion+"botonesC1.txt", 'r')
5033             nombre=output.readlines()[numLinea].strip()
5034             textFrase=nombre
5035             output.close()
5036         elif control == "1":
5037             output = open(Ubicacion+"botonesC2.txt", 'r')
5038             nombre=output.readlines()[numLinea].strip()
5039             textFrase=nombre
5040             output.close()
5041         elif control == "2":
5042             output = open(Ubicacion+"botonesC3.txt", 'r')
5043             nombre=output.readlines()[numLinea].strip()
5044             textFrase=nombre
5045             output.close()
5046
5047         while not done:
5048             for event in pygame.event.get():
5049                 if event.type==pygame.QUIT:
5050                     done=True
5051                 if event.type == pygame.KEYDOWN:
5052                     if active:
5053
5054                         if event.key == pygame.K_RETURN:
5055                             if control == "0":
5056                                 output =
open(Ubicacion+"botonesC1.txt", 'r+')
5057                                     frases=output.readlines()
5058                                     frases[numLinea]=textFrase+"\n"
5059                                     output.seek(0)
5060                                     output.writelines(frases)
5061                                     output.close()
5062                                     modifCodigo(archivo,plantilla)
5063                             elif control == "1":
5064                                 output =
open(Ubicacion+"botonesC2.txt", 'r+')
5065                                     frases=output.readlines()
5066                                     frases[numLinea]=textFrase+"\n"
```



```
5067         output.seek(0)
5068         output.writelines(frases)
5069         output.close()
5070         modifCodigo(archivo,plantilla)
5071         elif control == "2":
5072             output =
open(Ubicacion+"botonesC3.txt", 'r+')
5073             frases=output.readlines()
5074             frases[numLinea]=textFrase+"\n"
5075             output.seek(0)
5076             output.writelines(frases)
5077             output.close()
5078             modifCodigo(archivo,plantilla)
5079         elif event.key == pygame.K_BACKSPACE:
5080             textFrase = textFrase[:-1]
5081         else:
5082             textFrase += event.unicode
5083
5084         titulo = font2.render("Nombre", True, (0,0,0))
5085         pygame.draw.rect(screen, (234,230,202),box)
5086         pygame.draw.rect(screen, (204,202,202), (320,150,
140, 40))
5087         screen.blit(titulo, (box.x+30,box.y+10))
5088
5089         txt_surface = font.render(textFrase, True, (0,0,0))
5090         width=max(100,txt_surface.get_width()+15)
5091         if width==100:
5092             active=True
5093         else:
5094             active=False
5095             color=color_active if active else color_inactive
5096             if event.type==pygame.KEYDOWN:
5097                 if event.key==pygame.K_BACKSPACE:
5098                     textFrase = textFrase[:-1]
5099                     active=True
5100                     color=color_active
5101             screen.blit(txt_surface, (input_box.x+5,
input_box.y+10))
5102             pygame.draw.rect(screen, color, input_box, 2)
5103             pygame.display.flip()
```

Estas son las funciones utilizadas para la configuración de todos los botones de los dispositivos.

3.2.2 Mapa de navegación para operación

En este apartado se explica el modo de operación, en el cual se envía el código guardado en un archivo de texto dentro de la Raspberry hacia la ESP32 y este lo transmite para realizar la acción deseada en el dispositivo electrónico.

En la figura 3.8 se muestra el mapa de navegación para este modo.



INGENIERÍA ELÉCTRICA ELECTRÓNICA

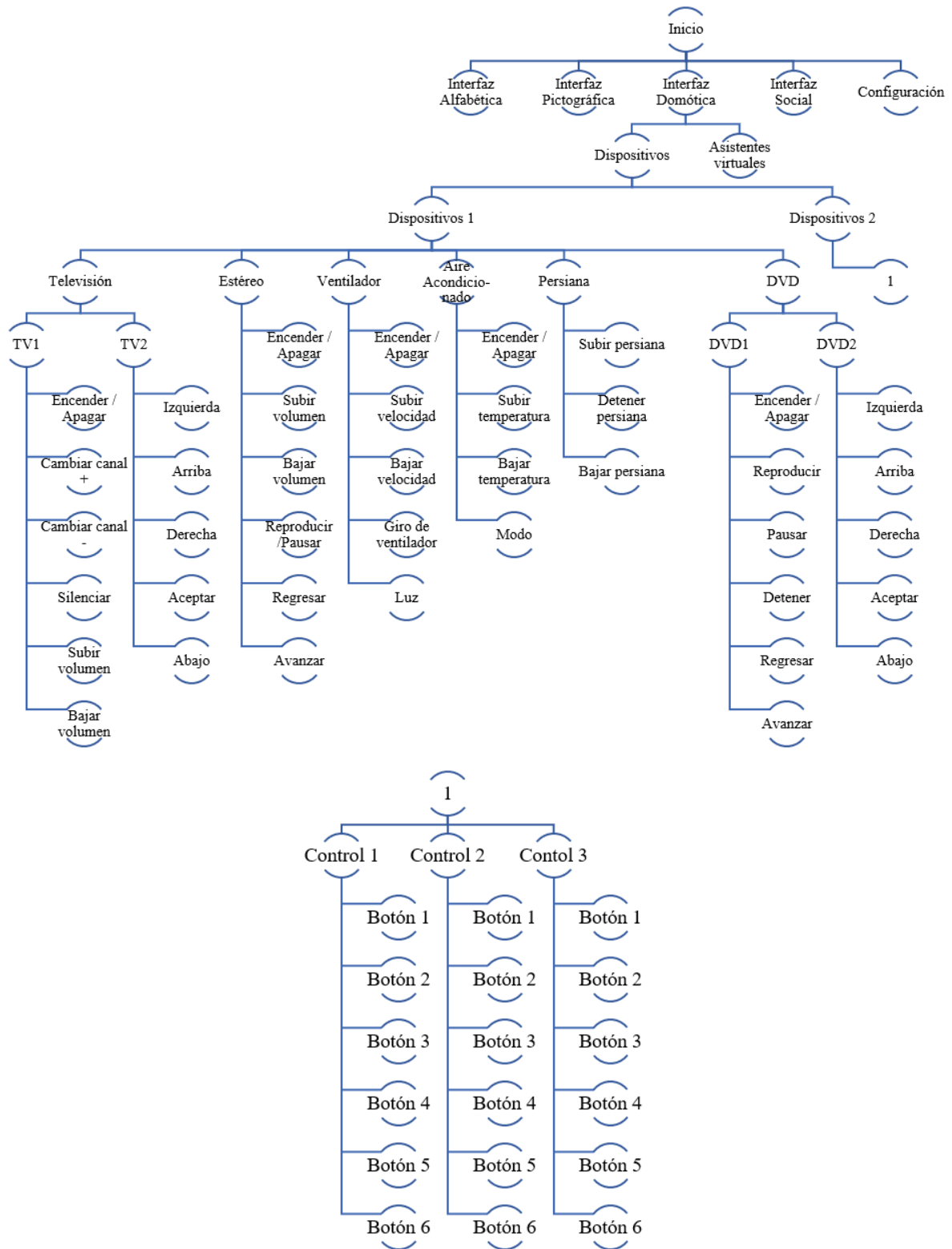


Figura 3.14 Mapa de navegación interfaz domótica-operación.



Se utiliza la misma forma de operación para la navegación entre los menús con las instrucciones mencionadas en el punto 3.2.1.

Al ingresar al menú Dispositivos del modo operación se realiza una revisión de la conexión entre la Raspberry Pi y la ESP32 (línea 1390), ya que, de no existir esta conexión, se muestra la plantilla que nos indica la falta de comunicación con el módulo IR, por lo tanto, es necesario revisar la conexión (línea 1396), posteriormente nos regresa al menú de configuración domótica (línea 1401) donde podremos volver a ingresar a este menú después de revisada la conexión.

Si la conexión existe nos permite ingresar a todos los dispositivos (línea 1392).

```
1381     def SelecciondeinterfazDomotica():
1382         while True:
1383             CargarParametros()
1384             seleccion(Domotica,CuaAlfa,135,0)
1385             x=0
1386             while x <= VelocidadCursor :
1387                 if GPIO.input(21) == GPIO.HIGH:
1388                     sonidoSeleccion()
1389                     try:
1390                         with serial.Serial("/dev/ttyUSB0",
115200,timeout=1) as esp32:
1391                             time.sleep(0.1)
1392                             interfazDispositivos()
1393                     except OSError:
1394                         print("OS")
1395                         screen.fill(ColorDeFondo)
1396                         screen.blit(ConexionModulo, (0, 0))
1397                         pygame.display.flip()
1398                         union = ("speak "+ Idioma + Tono + " '" +
"No hay conexión con el módulo infrarrojo, revisa tu conexión" + "' --
stdout|aplay")
1399                         os.system(union)
1400                         time.sleep(3)
1401                         SelecciondeinterfazDomotica()
1402                         time.sleep(0.01)
1403                         x=x+10
1404
1405                 seleccion(Domotica,CuaAlfa,400,0)
1406                 TiempoEspera(interfazAsistentes)
1407
1408                 seleccion(Domotica,CuadritoPeque,0,0)
1409                 TiempoEspera(Selecciondeinterfaz)
```

Una vez revisada la conexión se accede a los dispositivos, en el cual de la misma forma que en la configuración se divide en dos partes, “Dispositivos 1 y 2”.

En el menú “Dispositivos 1”, el modo de operación es igual para todos los botones, en el que se hace un llamado a la misma función, la diferencia es los argumentos que se envían en cada caso.

La primera acción que se realiza es reproducir en el sintetizador de voz la acción se seleccionó (línea 2245) y la segunda es enviar código hacia la ESP32 (línea 2246).



```
2237     def interfazTV():
2238         while True:
2239             CargarParametros()
2240             seleccion(TV,Cuadrito,0,0)
2241             x=0
2242             while x <= VelocidadCursor :
2243                 if GPIO.input(21) == GPIO.HIGH:
2244                     sonidoSeleccion()
2245                     vozDom("Encendiendo o apagando la Televisión")
2246                     envioCodigo("onTV.txt","TV1")
2247                     time.sleep(0.01)
2248                     x=x+10
```

Para reproducir la acción seleccionada por medio del sintetizador se manda a llamar a la función *vozDom()*, que es parecida a la función *voz()* ya existente, lo que cambia es la posición (líneas 3167, 3168) y el tamaño de la letra (línea 3164) al mostrarlo en la plantilla, esta tiene como argumento la oración que se va a reproducir.

```
3147     def vozDom(oracion):
3148         global frase
3149         global TamLet
3150         global PosYCuaTex
3151         global PosXCuaTex
3152         global SeleccionPlantillaPictografica
3153         global PlantillaNumerica
3154         global PlantillaAlfabetica
3155         global Tono
3156         global Idioma
3157         global Sonido
3158
3159         marcoNegro =
pygame.draw.rect(screen,Negro,(120,355,587,52))
3160         lienzoBlanco =
pygame.draw.rect(screen,Blanco,(122,358,583,46))
3161
3162         marcoNegro
3163         lienzoBlanco
3164         TamLet=46
3165         fuente = pygame.font.Font(None, TamLet)
3166         oracion=oracion.capitalize()
3167         PosXCuaTex=128
3168         PosYCuaTex=365
3169         mensaje = fuente.render(oracion, 1, (0, 0, 0))
3170         screen.blit(mensaje, (PosXCuaTex, PosYCuaTex))
3171         pygame.display.flip()
3172         union = ("espeak "+ Idioma + Tono + " '" + oracion + "'
--stdout|aplay")
3173         print(union)
3174         os.system(union)
3175         pygame.time.delay(VelocidadCursor)
3176         print(union)
3177         print("Fin de voz")
```



Después se llama a la función *envioCodigo()*, esta tiene como argumento el archivo de texto en el que se encuentra guardado el código para ese botón, y un segundo argumento para la plantilla a la que se regresa una vez terminado este proceso.

Se revisa la conexión entre ambos dispositivos nuevamente (línea 2191), ya que, de no existir, nos muestra el mismo anuncio mencionado antes y regresa a la plantilla en la que nos encontrábamos (línea 2209).

Al existir la conexión entre ambas placas, se abre el archivo de texto y se leen los valores en él (línea 2195), estos datos se guardan en una variable de tipo array, por lo que se procede a convertirlos a una variable tipo string (línea 2197) para su transmisión y se envían a la ESP32 (línea 2198), y una vez terminado este proceso se regresa al recorrido del menú.

```
2189     def envioCodigo (archivoTXT,plantilla):
2190         try:
2191             with serial.Serial("/dev/ttyUSB0", 115200,timeout=1)
as esp32:
2192                 time.sleep(0.1)
2193                 print("Enviando codigo")
2194                 abrirDoc = open(UbicacionDom+archivoTXT, "r")
2195                 lecturaDeLinea=abrirDoc.readlines()
2196                 abrirDoc.close()
2197                 valor=conversion(lecturaDeLinea)
2198                 esp32.write(valor.encode())
2199                 time.sleep(0.1)
2200                 print(valor)
2201             except OSError:
2202                 screen.fill(ColorDeFondo)
2203                 screen.blit(ConexionModulo, (0, 0))
2204                 pygame.display.flip()
2205                 pygame.display.flip()
2206                 union = ("espeak "+ Idioma + Tono + " '" + "No hay
conexión con el módulo infrarrojo, revisa tu conexión" + "' --
stdout|aplay")
2207                 os.system(union)
2208                 time.sleep(3)
2209                 plantillaAnteriorDomotica(plantilla)
```

La función *plantillaAnteriorDomotica()* funciona de la misma manera que la función *plantillaAnteriorDom()*, con la diferencia que las funciones llamadas al realizar la comparación son las del modo operación y no las del modo configuración.

```
2211     def plantillaAnteriorDomotica(plantilla):
2212         if plantilla=="TV1":
2213             interfazTV()
2214         elif plantilla=="TV2":
2215             interfazTV2()
2216         elif plantilla=="AireA":
2217             interfazAireAcondicionado()
```



La función *conversion()* se llama para convertir la variable, esto se realiza mediante un ciclo en el cual se lee cada valor del array y concatena todos estos en una sola variable tipo string (línea 8196). Una vez terminado el ciclo nos retorna como valor la variable convertida (línea 2187).

```
2183     def conversion(lecturaDeLinea):
2184         codigoStr=""
2185         for cad in lecturaDeLinea:
2186             codigoStr+=cad
2187         return codigoStr
```

En el caso de los “Dispositivos 2”, se trabaja de la misma forma que los “Dispositivos 1”, agregando la función *vistaNombresControles()* en los recorridos para mostrar los nombres de los controles en los “Dispositivos 2” (línea 1449).

```
1444     def interfazDispositivos2():
1445         while True:
1446             CargarParametros()
1447
1448             seleccionAsis(Dispositivos2,Cuadrito,0,0)
1449             vistaNombresControles()
1450             x=0
1451             while x <= VelocidadCursor :
1452                 if GPIO.input(21) == GPIO.HIGH:
1453                     sonidoSeleccion()
1454                     interfazControll()
1455                     time.sleep(0.01)
1456                     x=x+10
```

Al seleccionar un botón se realiza la función de *envioCodigo()*, aunque también en el recorrido se agrega la función *vistaNombresBotones()* para mostrar en la interfaz los nombres de los botones (línea 2832), además de cambiar la función para la reproducción de la voz (línea 2838).

```
2827     def interfazControll():
2828         while True:
2829
2830             CargarParametros()
2831             seleccionAsis(Botones,Cuadrito,0,0)
2832             vistaNombresBotones("0")
2833             x=0
2834             while x <= VelocidadCursor :
2835                 if GPIO.input(21) == GPIO.HIGH:
2836                     sonidoSeleccion()
2837                     envioCodigo("b1Controll.txt","Controll")
2838                     vozDomBot("0",0)
2839                     time.sleep(0.01)
2840                     x=x+10
```

La función *vozDomBot()* es prácticamente igual que la función *vozDom()*, donde el argumento cambia, ya que como es un botón que se puede modificar su nombre, esta no recibe la oración a reproducir, sino recibe el numero de control (para abrir el archivo de texto correspondiente) y el número de línea (para saber que botón se está seleccionando) para reproducir los nombres



correspondientes para cada uno, pero realiza las mismas acciones para mostrar visualmente el texto y reproducir la voz.

```
3102     def vozDomBot (control,numLinea):
3103         global frase
3104         global TamLet
3105         global PosYCuaTex
3106         global PosXCuaTex
3107         global SeleccionPlantillaPictografica
3108         global PlantillaNumerica
3109         global PlantillaAlfabetica
3110         global Tono
3111         global Idioma
3112         global Sonido
3113
3114         marcoNegro =
pygame.draw.rect (screen,Negro, (120,355,587,52))
3115         lienzoBlanco =
pygame.draw.rect (screen,Blanco, (122,358,583,46))
3116
3117         marcoNegro
3118         lienzoBlanco
3119         TamLet=46
3120         fuente = pygame.font.Font (None, TamLet)
3121         if control == "0":
3122             output = open (Ubicacion+"botonesC1.txt", 'r')
3123             oracion=output.readlines() [numLinea].strip()
3124             output.close()
3125         elif control == "1":
3126             output = open (Ubicacion+"botonesC2.txt", 'r')
3127             oracion=output.readlines() [numLinea].strip()
3128             output.close()
3129         elif control == "2":
3130             output = open (Ubicacion+"botonesC3.txt", 'r')
3131             oracion=output.readlines() [numLinea].strip()
3132             output.close()
3133             oracion=oracion.capitalize()
3134             PosXCuaTex=128
3135             PosYCuaTex=365
3136
3137             mensaje = fuente.render (oracion, 1, (0, 0, 0))
3138             screen.blit (mensaje, (PosXCuaTex, PosYCuaTex))
3139             pygame.display.flip()
3140             union = ("espeak "+ Idioma + Tono + " '" + oracion + "'
--stdout|aplay")
3141             print (union)
3142             os.system (union)
3143             pygame.time.delay (VelocidadCursor)
3144             print (union)
3145             print ("Fin de voz")
```

Estas son las funciones utilizadas para la operación de todos los botones de los dispositivos.

3.3 Interfaz para asistentes visuales

En este apartado se describe la incorporación en software para interactuar con un asistente virtual (Alexa o Google Assistant) si es que el paciente cuenta con alguno de ellos.

La programación para esta sección se realizó en Python directamente en el código del SAAC, incorporando nuevas plantillas únicamente para este menú (véase Anexo C), en donde se respetó la forma de navegar entre los menús.

Esta sección está conformada de cuatro plantillas las cuales contienen el nombre y logo del asistente (figura 3.15), en donde se desplegarán cinco frases por plantilla, dándonos un total de 20 frases por asistente, las cuales están predeterminadas dentro de un archivo de texto, pero estas pueden ser modificadas por el usuario para interactuar con ellos.



Figura 3.15 Plantilla Frases Google.

De la misma forma que la interfaz para el módulo IR, la interfaz para los asistentes virtuales se divide en dos modos, el primero donde se configuran las frases para reproducir y el segundo donde se reproducen dichas frases.

3.3.1 Mapa de navegación para configuración de frases para asistentes virtuales

En este apartado se describe el modo de configuración de frases para cada asistente. En la figura 3.16 se muestra el mapa de navegación para esta sección.

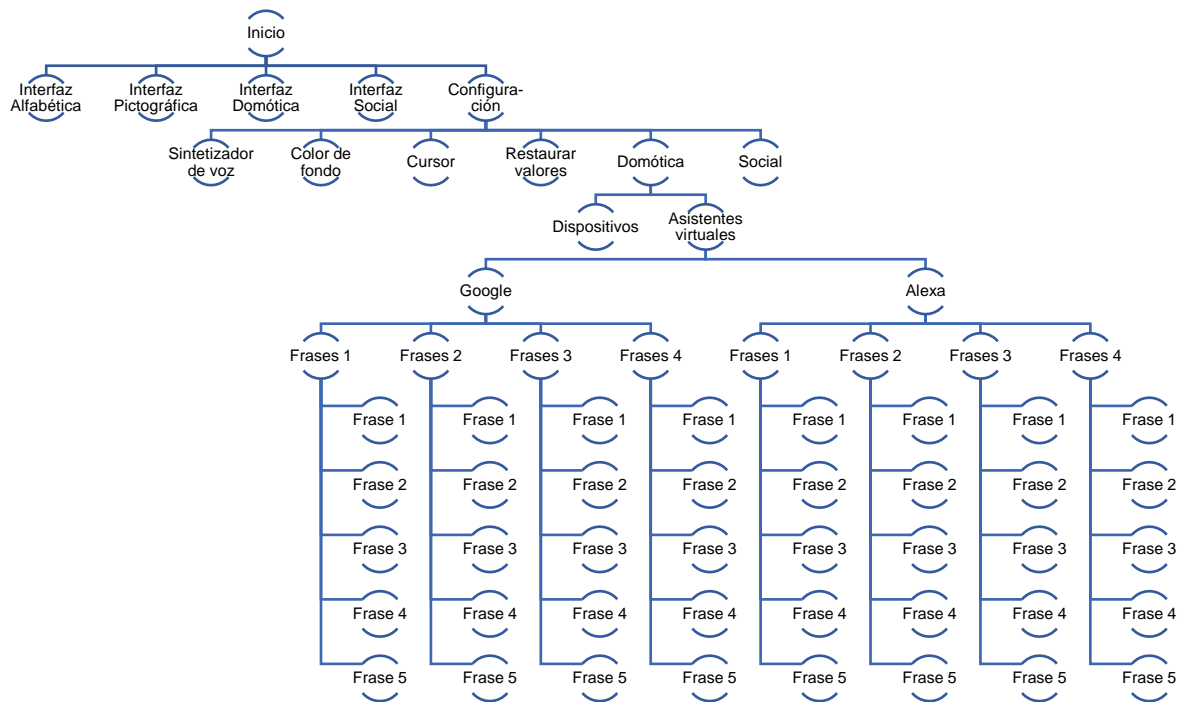


Figura 3.16 Mapa de navegación interfaz Asistentes virtuales-configuración.

Como se muestra en el mapa de navegación, existen cuatro menús de frases para cada asistente, los cuales están conformados por cinco frases, cabe mencionarse que estos menús no existen visualmente, sino que cada uno de ellos es una plantilla (véase figura 3.15) en el que se despliegan las cinco frases correspondientes a ese menú.

En el siguiente fragmento de código se explica la manera de configurar las primeras cinco frases para el asistente de Google, teniendo en cuenta que se realizan funciones parecidas para el resto de las frases y para el asistente de Alexa.

Primero se llama a la función *frasesGI()*, para poder mostrar las frases y posteriormente una vez seleccionada la frase que deseamos configurar, manda a llamar a la función *cuadroInputFrases()*.

```

3607     def ConfigGoogle():
3608         while True:
3609             CargarParametros()
3610             seleccionAsis(Google1,CuaFrasesAsis,0,15)
3611             frasesG1()
3612             x=0
3613             while x <= VelocidadCursor :
3614                 if GPIO.input(21) == GPIO.HIGH:
3615                     sonidoSeleccion()
3616                     cuadroInputFrases(0,"G","1",300,280)
3617                     time.sleep(0.01)
3618                     x=x+10

```




La función *frasesGI()* sirve para mostrar visualmente las frases dentro de la plantilla, esto se hace mediante un ciclo donde se leen las líneas del archivo de texto que contiene guardadas las frases (línea 3433), las concatena con el nombre del asistente (línea 3435) y las muestra en la interfaz (línea 3436) posicionándolas en el lugar correspondiente (3429,3438).

```
3428     def frasesGI():
3429         posicion=90
3430         for i in range(5):
3431             fuente = pygame.font.Font(None, 40)
3432             abrirDoc = open(Ubicacion+"Google.txt", "r")
3433             lecturaDeLinea=abrirDoc.readlines()[i].strip()
3434             abrirDoc.close()
3435             frase= fuente.render("Okey Google, "+lecturaDeLinea,
1, (Negro))
3436             screen.blit(frase, (60,posicion))
3437             i+=1
3438             posicion+=50
3439             pygame.display.update()
```

La función *cuadroInputFrases()* se utiliza para poder ingresar la frase que deseamos tener para posteriormente reproducirla, esto se realiza de la misma manera en la que se configuran los controles y los nombres de los botones explicados en el punto 3.2.2.

En esta función contamos con más argumentos los cuales nos determinan el menú al que se regresa (argumento numInterface, líneas 3558-3565, 3573-3580), y los límites que se tienen por frase (argumento tam, línea 3592-3593), además de la posición en la que se muestra en la ventana flotante (argumento pos, línea 3535).

```
3532     def
cuadroInputFrases(numLinea,asistente,numInterface,tam,pos):
3533         font = pygame.font.Font(None, 28)
3534         font2 = pygame.font.Font(None, 32)
3535         input_box = pygame.Rect(pos,200, tam, 40)
3536         box = pygame.Rect(200,150, 450, 110)
3537         color_inactive = pygame.Color('red2')
3538         color_active = pygame.Color('dodgerblue2')
3539         color = color_active
3540         active = True
3541         textFrase = ''
3542         done = False
3543
3544         while not done:
3545             for event in pygame.event.get():
3546                 if event.type==pygame.QUIT:
3547                     done=True
3548                 if event.type == pygame.KEYDOWN:
3549                     if active:
3550                         if event.key == pygame.K_RETURN:
3551                             if asistente == "G":
3552                                 output =
open(Ubicacion+"Google.txt", 'r+')
3553                                 frases=output.readlines()
3554                                 frases[numLinea]=textFrase+"\n"
```



```
3555         output.seek(0)
3556         output.writelines(frases)
3557         output.close()
3558         if numInterface == "1":
3559             ConfGoogle()
3560         elif numInterface=="2":
3561             ConfGoogle2()
3562         elif numInterface=="3":
3563             ConfGoogle3()
3564         elif numInterface=="4":
3565             ConfGoogle4()
3566         elif asistente == "A":
3567             output =
open(Ubicacion+"Alexa.txt", 'r+')
3568         frases=output.readlines()
3569         frases[numLinea]=textFrase+"\n"
3570         output.seek(0)
3571         output.writelines(frases)
3572         output.close()
3573         if numInterface == "1":
3574             ConfAlexa()
3575         elif numInterface == "2":
3576             ConfAlexa2()
3577         elif numInterface == "3":
3578             ConfAlexa3()
3579         elif numInterface == "4":
3580             ConfAlexa4()
3581         elif event.key == pygame.K_BACKSPACE:
3582             textFrase = textFrase[:-1]
3583         else:
3584             textFrase += event.unicode
3585
3586         titulo = font2.render("Configuración de Frases",
True, (0,0,0))
3587         pygame.draw.rect(screen, (234,230,202),box)
3588         pygame.draw.rect(screen, (204,202,202), (200,150,
450, 40))
3589         screen.blit(titulo, (box.x+100,box.y+10))
3590
3591         txt_surface = font.render(textFrase, True, (0,0,0))
3592         width=max(tam,txt_surface.get_width()+15)
3593         if width==tam:
3594             active=True
3595         else:
3596             active=False
3597         color=color_active if active else color_inactive
3598         if event.type==pygame.KEYDOWN:
3599             if event.key==pygame.K_BACKSPACE:
3600                 textFrase = textFrase[:-1]
3601                 active=True
3602                 color=color_active
3603         screen.blit(txt_surface, (input_box.x+5,
input_box.y+10))
3604         pygame.draw.rect(screen, color, input_box, 2)
```



3605

pygame.display.flip ()

Estas son las funciones utilizadas para la configuración de todas las frases de los asistentes.

3.3.2 Mapa de navegación para operación

En este apartado se describe el modo operación, en la que se reproduce la frase por medio del sintetizador para que el asistente pueda escuchar la petición.

En la figura 3.17 se muestra el mapa de navegación para este modo.

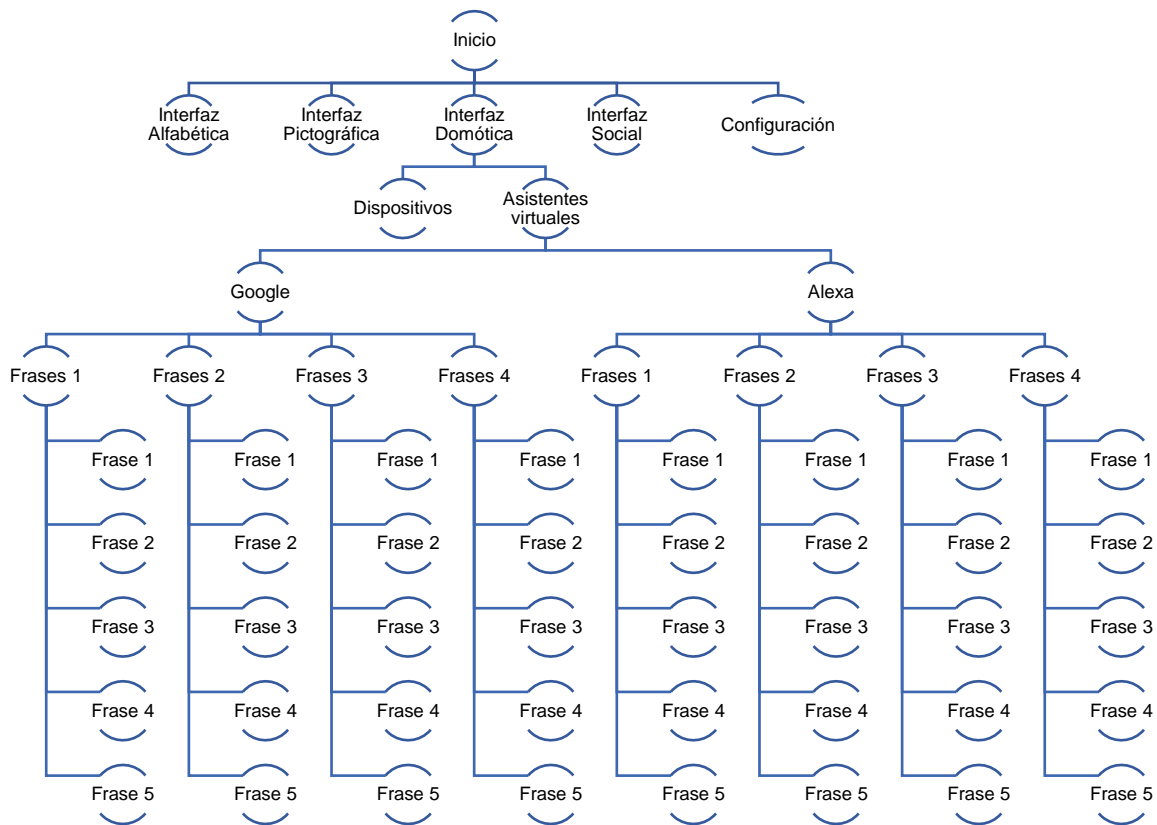


Figura 3.17 Mapa de navegación interfaz Asistentes virtuales-operación.

Al ingresar al menú de un asistente y seleccionar la frase que se desea reproducir se manda a llamar a la función `reproduccionFrasasG()`, la cual usará el sintetizador de voz para reproducirla.

```

1529     def interfazGoogle():
1530         while True:
1531             CargarParametros()
1532             seleccionAsis(Google1,CuaFrasasAsis,0,15)
1533             frasesG1()
1534             x=0
1535             while x <= VelocidadCursor :
1536                 if GPIO.input(21) == GPIO.HIGH:
1537                     sonidoSeleccion()
  
```



```
1538         reproduccionFraseG(0)
1539         time.sleep(0.01)
1540         x=x+10
```

La función *reproduccionFrasesG()* es la encargada de reproducir la frase, estas frases se encuentran juntas en un archivo de texto, por lo que se recibe como argumento el número de línea, donde se abre el archivo del asistente (línea 1516), se lee la línea deseada (línea 1517) y se reproduce esta, concatenada con el nombre del asistente (líneas 1519-1520).

```
1515     def reproduccionFraseG(linea):
1516         abrirDoc = open(Ubicacion+"Google.txt", "r")
1517         frase=abrirDoc.readlines()[linea].strip()
1518         abrirDoc.close()
1519         union = ("espeak "+ Idioma + Tono + " 'Okey Google, " +
frase + "' --stdout|aplay")
1520         os.system(union)
```

Estas son las funciones utilizadas para la operación de todas las frases de los asistentes.

CAPÍTULO 4. PRUEBAS Y RESULTADOS

En este apartado se describe el desarrollo de las pruebas realizadas para la creación del módulo IR y su incorporación al SAAC, la comunicación entre el SAAC y los asistentes virtuales, y la programación para distintas resoluciones del sistema.

4.1 Placa de prueba

Para realizar las pruebas correspondientes con las señales infrarrojas, se realizaron las conexiones del receptor y el circuito para controlar la transmisión con el LED IR (véase figura 4.1) en una placa fenólica perforada (véase figura 4.2), esto con la finalidad de poder realizar las pruebas de manera mucho más rápida, con la facilidad de contar con borneras de conexión para los jumpers de alimentación (voltaje 5 v y tierra) y los pines de datos para el receptor (S) y transmisor (L).

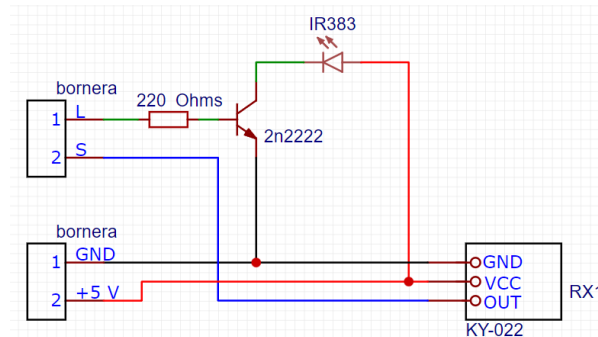


Figura 4.1 Esquemático de conexión de receptor transmisor IR

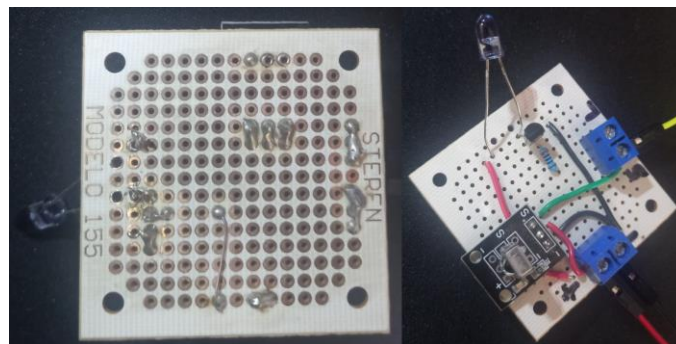


Figura 4.2 Placa perforada con receptor y transmisor IR.

4.2 Creación de interfaz gráfica para comunicación entre receptor IR y la Raspberry

La primera prueba que se realizó fue conectar directamente el receptor IR en la Raspberry (véase figura 4.3), utilizando un ejemplo de código encontrado en internet¹, donde se realizó una

¹ <https://github.com/Lime-Parallelogram/pyIR/blob/master/GUI.py>



interfaz gráfica de usuario (GUI) para poder obtener una señal infrarroja, y almacenar los datos obtenidos en un archivo de texto para su posterior uso.

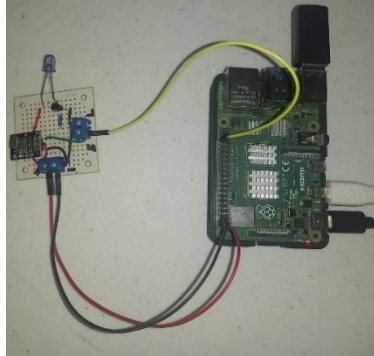


Figura 4.3 Conexión del receptor a la Raspberry.

La GUI realizada tenía el aspecto mostrado en la figura 4.4.



Figura 4.4 GUI de receptor IR en Raspberry.

Este sistema funcionaba con la ayuda de un mouse externo, con el que se seleccionaba el dispositivo a configurar, después se elegía el botón que se deseaba y se mostraba una leyenda “Esperando información...”, este es el momento en el que el sistema se ponía en espera de recibir una señal infrarroja por medio del sensor IR, donde el usuario debía apuntar el control hacia el receptor para guardar el código (véase figura 4.5).



Figura 4.5 Configuración de un botón de la Televisión.



Una vez presionado el botón del control hacia el receptor IR, se decodificaba la información como se muestra en el siguiente fragmento de código.

```
165 #Configuración del sensor
166 def getData():
167     numls = 0 #Número de unos consecutivos
168     command = [] #Pulsos y tiempos
169     binary = 1 #Decodificación de código binario
170     previousValue = 0 #Estado previo del PIN
171     value = GPIO.input(16) #Estado actual del PIN
172     while value: #Esperar que presionen el botón
173         value = GPIO.input(16)
174     startTime = datetime.now() #Hora de inicio
175     while True:
176         if value != previousValue: #Esperar un cambio de estado
177             now = datetime.now() #Registra la hora actual
178             pulseLength = now - startTime #Calcula el tiempo entre
179             #pulsos
180             startTime = now #Reinicia el tiempo de inicio
181             command.append((previousValue,
182                             pulseLength.microseconds)) #Agrega tiempo de pulso a la matriz (el valor
183             #anterior es 0/1 para saber si el tiempo esta ON o OFF)
184             #Interrumpe el código si se detecta un período alto
185             #prolongado (fin del código)
186             if value:
187                 numls += 1
188             else:
189                 numls = 0
190             if numls > 10000:
191                 break
192             #Vuelve a leer valores
193             previousValue = value
194             value = GPIO.input(16)
195             #Convierte la información a binario
196             for (typ, tme) in command:
197                 if typ == 1:
198                     if tme > 1000: #Código NEC 1687.5 microsegundos =1
199                     #lógico, por lo que 1000 es una diferencia grande
200                     binary = binary * 10 + 1
201                 else:
202                     binary *= 10
203             if len(str(binary)) > 34: #A veces el binario tiene dos
204             #caracteres al final
205             binary = int(str(binary)[:34])
206     return binary
```

Decodificada la señal, en la pantalla se mostraba el valor obtenido en forma hexadecimal (véase figura 4.6), luego era necesario presionar el botón de guardar para almacenar la información obtenida en un archivo de texto, también se contaba con un botón de reiniciar, el cual inicializaba todos los datos obtenidos para los botones de un control con el valor de 0.



Figura 4.6 Código obtenido.

Se realizaron varias lecturas de diferentes botones para el control de un televisor, y se obtuvieron los valores que se muestran en la figura 4.7.

```

Televisión.txt
Archivo Editar Búsqueda Ver Documento Ayuda
Encender/Apagar :
0x300f7d02f
Mute:
0x300f76897
Vol+:
0x300f7e01f
Vol- :
0x300f7708f
Ch+:
0
Ch- :
0

```

Figura 4.7 Archivo de texto con los códigos para el control de televisión.

Se realizaron pruebas con un control de Blu-ray, y se obtuvieron los valores que se muestran en la figura 4.8.

```

Dispositivo 1.txt
Archivo Editar Búsqueda Ver Documento Ayuda
Encender/Apagar :
0x340040d00
Mute:
0x340040d00
Vol+:
0x340040d00
Vol- :
0
Ch+:
0
Ch- :
0

```

Figura 4.8 Archivo de texto con los códigos para el control de Blu-ray.

Podemos darnos cuenta que todos los códigos guardados para el control Blu-ray fueron iguales, a pesar de tratarse de botones diferentes, así que no obtuvimos estas señales correctamente, por lo que se concluyó que este código no es funcional para la recepción de las señales, debido a que el objetivo de esta incorporación es crear un sistema capaz de aprender a la mayor cantidad de códigos para controles remotos del mercado, por lo que se procedió a buscar una alternativa.



4.3 Biblioteca LIRC en Raspberry PI4

Una opción para este problema fue utilizar la biblioteca llamada LIRC, la cual, en su documentación describe que fue elaborada para el control de señales infrarrojas.

Esta biblioteca se instala desde la terminal de la Raspberry, se realizó el procedimiento para la instalación y las configuraciones necesarias siguiendo las indicaciones que se encontraron como ejemplo en un sitio web¹, pero al realizar la configuración de la biblioteca se presentó el error “Permiso denegado”, el cual impedía el acceso a la carpeta para editarla (véase figura 4.9).

```
pi@raspberrypi:~ $ sudo apt-get install lirc
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
lirc ya está en su versión más reciente (0.10.1-6.3-deb10u1).
Los paquetes indicados a continuación se instalaron de forma automática y ya no son necesarios.
  gconf-service gconf2-common libexiv2-14 libgconf-2-4 libgfortran3
  libgmime-2.6-0 libmicrodns0 libncurses5 libssl1.0.2 rpi-eeeprom-images
  uuid-dev
Utilice «sudo apt autoremove» para eliminarlos.
0 actualizados, 0 nuevos se instalarán, 0 para eliminar y 4 no actualizados.
pi@raspberrypi:~ $ sudo cat>>/etc/modules <<EOF
> lirc_dev
> lirc_rpi gpio_in_pin=23 gpio_out_pin=22
> EOF
bash: /etc/modules: Permiso denegado
pi@raspberrypi:~ $
```

Figura 4.9 Terminal de Raspberry, instalación y configuración de la biblioteca LIRC.

Se intentó acceder a estas carpetas y editarlas de otras maneras, pero en ninguno de los intentos se logró configurar, así que se buscó otra opción de instalación.

Se realizó la desinstalación e instalación de la biblioteca nuevamente, pero con las indicaciones obtenidas en otro sitio², pero al realizar la configuración se obtuvo el error “No existe el fichero o directorio”, impidiéndonos la configuración de esta biblioteca (véase figura 4.10).

```
pi@raspberrypi:~ $ sudo apt-get install lirc
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
lirc ya está en su versión más reciente (0.10.1-6.3-deb10u1).
Los paquetes indicados a continuación se instalaron de forma automática y ya no son necesarios
.
  gconf-service gconf2-common libexiv2-14 libgconf-2-4 libgfortran3
  libgmime-2.6-0 libmicrodns0 libncurses5 libssl1.0.2 rpi-eeeprom-images
  uuid-dev
Utilice «sudo apt autoremove» para eliminarlos.
0 actualizados, 0 nuevos se instalarán, 0 para eliminar y 4 no actualizados.
pi@raspberrypi:~ $ sudo mv /etc/lirc/lirc_options.conf.dist /etc/lirc/lirc_options.conf
mv: no se puede efectuar `stat' sobre '/etc/lirc/lirc_options.conf.dist': No existe el fichero
o el directorio
pi@raspberrypi:~ $
```

Figura 4.10 Terminal de Raspberry, instalación y configuración de la biblioteca LIRC (2).

¹https://www.instructables.com/Creating-a-Raspberry-Pi-Universal-Remote-With-LIRC/?_ga=2.179489325.866734330.1614985589-1715663999.1614985588

²<https://www.instructables.com/Setup-IR-Remote-Control-Using-LIRC-for-the-Raspher/>



Se buscaron más opciones para la configuración, pero en un foro se encontró la información de que actualmente no hay soporte de esta biblioteca para la Raspberry Pi4, así que se concluyó que era necesario buscar una opción externa a la Raspberry, ya que no se contaba con los recursos necesarios para el control de señales infrarrojas dentro de esta.

4.4 Biblioteca IRremote en Arduino

Se analizaron las opciones en el mercado para buscar una alternativa externa, una opción económica y de gran documentación es Arduino, debido a su facilidad de programación y a la variedad de proyectos y bibliotecas existentes para estas placas.

Se decidió utilizar la placa Arduino UNO, ya que es muy comercial y económica. Se instaló la biblioteca IRremote para el control de señales infrarrojas, y se realizaron las conexiones necesarias para el receptor y transmisor IR (véase figura 4.11).

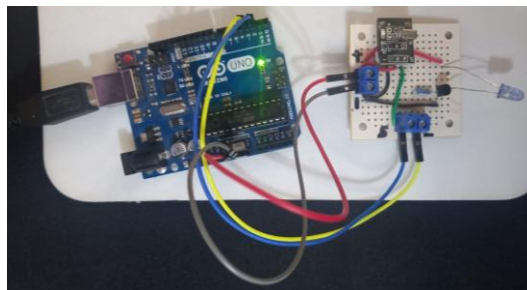


Figura 4.11 Conexión de receptor y transmisor al Arduino UNO.

Esta biblioteca cuenta con una carpeta de ejemplos, se utilizó el de recepción de señales (IRrecvDump.ino), y se realizó la lectura de dos señales, la información obtenida se observa en la figura 4.12 y 4.13.

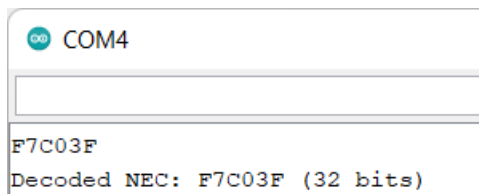


Figura 4.12 Señal de luces RGB.

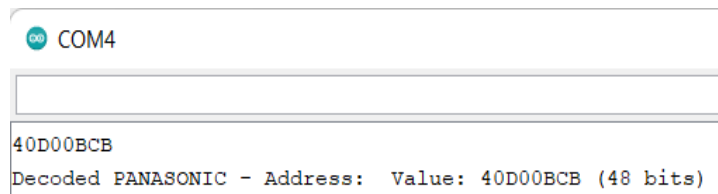


Figura 4.13 Señal de control de Blu-ray.

Una vez recopilada la información, se utilizó el ejemplo para la transmisión de las señales, al realizar el envío de la señal para unas luces RGB se envió correctamente, ya que se realizó la acción deseada; pero al enviar la señal del control del Blu-ray, se observó que no se realizó ninguna acción en el dispositivo. Se decidió realizar el proceso de obtener el código de diferentes botones de este control y realizar el envío, pero ninguna señal dio algún resultado al ser enviada.

Al presentarse este problema, se llegó a la conclusión que, por medio de este método, no se puede implementar la funcionalidad requerida para el módulo, así que se decidió buscar otra opción.



4.5 Biblioteca IRLib2 en Arduino

Se buscó una biblioteca que nos permitiera trabajar con una mayor cantidad de dispositivos, y se encontró la biblioteca IRLib2 que se descargó en línea¹.

Esta biblioteca tiene como códigos de ejemplo la recepción y el envío de señales infrarrojas tipo RAW, realizando una investigación se comprendió que este tipo, son señales no procesadas; es decir, no importa la marca o dispositivo del que provenga, esta es la señal completa sin decodificar.

Se utilizó el código para la recepción de señales RAW (rawRecv.ino), y se realizó la prueba para dos señales (control de luces RGB y control de Blu-ray), los resultados obtenidos se observan en la figura 4.14 y 4.15.

```
COM4
Ready to receive IR signals
Do a cut-and-paste of the following lines into the
designated location in rawSend.ino

#define RAW_DATA_LEN 68
uint16_t rawData[RAW_DATA_LEN]={
  9110, 4526, 562, 574, 566, 602, 534, 578,
  562, 574, 562, 574, 562, 578, 562, 574,
  562, 578, 558, 1682, 562, 1682, 562, 1678,
  566, 1678, 566, 574, 562, 1678, 566, 1678,
  562, 1682, 562, 1682, 562, 1682, 562, 578,
  558, 574, 566, 602, 534, 574, 562, 574,
  562, 574, 562, 578, 562, 574, 562, 1678,
  562, 1682, 562, 1678, 566, 1678, 562, 1682,
  562, 1682, 562, 1000};
```

Figura 4.14 Señal RAW de control RGB.

```
COM4
Ready to receive IR signals
Do a cut-and-paste of the following lines into the
designated location in rawSend.ino

#define RAW_DATA_LEN 100
uint16_t rawData[RAW_DATA_LEN]={
  3478, 1762, 414, 454, 406, 1334, 418, 454,
  414, 454, 414, 454, 418, 454, 414, 454,
  414, 454, 418, 454, 414, 454, 414, 454,
  418, 454, 418, 450, 414, 1326, 414, 454,
  418, 454, 414, 454, 414, 454, 418, 454,
  414, 454, 414, 1326, 418, 1322, 418, 454,
  414, 1326, 414, 454, 414, 454, 418, 454,
  414, 454, 414, 458, 414, 454, 414, 454,
  418, 450, 418, 1326, 414, 454, 414, 1326,
  418, 1322, 418, 1326, 414, 1322, 418, 454,
  414, 454, 414, 1326, 418, 450, 418, 1326,
  414, 1326, 414, 454, 418, 450, 418, 454,
  414, 1326, 414, 1000};
```

Figura 4.15 Señal RAW de control de Blu-ray.

Obtenidas las señales requeridas, se utilizó el código para el envío de señales RAW (rawSend.ino) con los códigos obtenidos, dando como resultado el funcionamiento en ambas señales, este proceso se repitió con diferentes botones de ambos controles, teniendo resultados exitosos en todos los casos, así que una vez logrado este paso, se buscó la manera de almacenar las señales obtenidas para su posterior lectura y envío.

4.6 Guardar códigos en la memoria EEPROM en Arduino

Para el almacenamiento de los códigos, la propuesta inicial fue hacerlo en la memoria EEPROM de Arduino para guardarla localmente. Se realizó el siguiente código.

```
1  #include <IRLibSendBase.h>
2  #include <IRLib_HashRaw.h>
3  #include <IRLibRecvPCI.h>
4  #include <EEPROM.h>
5
6  IRsendRaw mySender;
7  IRrecvPCI myReceiver(2);
8
9  uint16_t rawData;
10 uint16_t rawDataM;
```

¹ <https://github.com/cyborg5/IRLib2>



```
11  uint16_t rawDatam;
12  uint8_t tamData;
13
14  uint16_t rawDataS;
15  uint16_t rawDataMS;
16  uint16_t rawDatamS;
17  uint8_t tamDataS;
18
19  uint8_t freq=36;
20
21  void setup()
22  {
23    Serial.begin(9600);
24    Serial.println("CONECTADO");
25    pinMode(7,INPUT_PULLUP);
26    pinMode(8,INPUT_PULLUP);
27    pinMode(13,OUTPUT);
28  }
29
30  //Almacenamiento del código
31  void storeCode(void)
32  {
33    tamData=recvGlobal.recvLength;
34    EEPROM.write(0,tamData);
35
36    int otro=recvGlobal.decodeBuffer[1];
37    Serial.println(otro);
38    uint16_t value=(uint32_t)&(otro);
39    Serial.println(value);
40    rawData=(uint32_t)&(recvGlobal.decodeBuffer[1]);
41    rawDatam=rawData;
42    rawDataM=rawData>>8;
43    EEPROM.write(1,rawDataM);
44    EEPROM.write(2,rawDatam);
45
46    Serial.println("RECIBIDO ");
47    Serial.print("Tam:");
48    Serial.println(tamData,DEC);
49    Serial.print("Raw: ");
50    Serial.println(rawData,DEC);
51  }
52
53  void sendCode(void)
54  {
55    tamDataS=EEPROM.read(0);
56    rawDataMS=EEPROM.read(1);
57    rawDatamS=EEPROM.read(2);
58
59    rawDataMS=rawDataMS<<8;
60    rawDataS=rawDataMS+rawDatamS;
61
62    Serial.println("ENVIADO ");
63    Serial.print("Tam:");
64    Serial.println(tamDataS,DEC);
```



```
65     Serial.print("Raw: ");
66     Serial.println(rawDataS,DEC);
67     Serial.print("RawM:");
68     Serial.println(rawDataMS,DEC);
69     Serial.print("Rawm:");
70     Serial.println(rawDatamS,DEC);
71
72     mySender.send(rawDataS,tamDataS,freq); //Pass the buffer,length,
optionally frequency
73 }
74
75 void signalLed()
76 {
77     digitalWrite(13,HIGH);
78     delay(5000);
79     digitalWrite(13,LOW);
80 }
81 void loop()
82 {
83     int boton1=digitalRead(7);
84     int boton2=digitalRead(8);
85     if (boton1==LOW)
86     {
87         sendCode();
88     }
89     else if (boton2==LOW)
90     {
91         myReceiver.enableIRIn();
92         signalLed();
93         myReceiver.getResults();
94         storeCode();
95         myReceiver.disableIRIn();
96     }
97 }
```

La finalidad es guardar el valor en una localidad de la memoria EEPROM para posteriormente ser llamado, recordando que esta memoria mantiene los valores guardados sin importar si la alimentación de la placa está conectada o no.

Al realizar la prueba, el resultado fue exitoso ya que se logró guardar el valor de la señal y el tamaño de esta, esto se hizo en localidades distintas, y al realizar la prueba de desconectar la alimentación y volver a conectarla, la información permaneció guardada en las localidades.

El problema que se presentó fue que el valor guardado corresponde a el puntero donde empieza un buffer creado para la señal y no la señal en sí, por lo que al obtener un nuevo código o al desconectar la alimentación, el buffer se reinicia y se pierde la señal (esto es debido a la programación de la biblioteca utilizada).

No es posible guardar todo el buffer en la memoria EEPROM, ya que el Arduino UNO cuenta con una memoria de 1024 bytes, los cuales almacenan valores entre 0 a 255 por localidad, y las señales obtenidas están conformadas por cadenas de valores mayores a 255 en cada dato, por lo que se tendría que utilizar dos localidades o más por dato (en la figura 4.16 se puede observar



un ejemplo de esto donde se ve que la señal obtenida tiene un valor de 539, por lo que se guardó en dos localidades de memoria, una llamada Rawm y RawM, que al ser sumadas dan como resultado 539), y tomando en cuenta que la señal podría ser de hasta 100 valores, la memoria no es suficiente para guardar todas señales deseadas para el sistema, por ello se descartó esta opción y se buscó otra solución para almacenar la información.

```
COM4
CONECTADO
RECIBIDO
Tam: 68
Raw: 539
ENVIADO
Tam: 68
Raw: 539
RawM: 512
Rawm: 27
```

Figura 4.16 Valor guardo en memoria EEPROM.

4.7 Módulo microSD con Arduino

Una alternativa fue utilizar un módulo para una memoria microSD, para almacenar la información en archivos de texto, por lo que se procedió a llamar la biblioteca para el control del módulo microSD (SD.h), la cual está integrada dentro de IDE Arduino, y se realizaron las conexiones entre el módulo y la placa de Arduino UNO (véase figura 4.17).

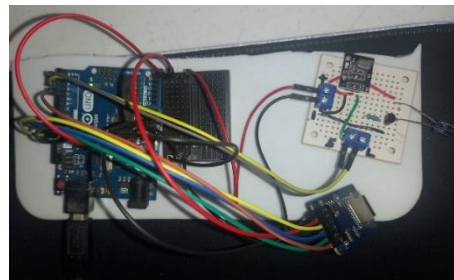


Figura 4.17 Conexión de módulo microSD.

Se realizó un código para obtener la señal (véase figura 4.18) y guardarla en un archivo de texto dentro de la memoria microSD (véase figura 4.19), y se obtuvo un resultado exitoso.

```
COM4
Iniciando tarjeta SD...exitosa.
Guardando valores de la señal...Acabamos.
```

Figura 4.18 Puerto COM4 con resultado de módulo microSD.

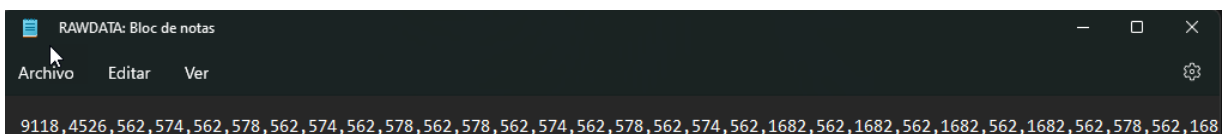


Figura 4.19 Archivo de texto generado con la señal obtenida.



Una vez con la señal almacenada, se desarrolló un código para el envío de la señal, donde se hizo la lectura de los datos guardados dentro del archivo de texto, y se envió la información hacia el LED IR para su transmisión, obteniendo resultados exitosos en las pruebas realizadas (véase figura 4.20).

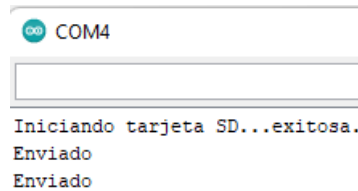


Figura 4.20 Puerto COM4 con resultado de enviado.

Con este paso logrado, se buscó la manera de comunicar el Arduino UNO con la Raspberry, esto con la finalidad de controlar desde la Raspberry el modo de funcionamiento de Arduino, donde se indica si debe estar en modo aprendizaje o en modo transmisión.

4.8 Conexión Serial entre Raspberry y Arduino

La comunicación serial es la más sencilla y común dentro de las placas de Arduino, el problema que se tiene, es que la Raspberry solamente cuenta con un puerto de comunicación serial, este puerto se encuentra ocupado por un módulo que solamente trabaja con este tipo de comunicación (módulo de transmisión de mensajes SMS) conectado a la Raspberry, por lo que se decidió buscar otro protocolo de comunicación.

4.9 Conexión I2C entre Raspberry y Arduino

Otro protocolo de comunicación es I2C, para realizar esta comunicación es necesario utilizar un convertidor de 5 V a 3.3 V, esto es debido a que Arduino trabaja con un 1 lógico de 5 V, pero Raspberry trabaja con un 1 lógico de 3.3 V, de modo que si se conecta de manera directa, al transmitir desde Arduino, es muy posible el quemar los pines de la Raspberry por la diferencia de voltajes, así que se realizaron las conexiones entre Arduino, Raspberry y el convertidor para sus pines de comunicación SDA y SCL (véase figura 4.21).

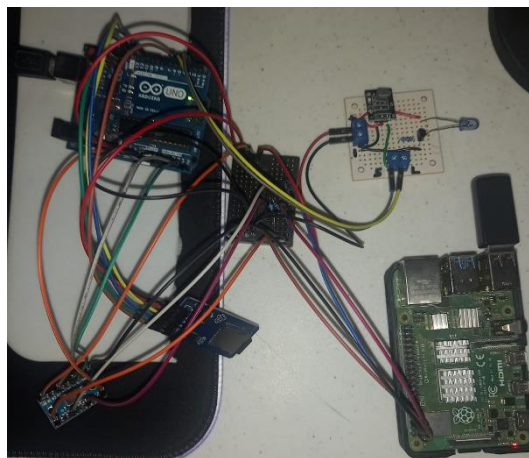


Figura 4.21 Conexión entre Raspberry y Arduino mediante I2C.



Se realizó el siguiente código en python.

```
1  from smbus2 import SMBus ##Paqueteria
2  import time
3
4  def r():
5      comando = "r\n"; ##rx
6      comando = comando.encode()
7      SMBus(1).write_i2c_block_data(0x04,0,comando)
##(addr[arduino],reg,data)
8      time.sleep(0.1)
9
10 def s():
11     comando = "s\n"; ##tx
12     comando = comando.encode()
13     SMBus(1).write_i2c_block_data(0x04,0,comando)
##(addr[arduino],reg,data)
14     time.sleep(0.1)
15
16 while True:
17     instruccion=input("Que hacer: ")
18     if instruccion=="r":
19         r()
20     elif instruccion=="s":
21         s()
```

Se envían las instrucciones desde la Raspberry hacia Arduino, para activar el modo de operación deseado dentro del Arduino (véase figura 4.22).

```
Shell x
Python 3.7.3 (/usr/bin/python3)
>>> %Run pruebaaaa.py
Que hacer: r
Que hacer: s
Que hacer:|
```

Figura 4.22 Terminal de Raspberry para el envío de instrucciones.

Se realizó un código para Arduino, este recibe las instrucciones desde la Raspberry, y dependiendo del valor obtenido, se determinan la función a realizar (véase figura 4.23), sea recibir una señal nueva y guardarla dentro de la memoria microSD o leer los datos de este archivo y transmitirlos por medio del LED IR.



```
COM4
Iniciando tarjeta SD...exitosa.
Iniciando tarjeta SD...exitosa.
Cadena recibida completa
r
BORRADO
Guardando valores de la señal...
Acabamos
Cadena recibida completa
s
Enviando IR
```

Figura 4.23 Puerto COM4 con los resultados de recepción y transmisión de señales.

Se realizó la prueba para diferentes botones y controles, utilizando el mismo archivo de texto, solamente actualizando la información, teniendo resultados exitosos, por lo que se procedió a realizar un código completo para todos los botones y dispositivos disponibles dentro del sistema (58 botones). Esto se realizó utilizando un switch case para la creación de los archivos de texto para cada botón, lectura del archivo y transmisión de la señal, utilizando el mayor número de caracteres para realizarlo.

Una vez terminado el código se procedió a compilarlo, pero como se observa en la figura 4.24, la memoria del Arduino UNO no es suficiente para cargar el código, ya que se excede su máximo en un 10%, por lo que se descartó este código y se procedió a buscar otra opción.

```
Error compilando para la tarjeta Arduino Uno.
case 'â': //aparato_b6
^~~~~
data section exceeds available space in board
El Sketch usa 22378 bytes (69%) del espacio de almacenamiento de programa. El máximo es 32256 bytes.
Las variables Globales usan 2267 bytes (110%) de la memoria dinámica, dejando -219 bytes para las variables locales. El máximo es 2048 bytes.
Not enough memory; see https://support.arduino.cc/hc/en-us/articles/360013825179 for tips on reducing your footprint.
Error compilando para la tarjeta Arduino Uno.
```

Figura 4.24 Barra de información de Arduino IDE.

4.10 Biblioteca IRLib2 en ESP32

Debido a las limitaciones de Arduino UNO, se consideró el cambio de placa a una más grade, el problema es que el costo subiría, así que se decidió utilizar la placa ESP32, esta puede programarse en el entorno de Arduino IDE, pero contiene mucha más memoria que la placa de Arduino UNO, con un costo similar a este.

Al realizar el cambio, se buscó instalar la biblioteca IRLib2, previamente probada para la placa de Arduino UNO, pero en la documentación se especifica que únicamente tiene soporte para placas Arduino AVR de 8 bits, y placas con procesador SAMD21 y SAMD51, debido a esto, se buscó otra opción de biblioteca para el control de señales infrarrojas con la placa ESP32.



4.11 Biblioteca IRremote en ESP32

Se encontró una biblioteca llamada IRremote (previamente utilizada en Arduino) como alternativa, pero esta, se encuentra modificada para poder utilizarla con la placa de ESP32, para el envío y recepción de señales infrarrojas, y se descarga en línea¹.

Se realizó la conexión del receptor y transmisor IR a la placa para las pruebas de los códigos (véase figura 4.25).

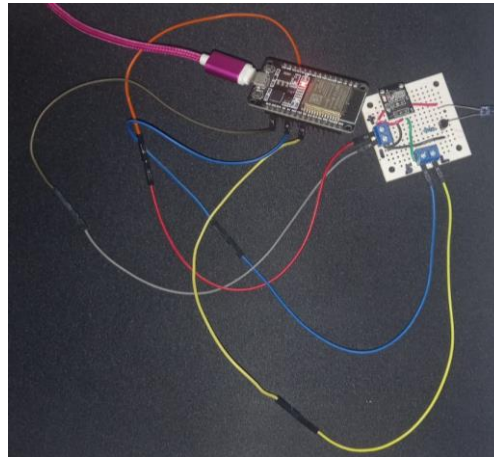


Figura 4.25 Conexión de receptor y transmisor a la ESP32.

Se utilizó el código de ejemplo para la recepción de señales RAW (IRrecvDumpV2.ino), y se obtuvieron señales como la que se muestra en la figura 4.26.

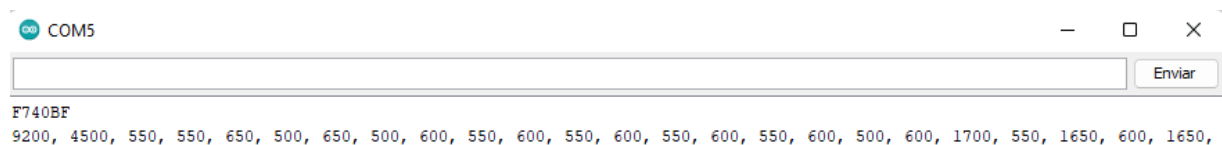


Figura 4.26 Puerto COM5 con los resultados de la señal obtenida.

Con esta señal obtenida, se utilizó el código de ejemplo para el envío de señales RAW (IRsendRawDemo.ino), este código se editó en cada prueba, ingresando la información obtenida de cada botón, al realizarlo, los dispositivos funcionaron adecuadamente según la acción correspondiente.

4.12 Módulo microSD en ESP32

Se procedió a guardar la información en un archivo de texto con la ayuda del módulo para memoria microSD, primero se realizó la prueba para poder tener conexión con el módulo microSD, y abrir un archivo de texto para guardar la información, haciendo las conexiones correspondientes, utilizando los códigos de ejemplos encontrado en la librería SD.h incluida en el IDE Arduino (véase figura 4.27).

¹ <https://github.com/SensorsIot/Definitive-Guide-to-IR/tree/master/ESP32-IRremote>

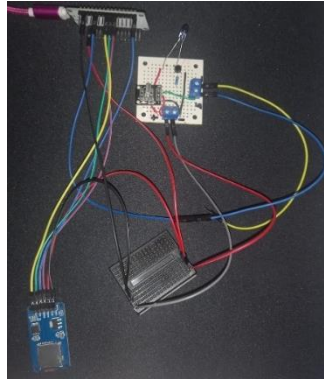


Figura 4.27 Conexión de módulo microSD y ESP32.

El resultado fue exitoso al hacer conexión con el módulo, pero al intentar abrir el archivo de texto no se logró (véase figura 4.28), se realizó esta prueba con distintos pines y memorias microSD, pero en ninguna se logró crear el archivo de texto, por lo que se descartó la idea de poder utilizar este módulo con la placa ESP32 y se buscó otra opción.

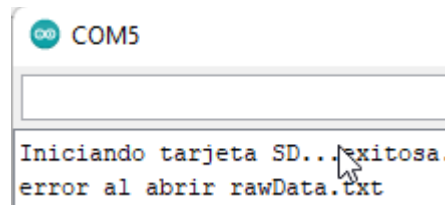


Figura 4.28 Puerto COM5 con resultados.

4.13 Conexión I2C entre Raspberry y ESP32

Como consecuencia del resultado negativo para guardar la información en la ESP32, ni en módulos externos, se buscó la forma de transmitir la señal obtenida, por medio del protocolo de comunicación I2C hacia la Raspberry, y poder guardar la información dentro de ella.

Se conectaron ambas placas con los pines correspondientes a la comunicación I2C, como se muestra en la figura 4.29, en esta conexión no es necesario el convertidor como se menciona en el punto 4.9 para la placa de Arduino UNO, debido a que la placa ESP32 tiene un 1 lógico de 3.3 V, al igual que la Raspberry.

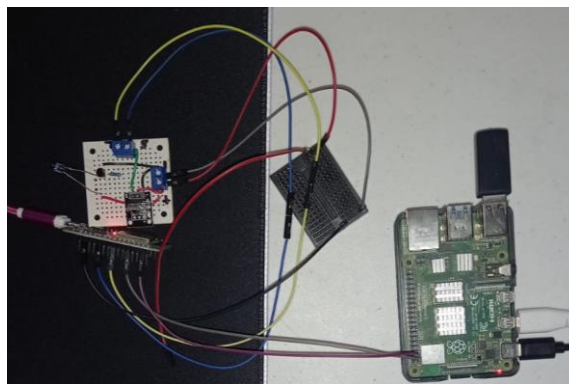


Figura 4.29 Conexión de comunicación I2C entre ESP32 y Raspberry.



Para la comunicación en la Raspberry, se utilizaron las librerías e Adafruit_PureIO.smbus y Raspberry_Pi_Master_for_ESP32_I2C_SLAVE.packer y para la ESP32 la librería WireSlave.h, dando como resultado un error al realizar la conexión entre ambas placas, ya que la terminal nos indica que no existe soporte para este tipo de comunicación, aunque los ejemplos encontrados en línea nos indican que es posible, en la práctica no funcionó (véase figura 4.30).

```
Shell x
ERROR: unsupported operand type(s) for &: 'list' and 'int'
ERROR: unsupported operand type(s) for &: 'list' and 'int'
ERROR: unsupported operand type(s) for &: 'list' and 'int'
ERROR: unsupported operand type(s) for &: 'list' and 'int'
ERROR: unsupported operand type(s) for &: 'list' and 'int'
ERROR: unsupported operand type(s) for &: 'list' and 'int'
ERROR: unsupported operand type(s) for &: 'list' and 'int'
ERROR: unsupported operand type(s) for &: 'list' and 'int'
ERROR: unsupported operand type(s) for &: 'list' and 'int'
ERROR: unsupported operand type(s) for &: 'list' and 'int'
```

Figura 4.30 Terminal de Raspberry con error de comunicación I2C.

4.14 Conexión Bluetooth entre Raspberry y ESP32

Se encontró la opción de conectar ambas placas mediante Bluetooth, el código realizado para la ESP32 es el siguiente.

```
1  #include "BluetoothSerial.h"
2  #include "esp_bt_device.h"
3
4  #if !defined(CONFIG_BT_ENABLED) ||
!defined(CONFIG_BLUEDROID_ENABLED)
5  #error Bluetooth is not enabled! Please run `make menuconfig` to and
enable it
6  #endif
7
8  BluetoothSerial SerialBT;
9
10 void setup() {
11     Serial.begin(115200);
12     SerialBT.begin("ESP32test"); //Bluetooth device name
13     Serial.println("The device started, now you can pair it with
bluetooth!");
14 }
15
16 void loop() {
17     if (Serial.available())
18     {
19         SerialBT.write(Serial.read());
20     }
21     if (SerialBT.available())
22     {
23         Serial.write(SerialBT.read());
24     }
25     delay(20);
26 }
```



Y para la Raspberry se utilizó el código que se muestra a continuación.

```
1  from bluetooth import *
2
3  def input_and_send():
4      print("\nType something\n")
5      while True:
6          data = input()
7          if len(data) == 0: break
8          sock.send(data)
9          sock.send("\n")
10
11 def rx_and_echo():
12     sock.send("\nsend anything\n")
13     while True:
14         data = sock.recv(buf_size)
15         if data:
16             print(data)
17             sock.send(data)
18
19 #MAC address of ESP32
20 addr = "24:62:AB:E3:E5:66"
21 #uuid = "94f39d29-7d6d-437d-973b-fba39e49d4ee"
22 #service_matches = find_service( uuid = uuid, address = addr )
23 service_matches = find_service( address = addr )
24
25 buf_size = 1024;
26
27 if len(service_matches) == 0:
28     print("couldn't find the SampleServer service =(")
29     sys.exit(0)
30
31 for s in range(len(service_matches)):
32     print("\nservice_matches: [" + str(s) + "]:")
33     print(service_matches[s])
34
35 first_match = service_matches[0]
36 port = first_match["port"]
37 name = first_match["name"]
38 host = first_match["host"]
39
40 port=1
41 print("connecting to \"%s\" on %s, port %s" % (name, host, port))
42
43 # Create the client socket
44 sock=BluetoothSocket(RFCOMM)
45 sock.connect((host, port))
46
47 print("connected")
48
49 #input_and_send()
50 rx_and_echo()
51
52 sock.close()
```



```
53 print("\n--- bye ---\n")
```

Al realizar esta conexión fue posible transmitir y enviar datos entre ambas placas bidireccionalmente. En la figura 4.31 se muestra el valor que se recibe desde la Raspberry (línea 1), y el valor que enviamos (línea 2).

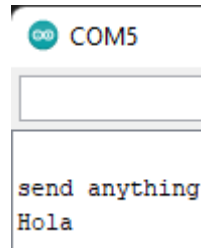


Figura 4.31 Puerto COM5 con envío y recepción de datos.

En la figura 4.32 obtenemos los resultados que obtuvimos dentro de la Raspberry, logrando una comunicación bidireccional exitosa, realizando la conexión entre ambas placas y recibiendo los datos provenientes de la ESP32.

```
Shell x
Connected

Python 3.7.3 (/usr/bin/python3)
>>> %Run enviarDatoBlue.py

service_matches: [0]:
{'service-classes': ['1101'], 'profiles': [('1101', 258)], 'name': 'ESP32SPP', 'description': None, 'provider': None, 'service-id': None, 'protocol': 'RFCOMM', 'port': 1, 'host': '24:62:AB:E3:E5:66'}
connecting to "ESP32SPP" on 24:62:AB:E3:E5:66, port 1
connected
b'H'
b'o'
b'l'
b'a'
b'\n'
```

Figura 4.32 Terminal de Raspberry con los resultados de la comunicación Bluetooth.

A pesar de que los resultados obtenidos fueron exitosos, se buscó otra opción de comunicación alámbrica, ya que, al tratarse de una conexión inalámbrica, esta puede llegar a ser inestable en algún momento, debido a interferencias externas.

4.15 Conexión USB entre Raspberry y ESP32

Se encontró la manera de conectar ambas placas mediante la conexión serial USB, para esta conexión no es necesario conectar los pines serial, sino que la conexión se hace mediante el cable USB, la cual, también sirve para alimentar a la ESP32 (véase figura 4.33).

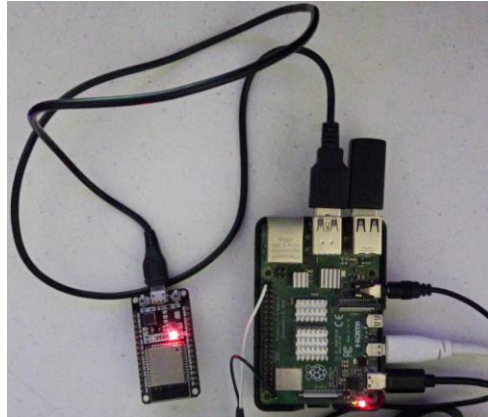


Figura 4.33 Conexión de comunicación serial USB entre ESP32 y Raspberry.

Se realizó un código para el envío de una instrucción desde la Raspberry que se muestra a continuación.

```
1  import serial
2  import time
3  from time import sleep
4
5  esp32=serial.Serial("/dev/ttyUSB0", baudrate=115200, timeout=1)
6  print("Empezamos")
7
8  while True:
9      comando=input("Mando: ")
10     esp32.write(comando.encode())
11
12     time.sleep(2)
13     print("Recibí: ")
14     if esp32.inWaiting()>0:
15         answer=esp32.readline()
16         value=(answer.decode("utf-8"))
17         print(value)
```

Y el código realizado para la ESP32 es el siguiente.

```
1  void setup()
2  {
3      Serial.begin(115200);
4      pinMode(2,OUTPUT);
5  }
6
7  void loop()
8  {
9      while (Serial.available()>0)
10     {
11         String valor=Serial.readStringUntil(';');
12         if (valor=="on")
13         {
14             digitalWrite(2,HIGH);
15             delay(600);
16             digitalWrite(2,LOW);
```



```

17         Serial.print("Prendí led de ESP32");
18     }
19 }
20 }

```

En donde la instrucción enviada desde la Raspberry prende el LED interno de la ESP32, y posteriormente recibe la cadena “Prendí led de ESP32” proveniente de la ESP32, completando así la comunicación bidireccional.

Los resultados de esta conexión se encuentran en la figura 4.34 (líneas de color blanco), además se observan unas líneas de error (líneas color rojo), estas se mostraron al interrumpir la conexión del cable entre ambas placas.

```

Shell x
Python 3.7.3 (/usr/bin/python3)
>>> %Run enviarDato.py
Empezamos
Mando: on;
Recibí:
Prendí led de ESP32
Mando: on;
Traceback (most recent call last):
  File "/usr/lib/python3/dist-packages/serial/serialposix.py", line 537, in write
    n = os.write(self.fd, d)
OSError: [Errno 5] Input/output error

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "/home/pi/Documents/Pruebas/ConexionSerial/enviarDato.py", line 10, in <module>
    esp32.write(comando.encode())
  File "/usr/lib/python3/dist-packages/serial/serialposix.py", line 571, in write
    raise SerialException('write failed: {}'.format(e))
serial.serialutil.SerialException: write failed: [Errno 5] Input/output error
>>> |

```

Figura 4.34 Terminal de Raspberry con los resultados de la comunicación serial USB.

Se desarrolló un código para la recepción de los códigos obtenidos en la ESP32, almacenamiento en archivos de texto dentro de la Raspberry, lectura de la información dentro del archivo y su envío hacia la ESP32 para la transmisión de la señal por medio del LED IR.

4.16 Conexión módulo IR

Una vez realizada la conexión se realizó la integración del módulo al SAAC, donde se respetó el diseño y la manera del funcionamiento del sistema, pero al operarlo, se obtuvo un error que al tener una interrupción en la conexión entre ambas placas como se muestra en la figura 4.34, el sistema se queda en un ciclo infinito, por lo que es necesario detener su ejecución y volverlo a empezar (véase figura 4.35).

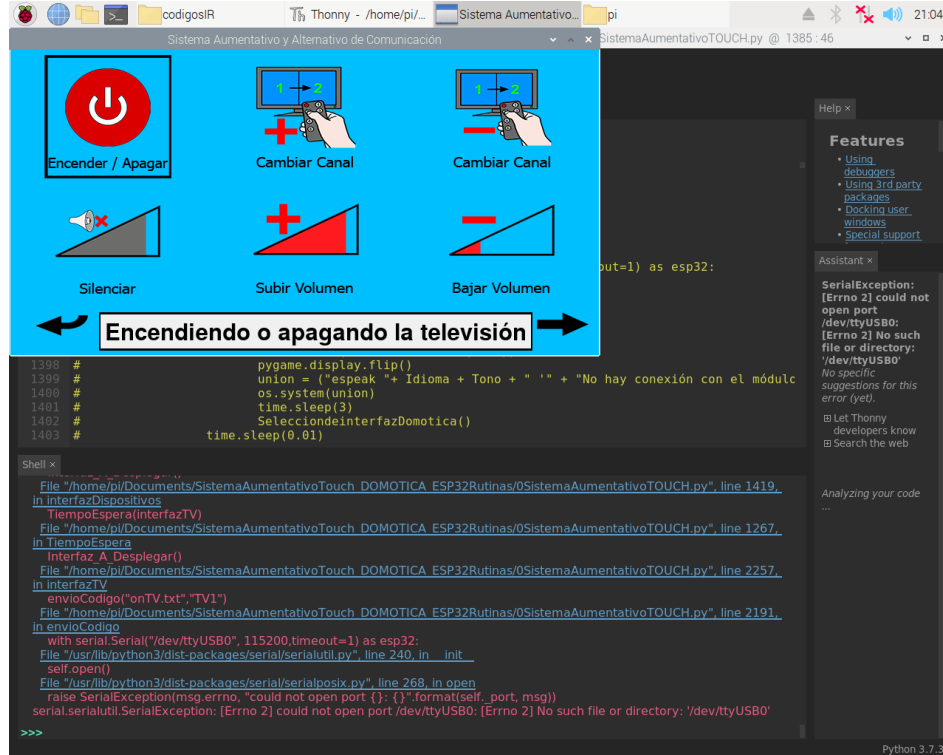


Figura 4.35 SAAC en ciclo infinito por error en la conexión /dev/ttyUSB0.

Para resolver este problema, se modificó el código integrando la función try y except como se muestra en el siguiente fragmento de código.

```

1389         try:
1390             with serial.Serial("/dev/ttyUSB0",
115200,timeout=1) as esp32:
1391                 time.sleep(0.1)
1392                 interfazDispositivos()
1393             except OSError:
1394                 print("OS")
1395                 screen.fill(ColorDeFondo)
1396                 screen.blit(ConexionModulo, (0, 0))
1397                 pygame.display.flip()
1398                 union = ("espeak "+ Idioma + Tono + " '" + "No
hay conexión con el módulo infrarrojo, revisa tu conexión" + "' --
stdout|aplay")
1399                 os.system(union)
1400                 time.sleep(3)
1401                 SeleccioneinterfazDomotica()

```

El sistema primeramente intentará hacer lo que se encuentra dentro del try, pero al encontrar un error de sistema (OSError), mostrará la plantilla que se observa en la figura 4.36 y regresará al menú anterior para que el usuario pueda seguir utilizando el sistema para sus otras funciones hasta ser corregido el error, de lo contrario el sistema trabajará con normalidad.



Figura 4.36 Plantilla de error de conexión.

4.17 Distancia y ángulo de recepción

Una vez realizada la conexión entre ambas placas se procedió a realizar las pruebas para la recepción de códigos para determinar la distancia y la posición correcta para la recepción de las señales infrarrojas, estas pruebas se hicieron a diferentes distancias y ángulos como se observa en la tabla 4.1 para la recepción del código F7C03F.

Tabla 4.1 Recepción de señales infrarrojas.

Señal obtenida	Distancia (cm)	Ángulo (°)	Resultado
F7C03F	<30	0	Exitoso
F7C03F	<30	$\pm 10 - 45$	Exitoso
F7C03F	<30	$\pm 45 - 90$	Exitoso
F7C03F	30-100	0	Exitoso
6BFE43AA	30-100	$\pm 10 - 45$	No exitoso
DC1E782A	30-100	$\pm 45 - 90$	No exitoso
F7C03F	>100	0	Exitoso
CE1972FD	>100	$\pm 10 - 45$	No exitoso
47A40EA1	>100	$\pm 45 - 90$	No exitoso

Con los resultados observados en la tabla se llegó a la conclusión que la mejor distancia para la recepción de una señal es menor a 30 cm del receptor IR con un ángulo de 0° a $\pm 90^\circ$, además de existir otras combinaciones válidas (véase figura 4.37).

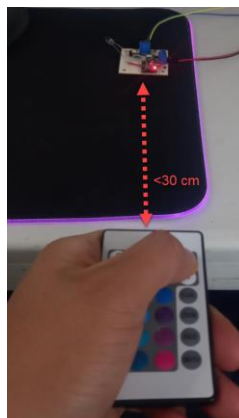


Figura 4.37 Posición para recepción de señales infrarrojas.



4.18 Guardado y actualización de un código

Para el guardado del código de una señal infrarroja se utilizó el segmento de código que se muestra a continuación.

```

4336         if esp32.inWaiting()>0:
4337             answer=esp32.readline()
4338             value=(answer.decode("utf-8"))
4339             txt = open(UbicacionDom+archivoTXT, 'w')
4340             txt.writelines(str(value))
4341             print(str(value))
4342             txt.close()

```

En este fragmento se lee la señal que procede de la ESP32 por medio del puerto serial USB, donde en la ESP32 recibe el código del receptor IR, decodifica la señal y la envía por medio del puerto Serial toda la información. Al leerla abre el archivo de texto y guarda la información leída en el puerto (véase figura 4.38), al hacer esto elimina la información dentro del archivo y guarda la nueva información por lo que al actualizar el código no es necesario realizar una tarea extra.

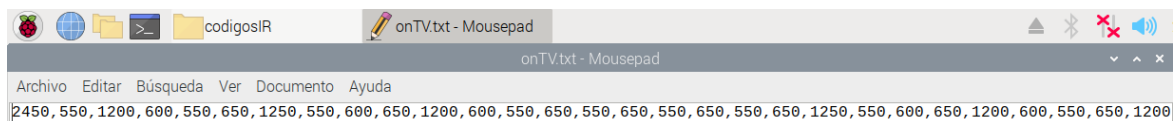


Figura 4.38 Archivo de texto con código de la señal infrarroja.

4.19 Transmisión de una señal

Al realizar el envío de la señal se realizó sin problema la transmisión siempre y cuando existiera la conexión entre ambas placas, pero un error que surgió es que al tener un archivo de texto completamente vacío (véase figura 4.39), si el usuario presionaba ese botón que aún no estaba configurado, el sistema mostraba un error ya que intentaba codificar algo que no existía y por lo cual se quedaba en un bucle infinito (véase figura 4.40), por lo que se optó crear el archivo de texto para cada botón con la cadena de "0000;", la cual no transmite nada pero hace que el sistema siga funcionando correctamente.

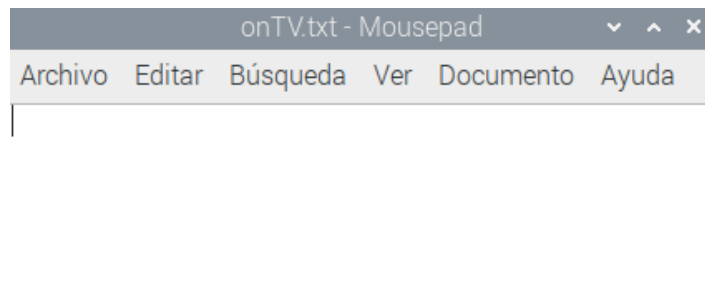


Figura 4.39 Archivo de texto vacío.

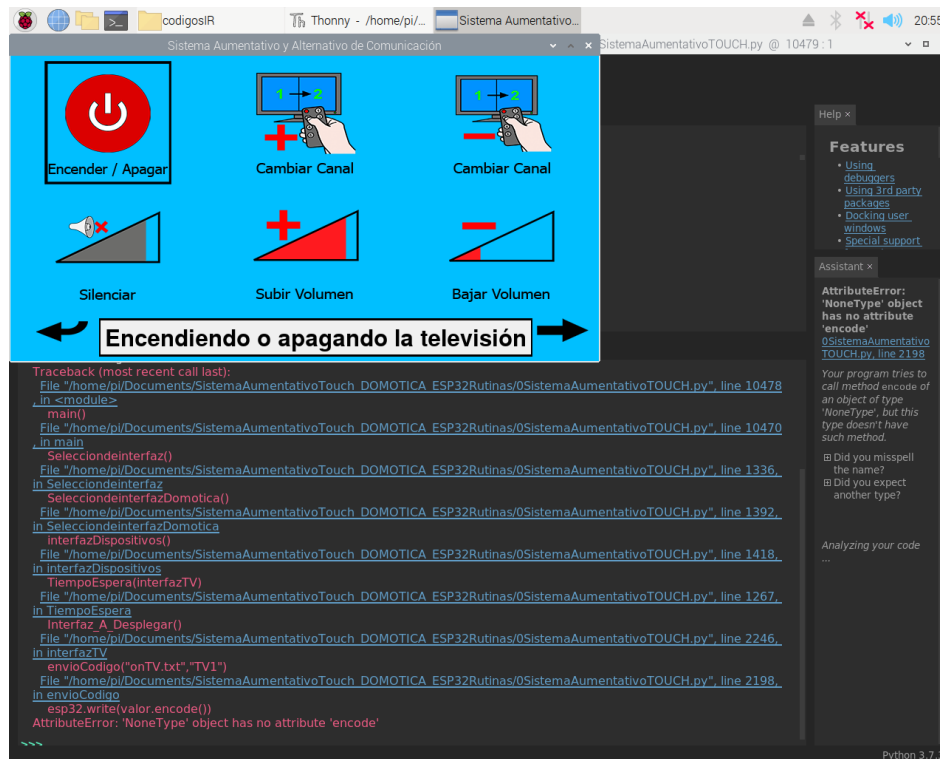


Figura 4.40 SAAC en bucle infinito con error al codificar los valores del archivo de texto vacío.

4.20 Distancia y ángulo de transmisión con LED IR

Se realizó la prueba para determinar la distancia y el ángulo de transmisión enviando una señal (botón de subir el volumen de una televisión) a diferentes distancias y ángulos.

La señal enviada se muestra a continuación.

9000, 4500, 550, 600, 550, 550, 550, 1700, 550, 550, 550, 600, 500, 600, 550, 600, 550, 550, 500, 1750, 500, 1750, 500, 600, 550, 1700, 500, 1750, 500, 1700, 550, 1700, 550, 600, 500, 1700, 600, 550, 500, 600, 550, 600, 500, 600, 550, 600, 550, 550, 500, 1750, 500, 600, 550, 1700, 550, 1700, 500, 1750, 550, 1700, 500, 1700, 550, 1700, 550, 1000;

La prueba se realizó utilizando un LED, y seis LED para obtener un rango de transmisión mayor (apuntando los seis LED en posiciones diferentes), los resultados obtenidos se muestran en la tabla 4.2.

Tabla 4.2 Envío de señales infrarrojas.

Distancia (m)	Ángulo (°)	Resultado con un LED	Resultado con seis LED
<1	0	Exitoso	Exitoso
<1	± 10 - 25	Exitoso	Exitoso
<1	± 25 - 45	No exitoso	Exitoso
<1	± 45 - 80	No exitoso	Exitoso
<1	± 90	No exitoso	No exitoso
1-5	0	Exitoso	Exitoso



1-5	$\pm 10 - 25$	Exitoso	Exitoso
1-5	$\pm 25 - 45$	No exitoso	Exitoso
1-5	$\pm 45 - 80$	No exitoso	Exitoso
1-5	± 90	No exitoso	No exitoso
5-10	0	Exitoso	Exitoso
5-10	$\pm 10 - 25$	No exitoso	Exitoso
5-10	$\pm 25 - 45$	No exitoso	Exitoso
5-10	$\pm 45 - 80$	No exitoso	Exitoso
5-10	± 90	No exitoso	No exitoso
>10	0	No exitoso	No exitoso
>10	$\pm 10 - 25$	No exitoso	No exitoso
>10	$\pm 25 - 45$	No exitoso	No exitoso
>10	$\pm 45 - 80$	No exitoso	No exitoso
>10	± 90	No exitoso	No exitoso

Como se observa en la tabla se obtiene un área de transmisión un poco reducida al utilizar un solo LED (véase figura 4.41).

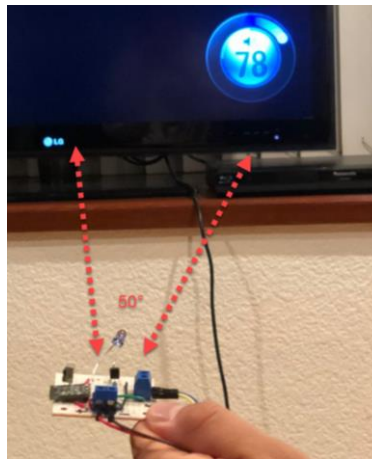


Figura 4.41 Área de transmisión con un LED IR.

Como se observa en la tabla se obtiene un área de transmisión mayor al utilizar seis LED (véase figura 4.42).

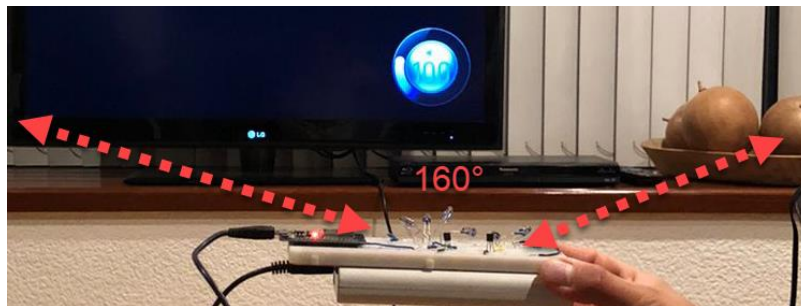





Figura 4.42 Área de transmisión con seis LED IR.



4.21 Funcionamiento dispositivos

Con la posición correcta obtenida con las pruebas para la recepción y transmisión de las señales, se procedió a realizar varias pruebas con diferentes dispositivos y sus funciones, estas se muestran en la tabla 4.3 con los resultados obtenidos en el funcionamiento de los dispositivos electrónicos, esto con la finalidad de probar si el sistema funciona correctamente al almacenar varios códigos de diferentes controles.

Tabla 4.3 Dispositivos y sus funciones.

Dispositivo	Botón	Código (señal RAW)	Resultado
Televisión	ON/OFF	{9000, 4500, 600, 550, 550, 550, 600, 1650, 550, 600, 550, 550, 550, 600, 1600, 600, 1650, 600, 550, 600, 1650, 550, 1700, 550, 1650, 600, 1700, 550, 1650, 600, 550, 550, 600, 550, 550, 1650, 600, 550, 550, 550, 600, 550, 550, 550, 550, 1650, 600, 1650, 600, 1700, 550, 550, 550, 1650, 600, 1700, 550, 1700, 550, 1650, 600, 1000}	
	Vol +	{9000, 4500, 550, 600, 550, 550, 550, 1700, 550, 550, 550, 600, 500, 600, 550, 600, 550, 550, 500, 1750, 500, 1750, 500, 600, 550, 1700, 500, 1750, 500, 1700, 550, 1700, 550, 1700, 550, 600, 500, 1700, 600, 550, 500, 600, 550, 600, 500, 550, 550, 550, 600, 550, 550, 500, 1750, 500, 1750, 500, 600, 550, 1700, 550, 1700, 500, 1750, 550, 1700, 500, 1700, 550, 1700, 550, 1000;}	
	Vol -	{8950, 4550, 600, 550, 600, 550, 550, 1700, 550, 550, 600, 500, 600, 550, 550, 550, 600, 550, 550, 1650, 600, 1650, 600, 550, 550, 1650, 600, 1650, 600, 1650, 600, 1650, 600, 1650, 600, 1650, 550, 1700, 550, 1700, 550, 550, 550, 550, 550, 600, 550, 550, 550, 600, 550, 550, 600, 550, 550, 550, 550, 550, 1700, 550, 1650, 600, 1650, 600, 1650, 600, 1650, 600, 1650, 550, 1000}	



ON/OFF

{3550, 1700, 450, 450, 400, 1350, 450, 400, 450, 400, 500, 400, 450, 400, 450, 450, 450, 400, 450, 400, 450, 450, 400, 450, 400, 500, 400, 450, 400, 450, 450, 450, 1300, 450, 1250, 500, 400, 450, 1300, 450, 400, 450, 450, 1250, 450, 1300, 450, 1300, 450, 1300, 450, 400, 450, 450, 1300, 400, 450, 450, 1300, 450, 400, 450, 450, 450, 400, 450, 1300, 450, 1000}



Blue-ray

Open

{3500, 1750, 450, 400, 450, 1300, 450, 450, 400, 450, 450, 450, 400, 450, 400, 450, 450, 400, 450, 450, 400, 450, 450, 400, 450, 450, 400, 1350, 400, 450, 450, 450, 400, 450, 450, 400, 450, 450, 450, 450, 1300, 400, 450, 450, 400, 450, 450, 400, 1350, 450, 1300, 450, 400, 450, 1300, 400, 1000}



Menú

{3500, 1750, 450, 400, 450, 1300, 450, 400, 450, 450, 400, 450, 450, 450, 450, 400, 450, 450, 400, 450, 400, 500, 1250, 450, 450, 450, 400, 450, 450, 450, 400, 450, 400, 450, 1300, 450, 400, 450, 450, 400, 450, 450, 450, 400, 450, 400, 450, 1300, 450, 1300, 450, 1300, 450, 400, 450, 1300, 450, 450, 450, 1000}





Red

{9200, 4500, 600, 550, 600, 550, 600, 550, 550, 600, 600, 500, 650, 500, 600, 550, 600, 550, 600, 1600, 650, 1650, 600, 1600, 650, 1600, 650, 550, 600, 1600, 650, 1600, 600, 1700, 550, 1700, 600, 1600, 650, 500, 600, 550, 600, 550, 600, 550, 600, 550, 600, 550, 600, 550, 600, 1650, 600, 1650, 600, 1650, 600, 1650, 600, 1650, 600, 1650, 600, 1650, 600, 1650, 600, 1000}



Luces RGB Green

{9200, 4500, 600, 550, 600, 550, 600, 550, 550, 650, 500, 600, 550, 600, 550, 600, 550, 550, 1700, 600, 1600, 650, 1650, 600, 1650, 600, 500, 650, 1600, 600, 1650, 600, 1650, 600, 550, 600, 550, 600, 500, 650, 500, 600, 550, 600, 550, 600, 550, 600, 1650, 550, 1650, 600, 1650, 600, 1650, 600, 1650, 600, 1650, 600, 1650, 600, 1000}



Blue

{9200, 4450, 650, 500, 650, 500, 650, 450, 650, 500, 650, 500, 650, 500, 650, 500, 650, 1600, 650, 1600, 650, 1600, 650, 500, 650, 1600, 650, 1600, 650, 1600, 650, 500, 650, 500, 650, 500, 650, 500, 650, 500, 650, 500, 650, 500, 650, 1600, 650, 1600, 650, 1600, 650, 1600, 650, 1600, 650, 1600, 650, 1000}





ON/OFF	{2450, 550, 1200, 600, 550, 650, 1250, 550, 600, 650, 1200, 550, 600, 650, 550, 650, 550, 650, 1250, 550, 600, 650, 1200, 550, 600, 650, 1200, 600, 550, 1000}	
Proyector	{2400, 650, 600, 600, 1150, 650, 600, 600, 1200, 600, 650, 550, 1200, 600, 1200, 600, 650, 600, 1200, 600, 600, 600, 1150, 650, 1200, 600, 650, 600, 1150, 650, 600, 600, 1150, 650, 600, 600, 650, 1000}	
Zoom +	{2400, 600, 1200, 600, 1200, 600, 650, 600, 1150, 650, 600, 600, 1200, 600, 1150, 650, 550, 1200, 600, 650, 550, 1200, 600, 1200, 650, 600, 600, 1150, 650, 600, 600, 1200, 600, 650, 550, 1200, 600, 650, 600, 650, 1000}	
Focus +		

Como se observa, el sistema funciona correctamente al controlar diferentes dispositivos y al guardar diferentes códigos de señales.

4.22 Configuración de nombres de controles y botones editables

Para la configuración de los controles y botones editables fue necesario delimitar el tamaño para cada uno de ellos, debido a que, sin esto, el usuario podría ingresar una cadena muy larga de caracteres, esto hará que visualmente no se pudieran diferenciar el termino y el inicio de cada una de las cadenas, por lo que se realizó un recuadro en el que este delimitaba el tamaño para cada uno, con un número aproximado de 9 caracteres para cada nombre (tomando en cuenta que las letras m y w toman más espacio, y las letras l e i toman menor espacio).

El recuadro creado (véase figura 4.43) se muestra de color azul cuando el usuario tiene espacio para agregar caracteres, pero al sobrepasar este límite se pondrá de color rojo (véase figura 4.44) donde ya no permitirá realizar ninguna acción más que borrar un carácter para proseguir.

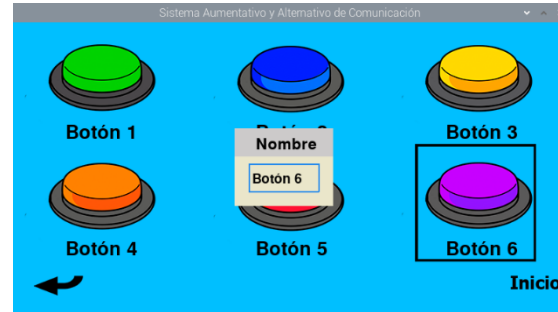
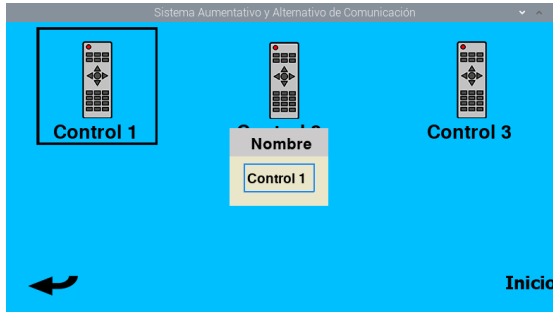


Figura 4.43 Configuración de nombres de controles y botones correcta.

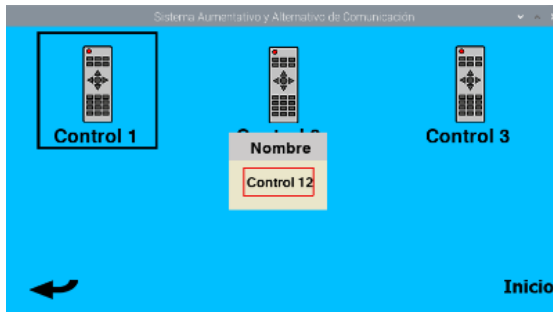


Figura 4.44 Configuración de nombres de controles y botones incorrecta.

4.23 Configuración del tamaño de las frases para la comunicación con asistentes virtuales

Para la configuración de las frases fue necesario delimitar el tamaño para cada una de ellas, ya que el usuario podría ingresar una cadena muy larga para las frases, que visualmente no se percibirían completas, por lo que se incorporó un recuadro en el que este delimitaba el tamaño para cada una de las frases.

Como se observan las plantillas para los asistentes en el Anexo C, estas contienen el logo del asistente a usar, por lo que las primeras dos frases deben ser más cortas que las otras, además debe tomarse en cuenta que la frase de activación es más larga en Google la cual es “Hey Google”, a diferencia con Alexa que solo se usa “Alexa”, por lo que los límites para cada una quedo de la siguiente manera:

- Google (primeras dos líneas): aproximadamente 30 caracteres.
- Google (siguientes líneas): aproximadamente 37 caracteres.
- Alexa (primeras dos líneas): aproximadamente 38 caracteres.
- Alexa (siguientes líneas): aproximadamente 43 caracteres.

Tomando en cuenta que las letras m y w toman más espacio, y las letras l e i toman menor espacio.

El recuadro creado (véase figura 4.45) se muestra de color azul cuando el usuario tiene espacio para agregar caracteres, pero al sobrepasar este límite se pondrá de color rojo (véase figura 4.46) donde ya no permitirá realizar ninguna acción más que borrar un carácter para proseguir.

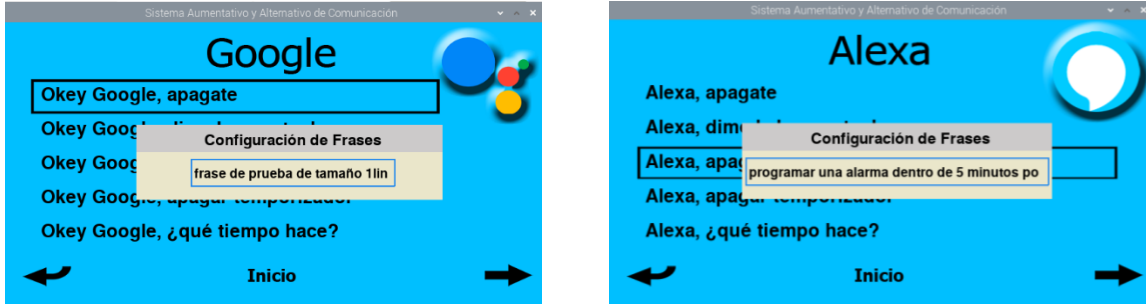


Figura 4.45 Configuración de frases correcta.

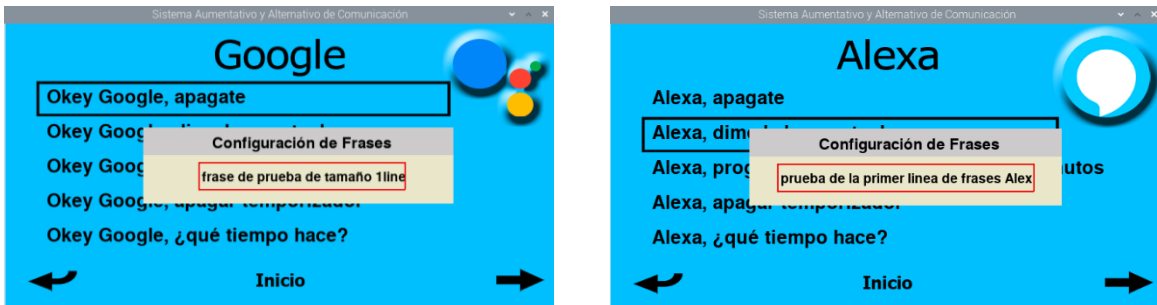


Figura 4.46 Configuración de frases incorrecta.

4.24 Prueba de frases para el asistente virtual

Se realizó la prueba con varias frases para revisar el funcionamiento de cada asistente como se muestra en la tabla 4.4.

Tabla 4.4 Funcionamiento de frases.

Asistente	Frase	Resultado
Google Assistant	Dime hora actual.	Son las 08:02 de la noche.
	¿Qué tiempo hace?	En este momento el termómetro marca 67°F con tormenta eléctrica en México.
	Dime las noticias de hoy	Aquí tienes las noticias más recientes.
	Dime un dato curioso	Aquí tienes algo increíble.
Alexa	programar alarma en 5 min	Listo, tu alarma está configurada para activarse 08:10 de la noche.
	Prende la luz	Se prendió la luz.
	Reproducir música	Se reprodujo música.
	Dime los recordatorios de hoy	No hay ningún recordatorio programado.
	Dime los resultados deportivos	Hola, me puedes pedir siguientes partidos y resultados de varios deportes y ligas.
	Buenas noches	Se ejecutó con éxito la rutina programada para detener música y apagar la luz.



Como se demuestra en la tabla, se realizaron las tareas solicitadas a los asistentes de manera exitosa, incorporando una rutina previamente programada dentro de Alexa, para poder realizar varias tareas en una sola frase.

4.25 Resolución del sistema

El sistema sobre el que se trabajó está realizado para pantallas HD (720p), pero uno de los problemas que se observaron, es que, al utilizar una pantalla de menor resolución, no era posible visualizar el sistema (véase figura 4.47), además de que al utilizar una de mayor resolución, el sistema no cubría todo el espacio, por lo cual, se perdía parte de la pantalla (véase figura 4.48).

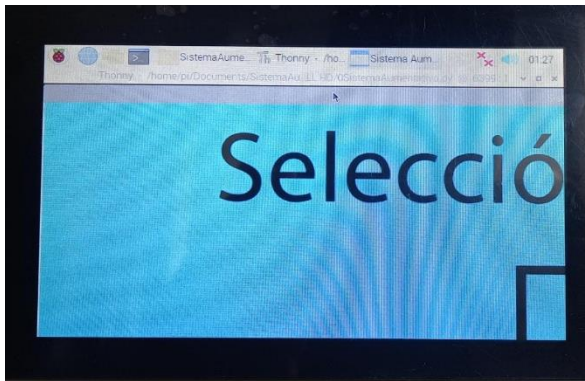


Figura 4.47 SAAC para pantalla HD en pantalla de 7"



Figura 4.48 SAAC para pantalla HD en pantalla FULL HD

Debido a esto, se redimensionó para dos tipos de pantallas diferentes:

1. Pantalla de 7", Touchscreen Display Raspberry Pi (480p).
2. Pantalla Full HD (1080p).

Para que la interfaz del SAAC ocupe el mayor espacio de las pantallas, se redimensionaron todas las plantillas, quedando con los siguientes tamaños:

- Pantalla de 7" a una resolución de 797 x 413 px.
- Pantalla Full HD a una resolución de 1915 x 1010 px.

Además de modificar las plantillas, fue necesario modificar la programación del sistema original, donde se editaron los tamaños de las ventanas de la interfaz, como se muestra en los siguientes fragmentos de códigos.

```
##### Pantalla 7" #####
15     SCREEN_WIDTH = 797     #CREAMOS UNA CONSTANTE QUE DARA EL ANCHO DE
NUESTRA VENTANA
16     SCREEN_HEIGHT = 413    #CREAMOS UNA CONSTANTE QUE DARA EL ALTO DE
NUESTRA VENTANA

##### Pantalla HD #####
16     SCREEN_WIDTH = 1280    #CREAMOS UNA CONSTANTE QUE DARA EL
ANCHO DE NUESTRA VENTANA
```



```

17      SCREEN_HEIGHT = 720      #CREAMOS UNA CONSTANTE QUE DARA EL
ALTO DE NUESTRA VENTANA

#####      Pantalla FULL HD      #####

14      SCREEN_WIDTH = 1915      #CREAMOS UNA CONSTANTE QUE DARA EL ANCHO DE
NUESTRA VENTANA
15      SCREEN_HEIGHT = 1010      #CREAMOS UNA CONSTANTE QUE DARA EL ALTO DE
NUESTRA VENTANA

```

Esto crea las ventanas, donde se despliegan las plantillas para cada una de las resoluciones, como se muestra en las figuras 4.49 y 4.50.



Figura 4.49 SAAC para pantalla 7"



Figura 4.50 SAAC para pantalla Full HD

Una vez realizada la ventana principal, se llaman las plantillas correspondientes para cada resolución. Al realizar esta acción, el sistema original cuenta con coordenadas exactas para los cursores, por lo que, al redimensionar las plantillas, estos cursores se desfasan para las nuevas resoluciones (véase figura 4.51, 4.52).



Figura 4.51 Cursor desfasado para pantalla 7"

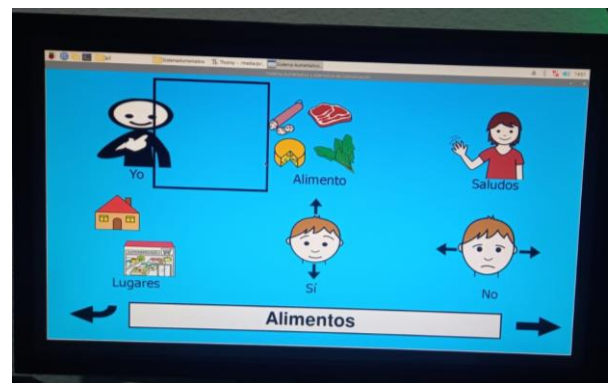


Figura 4.52 Cursor desfasado para pantalla Full HD

Así que se tuvo que modificar las coordenadas para cada una de las resoluciones, como se muestra en los siguientes fragmentos de códigos:

```

#####      Pantalla 7"      #####
2873      cambio(PrincipalCasa,0,0,"Yo")

```



2874	TiempoEspera (plantillaYo)
2875	
2876	
2877	cambio (PrincipalCasa,263,0,"Alimentos")
2878	TiempoEspera (plantillaAlimentos)
2879	
2880	
2881	
2882	cambio (PrincipalCasa,530,0,"Saludos")
2883	TiempoEspera (plantillaSaludos)
2884	
2885	
2886	
2887	cambio (PrincipalCasa,0,165,"Lugares")
2888	TiempoEspera (plantillaLugares)
2889	
2890	
2891	
2892	cambio (PrincipalCasa,263,165,"Sí")
#####	Pantalla HD #####
3412	cambio (PrincipalCasa,0,0,"Yo")
3413	TiempoEspera (plantillaYo)
3414	
3415	cambio (PrincipalCasa,428,0,"Alimentos")
3416	TiempoEspera (plantillaAlimentos)
3417	
3418	
3419	cambio (PrincipalCasa,853,0,"Saludos")
3420	TiempoEspera (plantillaSaludos)
3421	
3422	cambio (PrincipalCasa,0,290,"Lugares")
3423	TiempoEspera (plantillaLugares)
3424	
3425	cambio (PrincipalCasa,428,290,"Sí")
#####	Pantalla FULL HD #####
2783	cambio (PrincipalCasa,0,0,"Yo")
2784	TiempoEspera (plantillaYo)
2787	cambio (PrincipalCasa,650,0,"Alimentos")
2788	TiempoEspera (plantillaAlimentos)
2789	
2792	cambio (PrincipalCasa,1280,0,"Saludos")
2793	TiempoEspera (plantillaSaludos)
2794	
2797	cambio (PrincipalCasa,0,410,"Lugares")
2798	TiempoEspera (plantillaLugares)
2799	
2802	cambio (PrincipalCasa,650,410,"Sí")

Estas modificaciones se realizaron para todos los menús, lo que significó un número grande de cambios para cada uno de ellos, donde se realizaron una gran cantidad de pruebas para cuadrar los cursores con las plantillas en cada uno de los menús.



Además, se realizó el cambio de los tamaños y presentación de los cuadros donde se presentan las frases realizadas, y también los cursores de navegación, logrando así un recorrido correcto en las dos resoluciones nuevas (véase figura 4.53, 4.54).



Figura 4.53 Cursor para pantalla 7"



Figura 4.54 Cursor para pantalla Full HD

Con todos estos cambios realizados, se logró tener el SAAC para tres tipos diferentes de resoluciones, esto permite a los usuarios tener la posibilidad de utilizar una variedad de diferentes de tamaños de monitores, según la disponibilidad para cada usuario.



CONCLUSIONES

El módulo para recepción y transmisión de infrarrojos fue desarrollado utilizando un microcontrolador de bajo costo, y adicionalmente ocupa pocos componentes electrónicos, todos ellos de muy bajo costo; en las pruebas realizadas, se logró tener una funcionalidad adecuada respecto a lo esperado.

Se integró un nuevo menú llamado “Domótica”, el cual se divide en dos submenús, uno para el control de aparatos electrónicos y otro para interactuar con asistentes virtuales. Esto se realizó respetando el diseño y el método de navegación del SAAC original, para ello se crearon nuevos espacios en las plantillas existentes para el menú nuevo.

El módulo IR tiene la capacidad de recibir una instrucción desde la Raspberry para activar el modo de aprendizaje, lo cual activa el receptor IR para poder detectar una señal infrarroja, decodificarla y almacenar el código dentro de la Raspberry; en las pruebas realizadas se obtuvo con éxito la recepción de las señales infrarrojas provenientes de diferentes aparatos electrónicos de distintas marcas.

El módulo IR tiene la capacidad de recibir la información de una señal guardada dentro de la Raspberry, y transmitirla por medio del LED IR, para el control de aparatos electrónicos; en las pruebas realizadas se logró obtener un control correcto de las funciones de diferentes aparatos electrónicos configurados.

En el submenú llamado “Asistentes”, se incorporó la capacidad de interactuar con los asistentes virtuales mediante el uso de frases predeterminadas, las cuales pueden ser editadas a la preferencia del usuario; en las pruebas realizadas se logró con éxito la interacción del SAAC con los asistentes, donde estos respondieron de la manera esperada al reproducir las frases correspondientes a preguntas e invocación de rutinas.

Se crearon nuevas plantillas para el menú “Domótica”, las cuales se realizaron reutilizando pictogramas utilizados en el SAAC original e integrando nuevos pictogramas con el mismo diseño.

Por lo tanto, se concluye que el objetivo del proyecto fue cubierto de manera satisfactoria; sin embargo, aún hay trabajo que se puede realizar para mejorar las funcionalidades del SAAC, teniendo las siguientes propuestas:

- Implementar la incorporación del asistente virtual de manera directa, o, dicho de otro modo, que no sea que una reproducción de voz que deba ser escuchada por un asistente, si no por comandos dirigidos directamente a Google Assistant y Alexa, sin la necesidad de que el usuario cuente con uno de estos dispositivos de manera adicional.



- Idear un método aún más simple para que el usuario pueda cargar nuevos códigos IR al sistema.
- Adicionar la capacidad de automatizar tareas mediante los dispositivos que soportan IR, es decir poder crear algo semejante a las rutinas de los asistentes virtuales.
- Aumentar el alcance y rango de trabajo de las señales IR.



REFERENCIAS

- Alfonso Gago, J. F. (2012). Características básicas de los leds. En *Iluminación con tecnología LED* (págs. 2-20). España: Pananinfo.
- Al-Heeti, A. (16 de enero de 2021). *CNET*. Recuperado el 29 de junio de 2022, de <https://www.cnet.com/health/accessibility-devices-at-ces-2021-reflect-growing-focus-on-inclusive-tech/>
- Allan, A., Imeson, A., & Scheller, A. (s.f.). *Raspberry Pi*. Recuperado el 12 de junio de 2022, de <https://www.raspberrypi.com/documentation/computers/os.html>
- ARASAAC - Gobierno de Aragón. (s.f.). *ARASAAC*. Recuperado el 19 de enero de 2022, de <https://arasaac.org/aac/es>
- AREATECNOLOGIA. (s.f.). *AREATECNOLOGIA*. Recuperado el 12 de junio de 2022, de <https://www.areatecnologia.com/electricidad/pulsador.html#:~:text=Un%20pulsador%20el%C3%A9ctrico%20o%20bot%C3%B3n,vuelve%20a%20su%20posici%C3%B3n%20inicial.>
- Ayala, F. (11 de octubre de 2019). *UNIT ELECTRONICS*. Recuperado el 25 de mayo de 2022, de <https://uelectronics.com/transistores-bjt/#:~:text=Los%20transistores%20BJT%20pueden%20funcionar,como%20amplificador%20con%20ganancia%20variable.&text=Para%20que%20los%20transistores%20BJT,de%20corriente%20en%20un%20circuito.>
- Belli Gómez, G. (2018). Desarrollo de un sistema computacional para la comunicación verbal, que brinda apoyo a personas mudas con escasa movilidad mediante una interfaz visual y síntesis de voz. Ciudad de México: Tesis UNAM.
- C. Scorticati, M., & Micheli, F. (2006). Historia de la enfermedad de Parkinson. En *Enfermedad de Parkinson y trastornos relacionados* (págs. 1-9). Buenos Aires: Panamericana.
- C. Vanderheiden, G. (2002). A journey through early augmentative communication and computer access. *Journal of Rehabilitation Research and Development*, 39(6), 39-53.
- Charry, J. D., Cáceres, J. F., Salazar, A. C., López, L. P., & Solano, J. P. (2017). Trauma craneoencefálico. *Revista Chilena de Neurocirugía*, 43, 177-178.
- Chaskel, R., & Bonilla, M. F. (2017). Trastorno del espectro autista. *Curso continuo de actualización en pediatría CCAP*, 15(1), 19.
- Consejo Nacional para el Desarrollo y la Inclusión de las Personas con Discapacidad. (21 de junio de 2018). Recuperado el 15 de abril de 2021, de <https://www.gob.mx/conadis/articulos/la-esclerosis-lateral-amiotrofica-ela>
- Crespo, E. (2019 de junio de 15). *Aprendiendo Arduino*. Recuperado el 21 de 02 de 2022, de <https://aprendiendoarduino.wordpress.com/2019/06/15/ide-arduino-y-configuracion-3/>



- Delgadillo Martínez, O. (2020). Sistema computacional aumentativo y alternativo de comunicación como herramienta de apoyo a personas con discapacidad motora y del habla. Ecatepec de Morelos, Estado de México: Tesis UAEM.
- Electrónica J&M Computers & Technologies. (s.f.). *Electrónica J&M*. Recuperado el 13 de junio de 2022, de <https://electronica-jm.com/product/modulo-ky-022-sensor-receptor-infrarrojo-ir/>
- Electrónica Steren. (s.f.). *Steren*. Recuperado el 13 de junio de 2022, de <https://www.steren.com.mx/led-infrarrojo-de-5-mm-larga-distancia.html>
- Equipo BLOG Grupo Cajamar. (2021). *BLOG GRUPO COOPERATIVO CAJAMAR*. Recuperado el 22 de 03 de 2022, de <https://blog.grupocajamar.com/que-son-los-asistentes-virtuales-inteligentes/>
- Erazo Torricelli, R. (2004). Actualización en distrofias musculares. *Revista de Neurología*, 39(9), 860-861.
- ESPRESSIF SYSTEMS. (s.f.). *ESPRESSIF*. Recuperado el 10 de junio de 2022, de <https://www.espressif.com/en/products/modules/esp32>
- Fallas Sanabria, M. (2010). Esclerosis Lateral Amiotrófica. *Revista Medica de Costa Rica y Centroamerica*, LXVII(591), 89.
- Federación Autismo Madrid. (20 de abril de 2021). *Autismo Madrid*. Recuperado el 18 de enero de 2022, de <https://autismomadrid.es/articulo-especializado/que-son-los-sistemas-aumentativos-y-o-alternativos-de-comunicacion/>
- González Duque, R. (s.f.). ¿Qué es Python? En *Python para todos* (págs. 7-8).
- Gonzalez Rus, G. (s.f.). *Pulsadores, Conmutadores e interruptores: Sistemas de acceso para el discapacitado motórico*. Recuperado el 2022 de junio de 13, de chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://ioc.xtec.cat/materials/FP/Recursos/fp_edi_m04_/web/fp_edi_m04_htmlindex/WebContent/u5/media/pulsadors_commutadors_interruptors.pdf
- González, C. (s.f.). *Revista ElectroIndustria*. Recuperado el 20 de mayo de 2022, de <http://www.emb.cl/electroindustria/articulo.mvc?xid=416&ni=led-pequenos-gigantes-de-la-iluminacion#:~:text=Un%20diodo%20LED%2C%20acr%C3%B3nimo%20ingl%C3%A9s,atravesado%20por%20una%20corriente%20el%C3%A9ctrica.>
- Grupo Banco Mundial. (19 de marzo de 2021). *Banco Mundial BIRF-AIF*. Recuperado el 29 de junio de 2022, de <https://www.bancomundial.org/es/topic/disability#1>
- Guerra Carmenate, J. (s.f.). *Programarfacil.com*. Recuperado el 10 de junio de 2022, de <https://programarfacil.com/esp8266/esp32/>
- Inclusive Technology Ltd. (s.f.). *Inclusive Technology*. Recuperado el 29 de junio de 2022, de <http://www.inclusive.co.uk/integramouse-plus-p5725>



- Inclusive Technology Ltd. (s.f.). *Inclusive Technology*. Recuperado el 29 de junio de 2022, de <http://www.inclusive.co.uk/quha-zono-mouse#>
- Instituto Tecnológico de Aguascalientes. (s.f.). *Telecomunicaciones TICS*. Recuperado el 23 de mayo de 2022, de <https://telecomunicaciones2.webnode.mx/unidad-6/a6-6-comunicacion-por-infrarrojo-/>
- Isaac. (s.f.). *Hardwarelibre*. Recuperado el 25 de mayo de 2022, de <https://www.hwlibre.com/transistor-2n2222/>
- Ke, X., & Liu, J. (2017). Discapacidad intelectual. En *Manual de Salud Mental Infantil y Adolescentes de la IACAPAP* (pág. C.1 1). Ginebra: JM REY'S IACAPAP.
- Llamas, L. (s.f.). *LUIS LLAMAS Ingeniería, informática y diseño*. Recuperado el 25 de mayo de 2022, de <https://www.luisllamas.es/calculadora-de-transistor-bjt-como-interruptor/>
- Lozano, R. (13 de junio de 2021). *Talos Electronics*. Recuperado el 25 de mayo de 2022, de <https://www.taloselectronics.com/blogs/tutoriales/programar-esp32-con-ide-arduino>
- Malagón Valdez, J. (2007). Parálisis Cerebral. *MEDICINA*, 67(6/1), 586.
- Marín Prida, J. (2009). Esclerosis lateral amiotrófica: una actualización. *Revista Mexicana de Neurociencia*, 10(4), 281-282.
- MCI Electronics. (s.f.). *MCIElectronics*. Recuperado el 12 de junio de 2022, de <https://raspberrypi.cl/que-es-raspberry/>
- Mellado Villaseñor, G. (agosto de 2013). *Gobierno de México*. Recuperado el 21 de marzo de 2021, de <http://inaoe.repositorioinstitucional.mx/jspui/handle/1009/234>
- Mena Roa, M. (5 de octubre de 2020). *statista*. Recuperado el 22 de mayo de 2022, de <https://es.statista.com/grafico/22578/clasificacion-de-los-asistentes-de-voz/>
- mielectronicafacil. (2022). *mielectronicafacil*. Recuperado el 26 de 05 de 2022, de <https://mielectronicafacil.com/proyectos/hacer-circuito-impreso-pcb-facil/>
- National Center for Advancing Translational Sciences. (02 de enero de 2017). *Genetic and Rare Diseases Information Center*. Recuperado el 15 de abril de 2021, de <https://rarediseases.info.nih.gov/espanol/13212/sindrome-de-enclaustramiento>
- Pardo Madrid, T. (25 de abril de 2017). *Casaverde GRUPO*. Recuperado el 18 de enero de 2022, de <https://www.grupocasaverde.com/2017/04/25/que-son-los-saac/>
- Pérez Esteso, M. (11 de abril de 2016). *Geeky Theory*. Recuperado el 25 de mayo de 2022, de <https://geekytheory.com/easyeda-software-de-simulacion-de-circuitos-y-diseno-de-pcb/>
- Peterson, Z. (5 de octubre de 2020). *Altium*. Recuperado el 25 de mayo de 2022, de <https://resources.altium.com/es/p/what-is-a-pcb>
- Porras Betancourt, M., Núñez Orozco, L., Plascencia Álvarez, N., Quiñones Aguilar, S., & Sauri Suárez, S. (2007). Esclerosis múltiple. *Revista Mexicana de Neurociencia*, 8(1), 57.



- Premier Farnell Limite. (s.f.). *Newark México*. Recuperado el 20 de mayo de 2022, de <https://mexico.newark.com/display-seven-segment-display-technology>
- Raspberry Pi. (s.f.). *Raspberry Pi*. Recuperado el 12 de junio de 2022, de <https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/>
- REDACCIÓN. (s.f.). *DescubreArduino.com*. Recuperado el 10 de junio de 2022, de <https://descubrearduino.com/esp32-modulo-esp32-wroom-gpio-pinout/>
- Sensor IoT. (27 de noviembre de 2017). *GitHub*. Recuperado el 23 de mayo de 2022, de <https://github.com/SensorsIot/Definitive-Guide-to-IR/blob/master/ESP32-IRremote/examples/IRrecvDumpV2/IRrecvDumpV2.ino>
- Soro Camats, E., Basil, C., & Rosell, C. (2012). El alumnado con pluridiscapacidad: Características, evaluación y necesidades educativas. En *Pluridiscapacidad y contexto de intervención* (pág. 5). Barcelona: Institut de Ciències de l'Educació. Universitat de Barcelona.
- Statista. (febrero de 2019). *Statista*. Recuperado el 15 de abril de 2021, de <https://es.statista.com/estadisticas/972995/asistentes-virtuales-en-uso-en-el-mundo/>
- Valcarce, J. (s.f.). *Javier Valcarce*. Recuperado el 23 de mayo de 2022, de <http://javiervalcarce.eu/html/arduino-ir-remote-functioning-es.html#footnote2>
- Vendrell, J. (2001). Las afasias: semiología y tipos clínicos. *Revista de Neurología*, 32(10), 980.
- worldometer. (s.f.). *worldometer*. Recuperado el 29 de junio de 2022, de <https://www.worldometers.info/es/poblacion-mundial/>



FUENTES DE IMÁGENES

Figura 2.1. Picture Exchange Communication System (PECS) Recuperado de: <https://pecs-spain.com/el-sistema-de-comunicacion-por-el-intercambio-de-imagenes-pecs/>, el 21 de febrero de 2022.

Figura 2.2. Lenguaje bimodal Recuperado de: <https://www.pinterest.com.mx/pin/631066966514875326/>, el 21 de febrero de 2022.

Figura 2.3. Sistema pictográfico. Recuperado de: <https://zaguan.unizar.es/record/15100/files/TAZ-TFG-2014-928.pdf>, el 21 de febrero de 2022

Figura 2.4. Sistema gráfico Recuperado de: <https://aulaabierta.arasaac.org/arasaac-sistema-pictografico-de-referencia-en-la-cao>, el 21 de febrero de 2022.

Figura 2.5. SAAC electrónico Recuperado de: <https://www.fundacionconecta.org/2020/06/17/sistemas-alternativos-y-aumentativos-de-comunicacion-s-a-a-c/>, el 21 de febrero de 2022.

Figura 2.6 Diagrama de Raspberry Pi 4 Model B Recuperado de: <https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/>, el 12 de junio de 2022.

Figura 2.7 Pinout de Raspberry Pi4 Recuperado de: <https://www.infootec.net/activar-pins-en-raspberry-pi/>, el 29 de junio 2022.

Figura 2.8 Pinout de la placa ESP32 Recuperado de: <https://uelectronics.com/producto/esp32-38-pines-esp-wroom-32/>, el 29 de junio 2022.

Figura 2.9 Espectro de luz visible Recuperado de: <https://mconfort.com/que-es-radiacion-infrarroja/>, el 23 de mayo de 2022.

Figura 2.10 Longitud de onda de una señal de un control remoto Recuperado de: <http://javiervalcarce.eu/html/arduino-ir-remote-functioning-es.html#footnote2>, el 23 de mayo de 2022.

Figura 2.11 Ejemplo de una señal de un control remoto Recuperado de: <http://javiervalcarce.eu/html/arduino-ir-remote-functioning-es.html#footnote2>, el 23 de mayo de 2022.

Figura 2.12 Módulo KY-022 Recuperado de: https://dcelectronica.com.ar/index.php?route=product/product&path=128_133&product_id=28687, el 13 de junio de 2022.

Figura 2.13 Transmisor IR383 Recuperado de: <https://www.steren.com.mx/led-infrarrojo-de-5-mm-larga-distancia.html>, el 13 de junio de 2022.

Figura 2.14 Transistor TBJ 2n2222A TO-18 Recuperado de: <https://www.hwlibre.com/transistor-2n2222/>, el 13 de junio de 2022.



Figura 2.15 Circuito para transistor TBJ como interruptor Recuperado de:

<https://www.luisllamas.es/calculadora-de-transistor-bjt-como-interruptor/>, el 13 de junio de 2022.

Pulsador Recuperado de: <https://kinuma.com/es/atencion-observacion-memoria/7198-pulsador-de-respuesta-con-sonido-765023337778.html>, el 13 de junio de 2022.

Interruptor de mercurio Recuperado de: <https://aulasteam.com/sensores-de-inclinacion/>, el 13 de junio de 2022.

Sensibilidad Recuperado de: <https://uelectronics.com/producto/boton-tactil-capacitivo-tp223b/>, el 13 de junio de 2022.

Palanca Recuperado de: <https://www.directindustry.es/prod/idec/product-7180-609654.html>, el 13 de junio de 2022.

Pedal Recuperado de: <https://www.amazon.com.mx/Twidec-Interruptor-antideslizante-moment%C3%A1neo-Industrial/dp/B09S6P63KV>, el 13 de junio de 2022.

Soplo-succión Recuperado de: <http://educespecial-tecadaptada.blogspot.com/2016/03/conmutador-de-soplosuccion.html>, el 13 de junio de 2022.

Microfono Recuperado de: <https://grupocasmarmx.com/producto/microfono-dinamico-voz-musica-cuello-flexible-interruptor-pedestal-de-mesa-y-cable-mod-1622/>, el 13 de junio de 2022.

Luz Recuperado de: <https://es.aliexpress.com/i/1005002106595019.html>, el 13 de junio de 2022.

Joystick Recuperado de: <https://www.makeelectronico.com/producto/joystick-2-ejes-xy-con-pulsador/>, el 13 de junio de 2022.

Figura 2.16 Diodo LED Recuperado de: <https://www.simbologia-electronica.com/simbolos-electricos-electronicos/simbolos-iluminacion.htm>, el 21 de marzo de 2022.

Figura 2.17 Display de 7 segmentos y sus conexiones Recuperado de: <chrome-extension://efaidnbmninnibpcjpcglclefindmkaj/viewer.html?pdfurl=http%3A%2F%2Fprofesor.es.fi-b.unam.mx%2Fnormaelva%2Fpractica4.pdf&clen=2375466&chunk=true>, el 21 de marzo de 2022.

Figura 2.18 Pantalla LCD Recuperado de: https://es.123rf.com/photo_83544569_conjunto-de-dispositivos-electr%C3%B3nicos-planos-dispositivos-electr%C3%B3nicos-en-un-estilo-plano-los-dispositivos.html, el 21 de marzo de 2022.

Figura 2.19 Display LCD de 2x16 Recuperado de: <https://www.edutronika.com.mx/tienda/Pantalla-LCD-16x2-p405602351>, el 21 de marzo de 2022.

Figura 2.20 Diagrama esquemático de un píxel de una LCD Recuperado de: <chrome-extension://efaidnbmninnibpcjpcglclefindmkaj/viewer.html?pdfurl=https%3A%2F%2Finaoe.repositorioinstitucional.mx%2Fjspui%2Fbitstream%2F1009%2F234%2F1%2FMelladoVG.pdf&clen=3635355>, el 21 de marzo de 2022.



Figura 2.21 Arduino IDE Recuperado de: Autoria propia, el 14 de junio de 2022.

Figura 2.22 Menú Archivo Recuperado de: Autoria propia, el 14 de junio de 2022.

Figura 2.23 Botón de gestor de tarjetas adicionales Recuperado de: Autoria propia, el 14 de junio de 2022.

Figura 2.24 Gestor de URLs Adicionales de Tarjetas Recuperado de: Autoria propia, el 14 de junio de 2022.

Figura 2.25 Menú Programa> Incluir librería Recuperado de: Autoria propia, el 23 de junio de 2022.

Figura 2.26 Menú Herramientas Recuperado de: Autoria propia, el 14 de junio de 2022.

Figura 2.27 Gestor de tarjetas Recuperado de: Autoria propia, el 14 de junio de 2022.

Figura 2.28 Selección de placa Recuperado de: Autoria propia, el 14 de junio de 2022.

Figura 2.29 Monitor Serie Arduino Recuperado de: Autoria propia, el 14 de junio de 2022.

Figura 2.30 Interfaz principal de EasyEDA Recuperado de: Autoria propia, el 14 de junio de 2022.



ANEXO

Anexo A. Código de ESP32

```
1  #include <IRremote.h>
2
3  IRrecv irRecibo(4);
4  IRsend irEnvio(5);
5  decode_results resultados;
6
7  #define led 21
8  int tamCod=0;
9  String aux="";
10 unsigned int dato;
11 unsigned int codigo[100];
12 int freq=36;
13 String valor="";
14
15 void setup()
16 {
17   Serial.begin(115200);
18   Serial.setRxBufferSize(1024);
19   pinMode(led,OUTPUT);
20 }
21
22 void lectura(decode_results *resultados)
23 {
24   int cont = resultados->rawlen;
25   for (int i=1; i<cont; i++)
26   {
27     int valor=(resultados->rawbuf[i]*USECPERTICK);
28     Serial.print(valor);
29     Serial.print(",");
30   }
31   Serial.print("1000");
32   Serial.print(";");
33 }
34
35 void loop()
36 {
37   while (Serial.available(>0)
38   {
39     String valor=Serial.readStringUntil(';');
40     if (valor== "r")
41     {
42       irRecibo.enableIRIn();
43       digitalWrite(led,HIGH);
44       delay(14000);
45       digitalWrite(led,LOW);
46       if (irRecibo.decode(&resultados))
47       {
48         lectura(&resultados);
49         irRecibo.resume();
```



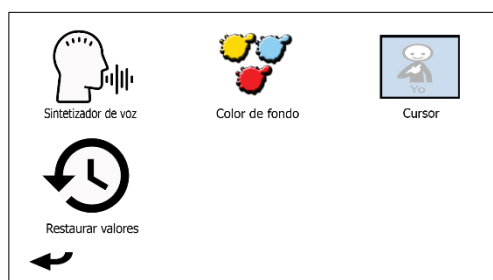
```

50     }
51     }
52     else
53     {
54         int tamStr=valor.length();
55         tamCod=0;
56         aux="";
57         for(int i=0;i<tamStr;i++)
58         {
59             char byt=valor[i];
60             if(byt==' ')
61             {
62                 dato=aux.toInt();
63                 codigo[tamCod]=dato;
64                 aux="";
65                 tamCod++;
66             }
67             else
68             {
69                 aux+=valor[i];
70             }
71         }
72         tamCod++;
73         irEnvio.sendRaw(codigo,tamCod,freq);
74     }
75 }
76 }

```

Anexo B. Plantillas para incorporación del módulo IR al SAAC

B.1 Plantillas ya existentes modificadas





B.2 Plantillas nuevas

Configuración de interfaz




Dispositivos Asistentes Virtuales


Inicio

Selección de interfaz




Dispositivos Asistentes Virtuales

Inicio





Televisión Estéreo Ventilador








Aire Acondicionado Persiana DVD

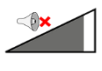


Inicio

Inicio








Encender / Apagar Cambiar Canal Cambiar Canal








Silenciar Subir Volumen Bajar Volumen

Inicio








Encender / Apagar Subir Volumen Bajar Volumen







Reproducir / Pausar Regresar Avanzar

Inicio








Encender / Apagar Subir Velocidad Bajar Velocidad





Giro de Ventilador Luz

Inicio








Encender / Apagar Subir Temperatura Bajar Temperatura




Modo


Inicio



Subir Persiana Detener Persiana Bajar Persiana

Inicio



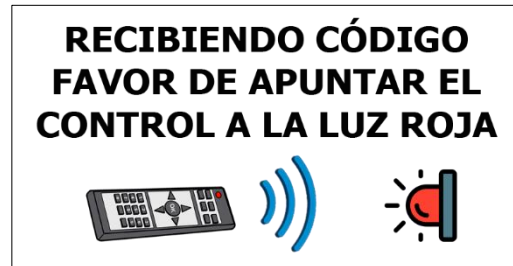
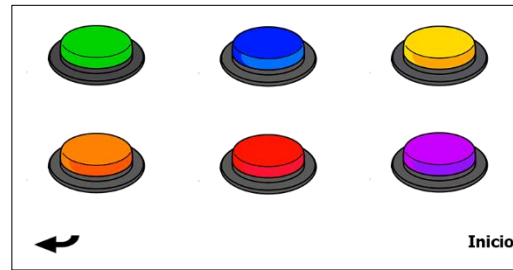
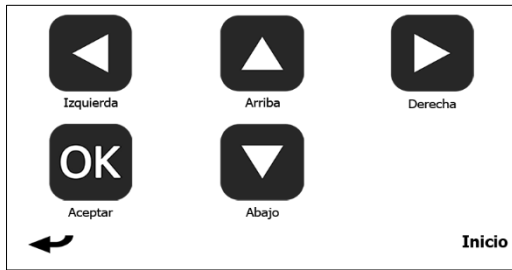


Encender / Apagar Reproducir Pausar

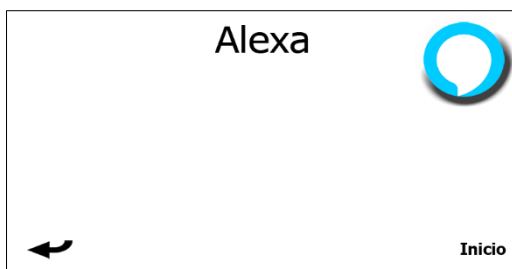




Detener Regresar Avanzar

Inicio



Anexo C. Plantillas para incorporación de Asistentes virtuales





Facultad de
Estudios
Superiores
Aragón

INGENIERÍA ELÉCTRICA ELECTRÓNICA

