INSTITUTO TECNOLÓGICO DE COSTA RICA

ESCUELA DE ELECTRÓNICA
CARRERA DE INGENIERÍA EN ELECTRÓNICA

# Diseño de un sistema de control multiprotocolo de bajo costo para personas con movilidad severamente limitada

Informe de Trabajo Final de Graduación para optar por el título de
Ingeniero en Electrónica con el grado académico de Licenciatura

**André Josué Moya Cordero**

Cartago, Diciembre, 2025

Diseño de un sistema de control multiprotocolo de bajo costo para personas con movilidad severamente limitada 

**INSTITUTO TECNOLÓGICO DE COSTA RICA**

**ESCUELA DE INGENIERÍA ELECTRÓNICA**

**TRABAJO FINAL DE GRADUACIÓN**

**ACTA DE APROBACIÓN**

**Defensa del Trabajo Final de Graduación**
**Requisito para optar por el título de Ingeniero en Electrónica**
**Grado Académico de Licenciatura**
**Instituto Tecnológico de Costa Rica**

El Tribunal Evaluador aprueba la defensa del Trabajo Final de Graduación denominado **Diseño de un sistema de control multiprotocolo de bajo costo para personas con movilidad severamente limitada**, realizado por el señor André Josué Moya Cordero y, hace constar que cumple con las normas establecidas por la Escuela de Ingeniería Electrónica del Instituto Tecnológico de Costa Rica.

Miembros del Tribunal Evaluador

SERGIO ARTURO MORALES HERNANDEZ (FIRMA)
PERSONA FISICA, CPF-02-0455-0934.
Fecha declarada: 01/12/2025 06:47:49 PM
Esta es una representación gráfica únicamente, verifique la validez de la firma.

Lic.-Ing. Sergio Morales Hernández

TEC | Tecnológico de Costa Rica
Firmado digitalmente por HUGO ANDRES SANCHEZ ORTIZ (FIRMA)
Fecha: 2025.12.01 12:57:29 +01'00'

M.Sc.-Ing. Hugo Sánchez Ortiz

PAOLA VEGA CASTILLO (FIRMA)
PERSONA FISICA, CPF-01-0937-0493.
Fecha declarada: 02/12/2025 10:58:42 AM
Esta representación visual no es fuente de confianza. Valide siempre la firma.

Dr.-Ing. Paola Vega Castillo
Profesora asesora

Cartago, 27 de noviembre, 2025

## Declaratoria de Autenticidad

Yo, André Josué Moya Cordero, estudiante de la carrera de Ingeniería en Electrónica del Instituto Tecnológico de Costa Rica, declaro bajo juramento que el presente documento titulado "Diseño de un sistema de control multiprotocolo de bajo costo basado en seguimiento ocular" es una producción intelectual íntegra y original de mi autoría. Cualquier información tomada de fuentes externas ha sido debidamente citada y referenciada. Reconozco que cualquier acto de plagio o falta de honestidad académica podrá ser sancionado de acuerdo con las normativas institucionales vigentes.

André Josué Moya Cordero
Cartago, 2 de diciembre de 2025
Céd. 117140097

# Resumen

Este proyecto presenta un sistema de interacción accesible y de bajo costo que permite a los usuarios controlar dispositivos domésticos utilizando únicamente el seguimiento ocular y la detección de parpadeos. La solución propuesta integra una interfaz gráfica controlada con la mirada, la cual envía solicitudes a un módulo central capaz de comunicarse mediante protocolos Wi-Fi, Bluetooth, infrarrojo y conexiones cableadas. Al eliminar la necesidad de interacción física, el sistema busca apoyar a personas con movilidad severamente limitada, mejorando su autonomía y seguridad en entornos cotidianos. El proyecto abarca los fundamentos teóricos de la interacción basada en la mirada, el diseño e implementación del sistema, y una evaluación experimental de su desempeño, seguida de un análisis de resultados y recomendaciones para trabajos futuros.

**Palabras clave:** Seguimiento ocular, detección de parpadeos, accesibilidad, sistema de bajo costo, control multiprotocolo, interfaz controlada con la mirada, control para casa inteligente.

# Abstract

This project presents a low-cost and accessible human–computer interaction system that enables users to control household devices using eye tracking and blink detection. The proposed solution integrates an eye-controlled graphical that sends requests to a central module capable of communicating through multiple protocols, including Wi-Fi, Bluetooth, infrared, and wired connections. By eliminating the need for physical interaction, the system aims to support individuals with severe mobility limitations, enhancing their autonomy and safety in daily environments. The project covers the theoretical foundations of gaze-based interaction, the design and implementation of the system, and an experimental evaluation of its performance, followed by an analysis of results and recommendations for future improvements.

**Keywords:** eye-tracking, assistive technology, human–computer interaction, low-cost system, multiprotocol control, smart home control.

# Índice general

# Capítulo 1

# Contexto

## 1.1. Introducción

Las personas con movilidad severamente limitada, como aquellas que padecen síndrome de enclaustramiento (locked-in syndrome) [12], enfrentan un nivel extremo de dependencia debido su incapacidad de mover el cuerpo. En la muchos casos solo pueden mover los ojos, lo que hace imposible realizar tareas cotidianas sin ayuda de un cuidador Esta situación genera riesgos significativos ante emergencias o ausencia del cuidador y que la persona queda completamente indefensa. Esta falta de autonomía también afecta negativamente la salud mental y la calidad de vida del paciente. Existen múltiples tecnologías orientadas a mejorar la autonomía de personas con movilidad limitada, incluyendo sistemas comerciales de seguimiento ocular o control por voz, pero muchas de estas soluciones tienen un costo alto o suelen necesitar componentes especializados que sin conocimientos técnicos se vuelve complicado utilizarlos. Por ejemplo, el Tobii Dynavox I-Series [27] ofrece un sistema avanzado de comunicación aumentada, pero su costo supera los 10 000 dólares, mientras que dispositivos más accesibles, como el Tobii Eye Tracker 5 [28], requieren software adicional, carecen de reconocimiento de parpadeo y no permiten controlar dispositivos físicos del entorno. En este contexto, en la Escuela de Ingeniería Electrónica del Instituto Tecnológico de Costa Rica se busca desarrollar un sistema orientado a brindar mayor autonomía a personas con movilidad severamente limitada mediante el control de dispositivos domésticos utilizando únicamente comandos derivados del seguimiento ocular y detección de parpadeos. Como parte del entorno tecnológico del proyecto, se aprovecha el software de seguimiento ocular de GazeFollower desarrollado por Gancheng Zhu de la Universidad de Zhejiang, el cual permite inicializar seguimiento ocular con tan solo una computadora y una cámara, la cual puede ser la de las computadoras portátiles, lo que la hace muy accesible.

### 1.1.1. Estructura del documento

Este proyecto se estructura en seis capítulos, cada uno abordando un aspecto específico del desarrollo del sistema.

- **Capítulo 1** presenta el contexto general del proyecto, su propósito, la relevancia de desarrollar una solución accesible y de bajo costo, así como una visión general del sistema propuesto.
- **Capítulo 2** reúne los fundamentos teóricos y el contenido técnico necesario para el diseño y la comprensión del funcionamiento del sistema, incluyendo conceptos relacionados con seguimiento ocular y tecnologías de control de dispositivos.
- **Capítulo 3** describe el diseño e implementación del sistema, detallando los componentes de software y hardware, la arquitectura general y los procesos de integración.

- **Capítulo 4** presenta los resultados experimentales obtenidos para evaluar el desempeño y la funcionalidad del sistema.
- **Capítulo 5** ofrece un análisis de los resultados, discutiendo las ventajas, viabilidad y limitaciones
- **Capítulo 6** expone las conclusiones finales y recomendaciones para trabajos futuros.

## 1.2. Problemática General

A pesar de los avances en accesibilidad digital, aún existe un vacío significativo en tecnologías de bajo costo que permitan a personas con movilidad severamente limitada controlar dispositivos cotidianos mediante movimientos oculares. Actualmente existen soluciones como el Tobii Dynavox I-Series [27], el cual es un dispositivo de asistencia tecnológica que combina un hardware robusto de seguimiento ocular de alta presición. El dispositivo incluye software avanzado de comunicación y control para comunicación alternativa y control de computadoras, pero al utilizar hardware especializado tiene un alto costo y es poco accesible para la mayoría de usuarios. Otras soluciones más accesibles como el Tobii Eye Tracker 5 [28], el cual es un rastreador ocular principalmente para el mercado de los video juegos de alto rendimiento, pero requiere software adicional y no permiten interactuar con dispositivos físicos ni detectar parpadeos de forma nativa lo cual dificulta su adopción en entornos domésticos para control de actuadores.

## 1.3. Justificación del Proyecto

La necesidad de contar con soluciones accesibles, escalables y de bajo costo para el control de dispositivos domésticos motiva el desarrollo de un sistema que pueda ser utilizado por personas con movilidad severamente limitada. A diferencia de muchas alternativas comerciales, que requieren hardware especializado o se restringen a un único protocolo de comunicación, este proyecto busca implementar una herramienta capaz de operar con múltiples tecnologías del entorno (Wi-Fi, Bluetooth, infrarrojo y control cableado).

El hecho de que el sistema pueda ser controlado únicamente mediante la dirección de la mirada y la detección de parpadeos permite ofrecer una alternativa accesible para personas con discapacidades motoras severas o con movilidad muy limitada, lo que contribuye a mejorar la autonomía y la seguridad del usuario. Al combinar esta forma de control con la capacidad de operar dispositivos que utilizan distintos protocolos de comunicación, el sistema se convierte en una herramienta flexible y adaptable a una amplia variedad de situaciones y necesidades reales.

## 1.4. Descripción General de la Solución

El sistema desarrollado consiste en una interfaz gráfica controlada mediante seguimiento ocular y detección de parpadeos, conectada a un módulo principal capaz de enviar comandos a diferentes dispositivos domésticos mediante protocolos como Wi-Fi, Bluetooth, infrarrojo o conexión cableada de bajo costo, accesible y escalable.

# Capítulo 2

# Fundamentos teóricos y prácticos

Este capítulo presenta los conceptos y tecnologías esenciales que sustentan el desarrollo del sistema propuesto. Se describen los principales protocolos de comunicación utilizados para el control de dispositivos, las características relevantes de la Raspberry Pi como plataforma embebida, los componentes electrónicos empleados y las bibliotecas de software necesarias para la implementación. Finalmente, se introducen las bases teóricas del seguimiento ocular y las técnicas de procesamiento de señal aplicadas para mejorar la estabilidad del cursor. Estos fundamentos permiten contextualizar las decisiones de diseño y comprender el funcionamiento general del sistema desarrollado.

### 2.0.1. Protocolos de comunicación

#### Conexión por infrarrojo

La comunicación por infrarrojo se basa en la transmisión de datos mediante luz infrarroja modulada (IR)[10] , generalmente a 38 kHz. Este método es ampliamente utilizado en controles remotos debido a su simplicidad y bajo costo.

**Protocolo NEC** El protocolo NEC es uno de los estándares más comunes en dispositivos controlados por Infarrrojo. Los datos se suelen enviar en tramas de 32 bits, en donde se inicia con un pulso de 9 ms seguido en 1, seguidos por un espacio de 4.5 ms en 0. Los bits se representan de la forma:

- Bit 0: pulso de 562 µs + espacio de 562 µs.
- Bit 1: pulso de 562 µs + espacio de 1.687 ms.

El NEC estándar suele tener enviar las tramas de código en este formato:

1. Un byte que representa la dirección del dispositivo.
2. Un byte con la dirección invertida, utilizado como mecanismo de verificación de errores.
3. Un byte correspondiente al comando enviado.
4. Un byte con el comando invertido, también empleado para la detección de errores.

Existen variantes como NEC Extendido, RC5 de Philips y SIRC de Sony, caracterizadas por diferencias en formato, longitud y frecuencia de modulación. Muchos fabricantes emplean protocolos propietarios, lo que puede dificultar la decodificación y reproducción genérica de sus señales.

**Controles universales** Los controles universales son dispositivos permiten controlar diferentes equipos desde un único control. Los controles universales funcionan mediante el aprendizaje o la inclusión de bases de datos de códigos IR. Su comportamiento consiste en reproducir la secuencia exacta de pulsos y espacios del control original, respetando tanto la frecuencia portadora como la codificación temporal de cada bit, lo que permite controlar muchos dispositivos infrarrojo diferentes.

#### Bluetooth

Bluetooth es una tecnología inalámbrica de corto alcance que opera en la banda de 2.4 GHz [9] utilizando salto de frecuencia (FHSS) para reducir interferencias. Cada dispositivo posee una dirección MAC única que permite su identificación en el proceso de emparejamiento.

Los dispositivos pueden formar *piconets* donde un nodo cumple el rol de maestro y el resto actúa como esclavos. Existen dos variantes principales:

- **Bluetooth Clásico (BR/EDR):** orientado a transmisión continua de datos y perfiles como A2DP o AVRCP.
- **Bluetooth Low Energy (BLE):** diseñado para bajo consumo y comunicación esporádica.

Muchos dispositivos solo permiten una conexión activa por enlace, lo que limita el control simultáneo desde múltiples fuentes. Algunos equipos modernos permiten conexiones múltiples, dependiendo del hardware y firmware.

#### Wi-Fi y HTTP

La comunicación Wi-Fi se basa en los estándares IEEE 802.11[8] y utiliza la pila TCP/IP. Sobre ella se ejecuta HTTP, un protocolo simple y ampliamente soportado para intercambio de datos estructurados mediante métodos GET y POST.

En muchos dispositivos comerciales, las interfaces de control no están disponibles para el usuario debido a restricciones impuestas por motivos de seguridad o decisiones de diseño del fabricante. Esta limitación obliga a utilizar aplicaciones propietarias o servicios remotos que dependen directamente de la empresa propietaria.

Cuando el sistema se implementa de forma local es posible emplear HTTP como mecanismo de comunicación para desarrollar soluciones que no dependan de plataformas externas y que permitan un control completo dentro de la red local.

### 2.0.2. Raspberry Pi

La Raspberry Pi es un computador de placa reducida orientado a proyectos embebidos.[24] Integra procesador, memoria, puertos USB, HDMI, conectividad inalámbrica y un cabezal de 40 pines GPIO. Su versatilidad permite controlar dispositivos electrónicos, procesar datos y ejecutar sistemas operativos basados en Linux.

#### Sistema operativo y entorno Linux

El sistema empleado ejecuta Raspberry Pi OS Bookworm versión 12, con entorno de usuario de 32 bits y kernel de 64 bits. Esta combinación permite cierto aprovechamiento de la arquitectura de 64 bits, pero puede generar incompatibilidades con bibliotecas precompiladas.

Linux gestiona el hardware mediante controladores en /dev, mientras que los GPIO se controlan desde Python usando bibliotecas como pigpio y RPi.GPIO.

#### Configuración de red

La gestión de Wi-Fi y Bluetooth se realiza mediante herramientas estándar:

- `nmcli` para administración de redes Wi-Fi.
- `wpa_supplicant` y NetworkManager para conexión automática.
- `bluetoothctl` y la pila BlueZ para gestión de adaptadores y emparejamientos.

Esto permite automatizar conexiones y controlar la red desde scripts o aplicaciones de Python.

#### Control de pines GPIO

Los GPIO de la Raspberry Pi operan a 3.3 V y pueden suministrar aproximadamente 16 mA por pin, con un límite total cercano a 50 mA. La Raspberry Pi también incluye pines de alimentación de 5 V y 3.3 V, así como pines GND, que pueden entregar corrientes considerablemente mayores al no estar limitados por el controlador de los GPIO, sino por la capacidad de la fuente de alimentación. Estos pines se utilizan para alimentar módulos externos, sensores o circuitos que requieran más corriente de la que un GPIO puede proporcionar.

La placa no incorpora entradas analógicas, por lo que es necesario utilizar un convertidor ADC en caso de trabajar con señales analógicas.

Los pines soportan funciones I2C, SPI, UART y PWM. Su control se realiza principalmente mediante:

- **pigpio**: orientado a temporización precisa, generación estable de PWM y control avanzado de señales.
- **RPi.GPIO**: utilizado para control digital básico y operaciones de entrada/salida simples.

### 2.0.3. Hardware

#### Receptor infrarrojo VS1838

Dispositivo receptor diseñado para captar señales de infrarrojo moduladas a 38 kHz y convertirlas en tramas de bits[7]. Es ampliamente utilizado de recepción y control con señales infrarrojo debido a su robustez y facilidad de integración.

#### Driver ULN2003

Es un arreglo Darlington capaz de manejar cargas de hasta 500 mA por canal[26]. Ofrece aislamiento eléctrico y permite controlar dispositivos de mayor potencia como motores, relés o LEDs de alta corriente utilizando una fuente externa.

#### Transistor 2N3904

Transistor NPN de propósito general con frecuencia de conmutación de 300 MHz[14]. Su velocidad lo hace apto para modular LEDs infrarrojos a 38 kHz.

#### LED infrarrojo

LED de 940 nm[31] utilizado para transmitir señales moduladas permite generar pulsos precisos compatibles con protocolos IR comunes.

### 2.0.4. Bibliotecas de Python

Para el desarrollo del proyecto se utilizaron bibliotecas de python que se describen a continuación:

#### Flask[16]

Microframework diseñado para implementar servidores web. Maneja solicitudes HTTP a través de rutas y admite ejecución multithread. Dado que ciertas funciones bloqueantes pueden detener el servidor, permite delegar procesos a subprocesos o hilos independientes.

#### pigpio[3]

Biblioteca orientada a control preciso de GPIO. Requiere el demonio `pigpiod` de la Raspberry pi para permitir acceso concurrente y alta precisión temporal, lo cual es fundamental para generar o medir señales moduladas.

#### RPi.GPIO[_pypipypi]

Librería nativa para control digital básico de los GPIO de la Raspberry pi.

#### BlueZ[6]

Implementación oficial de Bluetooth en Linux. Incluye herramientas como `bluetoothctl` para gestión de dispositivos.

#### time[22]

Biblioteca estándar para medición de intervalos y retardos, utilizada frecuentemente en comunicaciones sensibles al tiempo, como el infrarrojo.

#### requests[19]

Librería de alto nivel para realizar peticiones HTTP mediante GET, POST u otros métodos.

#### subprocess[21]

Permite ejecutar comandos del sistema operativo y controlar su entrada y salida estándar.

#### shlex[?]

Convierte textos en listas de argumentos seguros, facilitando la ejecución de comandos complejos mediante `subprocess`.

#### pygame[20]

Conjunto de librerías para desarrollo de interfaces gráficas interactivas y captura de eventos.

### 2.0.5. Software auxiliar en Linux

Para el desarrollo del proyecto se utilizaron programas y aplicaciones de Linux, las cuales se describen a continuación:

#### VLC[29]

Reproductor multimedia con soporte para protocolos como RTMP, RTSP y HTTP. En Linux puede ejecutarse desde consola mediante `cvlc`, permitiendo automatizar transmisión o reproducción de audio sin necesidad de interfaz gráfica.

#### bluetoothctl[5]

Herramienta de BlueZ para gestionar dispositivos Bluetooth desde la terminal, utilizada para escaneo, emparejamiento, conexión y administración de perfiles.

#### pigpiod[3]

Demonio que habilita el acceso concurrente a los pines GPIO, necesario para el funcionamiento de la librería `pigpio`.

### 2.0.6. Software de seguimiento ocular *GazeFollower*

*GazeFollower*[32] es un sistema de seguimiento ocular de código abierto que emplea redes neuronales para estimar el punto de fijación del usuario en una pantalla a partir de una cámara convencional. El modelo procesa el rostro y los ojos para predecir las coordenadas $(x, y)$ de la mirada.

Incluye herramientas de calibración, registro en tiempo real y modelos entrenados con más de siete millones de imágenes. Aunque permite obtener resultados funcionales en escenarios experimentales, su precisión depende de la iluminación, la calibración y las condiciones de captura.

### 2.0.7. Procesamiento de señal para seguimiento ocular

#### Filtro exponencial

El filtro exponencial o *Exponential Moving Average* (EMA)[13] es un método de suavizado que combina el valor actual de la señal con el valor filtrado anterior:

$$y_t = \alpha x_t + (1 - \alpha)\, y_{t-1}$$

Valores bajos de $\alpha$ generan mayor estabilidad y menor sensibilidad a cambios rápidos mientras que valores altos permiten una respuesta más ágil. Es ampliamente utilizado en control, visión por computadora y seguimiento de objetos para reducir ruido y oscilaciones en mediciones.

#### Métodos de asistencia de puntería (*Aim Assist*)

El *aim assist* [15] [11] agrupa técnicas destinadas a facilitar la selección de elementos en una interfaz cuando el dispositivo de entrada presenta inestabilidad o limitaciones de precisión . Entre los métodos más comunes se encuentran:

- **Atracción al objetivo:** ajusta la posición del puntero cuando se aproxima a un elemento interactivo.
- **Magnificación local:** aumenta visualmente el área cercana al puntero para facilitar la selección.
- **Filtrado adaptativo:** modifica la sensibilidad del puntero según su proximidad al objetivo.

Estas técnicas son útiles en sistemas de accesibilidad y control asistido.

# Capítulo 3

# Diseño e implementación del sistema

El sistema de domótica se diseñó para ser controlado mediante una interfaz que utiliza de cursor un *eye tracker*, la interfaz envía comandos a través de un cliente HTTP hacia una Raspberry Pi que funciona como servidor HTTP utilizando Flask. Para la implementación, se desarrollaron diversos módulos de control:

- Módulo Infrarrojo
- Módulo Bluetooth
- Módulo Wi-Fi
- Módulo Cableado
- Servidor HTTP
- Cliente HTTP
- Interfaz
- Cursor (eye tracker)

Este diseño permite integrar distintos protocolos de comunicación (Bluetooth, Wi-Fi, infrarrojo y control directo por GPIO) bajo una misma interfaz de usuario accesible mediante seguimiento ocular.

### 3.0.1. Orden de programación

El proyecto se organiza en la carpeta `TFG_Andre_Moya`, que contiene los módulos principales. El archivo `tfg.py` funciona como servidor HTTP basado en Flask y se encarga de invocar las funciones de control en las demás carpetas según el comando recibido.

El orden de comunicación entre módulos del software incluido en el servidor de la Raspberry Pi se representa en la figura 3.1.

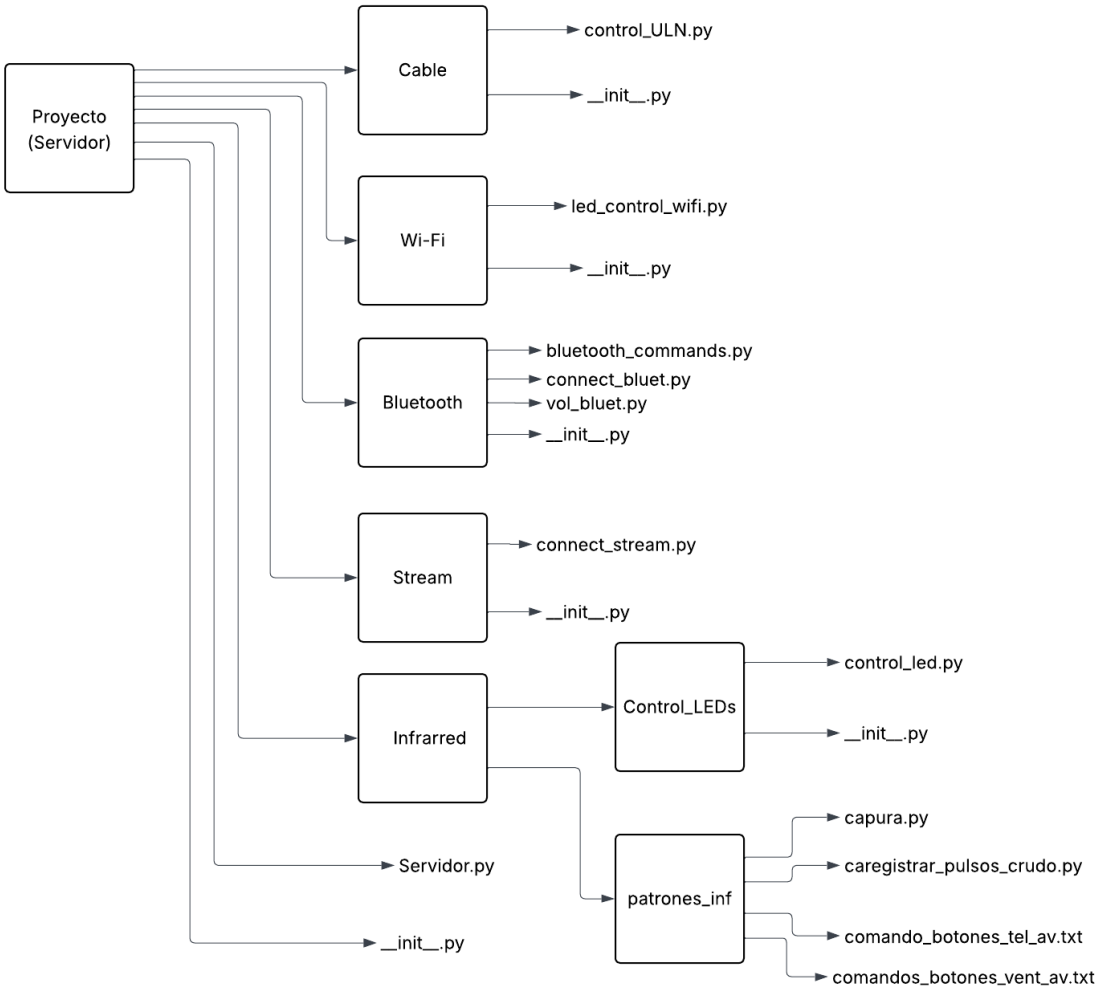


Figura 3.1: Diagrama de orden de programación del servidor Raspberry Pi

La orden de comunicación entre módulos del software incluidos en la computadora portátil que posee el software de seguimiento ocular, que en este caso es el cliente HTTP se presenta en la figura 3.2

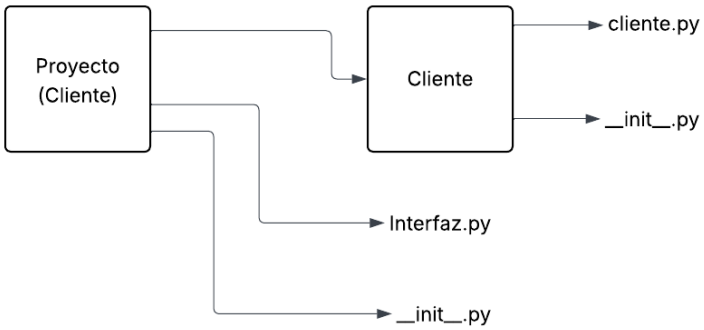


Figura 3.2: Diagrama de orden de programación del cliente en la computadora portátil

### 3.0.2. Módulo Infrarrojo

El objetivo de este módulo es controlar dispositivos electrónicos que utilicen señales infrarrojas, como ventiladores, televisores, radios o sistemas de ventilación. En este proyecto se controla un televisor Roku y un ventilador Basic Living.

Para esto fue necesario utilizar la biblioteca pigpio, y para correr el programa en la Raspberry Pi, se ejecutó el siguiente comando en la terminal:

```
1    sudo apt install pigpio python3-pigpio
```

Esto instaló lo necesario para el control infrarrojo.

#### Patrones Infrarrojo

Para implementar el control, se identificó el protocolo infrarrojo de cada dispositivo utilizando el receptor VS1838. Con el código `captura.py`, se analizaron las tramas de los controles. De lo cual se encontró:

- El televisor utiliza el protocolo NEC extendido, compuesto por:
  - 16 bits para la dirección del dispositivo
  - 8 bits para el comando
  - 8 bits para el complemento del comando
- El ventilador no utiliza el protocolo NEC, pero emplea la misma portadora de 38 kHz.

Para el televisor, se presionó cada botón al menos 10 veces y se promediaron las tramas para minimizar errores temporales. Los datos se almacenaron en un archivo de texto con la estructura `comando: [[estado 1, tiempo 1],[estado 2,tiempo 2],(...),[estado n, tiempo n]]` y se guardaron en `comando_botones_tel_av.txt`.

El ventilador siguió el mismo proceso, pero se utilizo el código de `caregistrar_pulsos_crudos.py`, se presionó cada botón al menos 10 veces y luego se guardaron los pulsos con el formato anterior en `comando_botones_vent_av.txt`.

#### Control Infrarrojo

Se desarrolló el código `controlled.py`, que lee los archivos de comandos y los carga en un diccionario llamado `mapa_botones`, de la forma:

- `tel: botones_tel`
- `vent: botones_vent`

Cada subdiccionario almacena la lista de estados y tiempos de cada botón. Para enviar un patrón IR, se utiliza la función `control_led(disp, boton, map_botones)`, que:

1. Obtiene el diccionario de botones correspondiente al dispositivo que se desea controlar, especificado por el parámetro `disp`. De ahí que disp es una de las dos claves de mapa_botones
2. Convierte los datos del diccionario a enteros mediante la función `parsear_patron`. Como los patrones IR se guardan en archivos de texto, al importarlos en Python se leen como cadenas (`string`). `parsear_patron` transforma estos valores en enteros (`int`), lo cual es necesario para poder modular correctamente la señal IR.

3. Modula la señal a 38 kHz utilizando la función `crear_pulsos_modulados`. La señal modulada se guarda en la variable `wf` y representa la forma de onda IR lista para ser enviada.

4. Envía la señal en bloques (*chunks*) usando las funciones `pi.wave_add_generic()` y `pi.wave_send_once`. La división en bloques se realiza para evitar exceder el límite de memoria de `pigpio`. Cada bloque se envía de manera secuencial, lo que permite transmitir la señal completa sin errores, incluso para patrones largos o complejos como los del ventilador con protocolo desconocido.

El servidor HTTP solo ejecuta dos funciones: `cargar_botones()` y `control_led()`, cambiando únicamente los parámetros de dispositivo y botón según sea necesario.

### Circuito del Receptor IR

El receptor infrarrojo se implementó utilizando un módulo **VS1838**, el cual se alimenta con 3.3 V. Para garantizar lecturas correctas en los pines GPIO de la Raspberry Pi, se incorporó una resistencia pull-up de 10 kΩ entre la salida del módulo y Vcc. Este diseño permite que la señal entregada por el VS1838 sea lógica alta cuando no se detecta infrarrojo y baja cuando el haz infrarrojo es capturado. El circuito se muestra en la Figura 3.3 y los componentes utilizandos en la tabla 3.1:

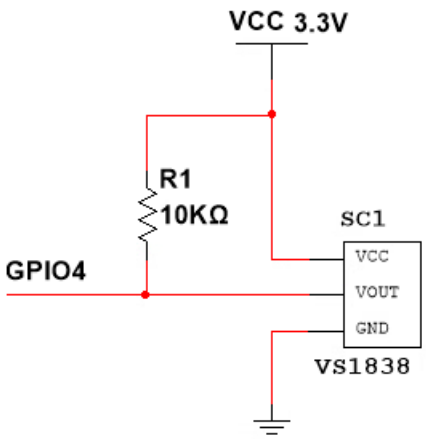


Figura 3.3: Circuito del receptor infrarrojo con módulo VS1838 y conexión al GPIO de la Raspberry Pi.

Cuadro 3.1: Componentes del receptor IR

| **Componente** | **Valor** |
|---|---|
| VS1838 | N.A. |
| Resistencia pull-up | 10 kΩ |

### Control del LED emisor IR

Para el control de infrarojo se utilizó un LED emisor de 950 nm. Como los pines GPIO de la Raspberry Pi no pueden suministrar la corriente necesaria para el LED directamente, se utilizó un transistor **2N3904** para amplificar la corriente. Se utilizaron los componentes que se observan en la tabla 3.2

Cuadro 3.2: Componentes del emisor IR

| **Componente** | **Valor** |
|---|---|
| LED 950 nm | N.A. |
| Transistor 2N3904 | N.A. |
| Resistencia pull-down | 10 kΩ |
| Resistencia de base | 500 Ω |
| Resistencia de colector | 50 Ω |

La resistencia pull-down mantiene la entrada del transistor en nivel bajo cuando no hay señal, mientras que las resistencias de base y de colector aseguran una correcta polarización del transistor.

Para calcular la corriente del colector en saturación ($I_c$), la cual es la corriente del LED y de la resistencia del colector, se hace de la siguiente manera:

$$I_c = \frac{V_R}{R_c} = \frac{V_{cc} - V_{LED} - V_{EC}}{R_c} = \frac{5 - 1{,}6 - 0{,}2}{50} = 64 \text{ mA} \tag{3.1}$$

Donde:

- $V_R$ es el voltaje en la resistencia del colector.
- $V_{cc} = 5$ V es la alimentación del circuito.
- $V_{LED} = 1{,}6$ V es la caída de voltaje del LED en conducción obtenido de la hoja de datos del LED
- $V_{EC} = 0{,}2$ V es la caída de tensión entre el colector y el emisor en saturación obtenido de la hoja de datos del transitor.
- $R_c = 50\Omega$ es la resistencia del colector.

Este diseño asegura que el LED funcione en saturación cuando la base del transistor recibe un voltaje alto de 3.3 V desde la Raspberry Pi, suministrando así la corriente adecuada sin exceder la capacidad del pin GPIO y sin sobrepasarse en la corriente máxima del LED y el transistor. El circuito completo se muestra en la Figura 3.4.

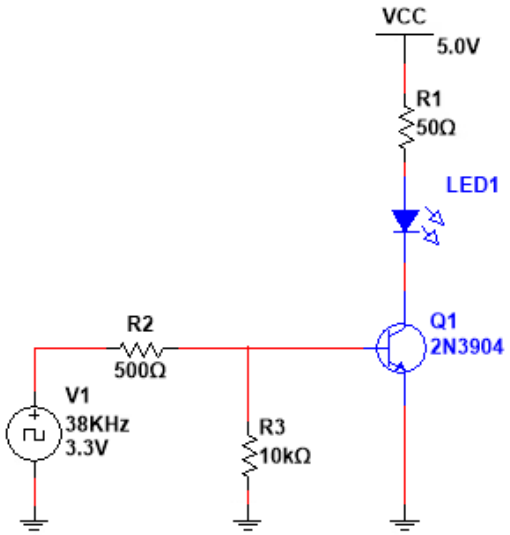


Figura 3.4: Circuito del emisor infrarrojo con LED de 950 nm y transistor 2N3904 para amplificación.

### 3.0.3. Módulo Bluetooth

Se utilizó un parlante JBL Flip 4 con Bluetooth Clásico. Este dispositivo solo permite recibir comandos del dispositivo que está transmitiendo audio en ese momento, lo que impide que la Raspberry Pi controle directamente el parlante si otro dispositivo está conectado reproduciendo audio. Para solucionar esto se implementó un flujo de audio indirecto: la tablet, computadora u otro dispositivo transmite el audio a la Raspberry Pi, que a su vez lo reproduce al parlante. De esta forma, la Raspberry Pi puede controlar la conexión y el volumen del parlante mientras reproduce el audio.

#### Estructura de Software Bluetooth

En la Carpeta Bluetooth hay tres archivos principales:

- `bluetooth_commands.py`
- `connect_bluet.py`
- `vol_bluet.py`

**connect_bluet.py:** Establece o finaliza conexión con la dirección MAC del parlante usando `bluetoothctl` y configura el *default sink* en PulseAudio, esto ayuda para no tener que utilizar la dirección MAC del parlante cada vez que se quiere controlar el parlante. Solamente se configura como dispositivo de defecto.

**vol_bluet.py:** Permite subir y bajar el volumen usando pactl set-sink-volume @DEFAULT_SINK@ <vol>.

**bluetooth_commands.py:** Integra las funciones anteriores y la función de connect_stream que se encuentra en la carpeta de Stream. Tiene la función dic_blue_commands(command, connect). En donde command puede ser:

- connect: Se conecta al parlante y al stream por VLC.
- disconnect
- +5 %: Sube el volumen 5 %
- -5 %: Baja el volumen -5 %
- vol: Sube el volumen al valor de vol.

Connect es una bandera que indica si el parlante esta conectado o desconectado.

#### Módulo de Transmisión de Audio (Stream)

Se encuentra en la carpeta Stream, en donde el script connect_stream.py se encarga de hacer la conexión a la transmisión, este utiliza subprocess y time para reproducir o detener el flujo RTMP de la transmisión.

**Configuración del Bluetooth en la Raspberry Pi**

Se abre la herramienta de configuración con el siguiente comando:

```
sudo raspi-config
```

Esto muestra una interfaz de texto donde es posible modificar diversas opciones del sistema.

Dentro de la interfaz, se debe navegar a **Interface Options** y seleccionar **P3 Bluetooth**. Luego, se habilita el Bluetooth eligiendo **Enable**.

Luego, se instalaron los paquetes necesarios para permitir el audio mediante Bluetooth:

```
sudo apt install pulseaudio pulseaudio-module-bluetooth bluez
```

Finalmente, se habilita y se arranca el servicio de Bluetooth con los siguientes comandos:

```
sudo systemctl enable bluetooth
sudo systemctl start bluetooth
```

Con esto, el servicio de Bluetooth queda activo y se iniciará automáticamente cada vez que se inicia la Raspberry Pi.

### 3.0.4. Módulo Wi-Fi

Debido a que no se dispone de componentes que puedan ser controlados por Wi-Fi de manera abierta (la mayoría requieren aplicaciones propietarias con comunicaciones encriptadas), fue necesario simular un dispositivo controlable por Wi-Fi. Para esto se utilizó un ESP32, que cuenta con antena Wi-Fi y permite configurar un servidor HTTP para recibir órdenes.

En la carpeta WiFi solo se incluye el archivo led_control_wifi.py, en el cual el usuario puede enviar solicitudes HTTP para encender o apagar el LED mediante la función:

$$control\_led\_wifi(control)$$

donde control puede ser on u off. Al recibir on el ESP32 pone el pin conectado al LED en alto (3.3 V) y al recibir off lo ponee en bajo. (0 V)

### 3.0.5. Módulo Cableado

El objetivo de este módulo es controlar dispositivos físicos conectados por cable a la Raspberry Pi. Para este proyecto se controlan un motor paso a paso, un LED, un buzzer pasivo y un buzzer activo mediante GPIO y señales PWM. Esto permite demostrar la viabilidad de controlar actuadores físicos y simular aplicaciones como dimmers, alarmas y movimiento controlado de un motor.

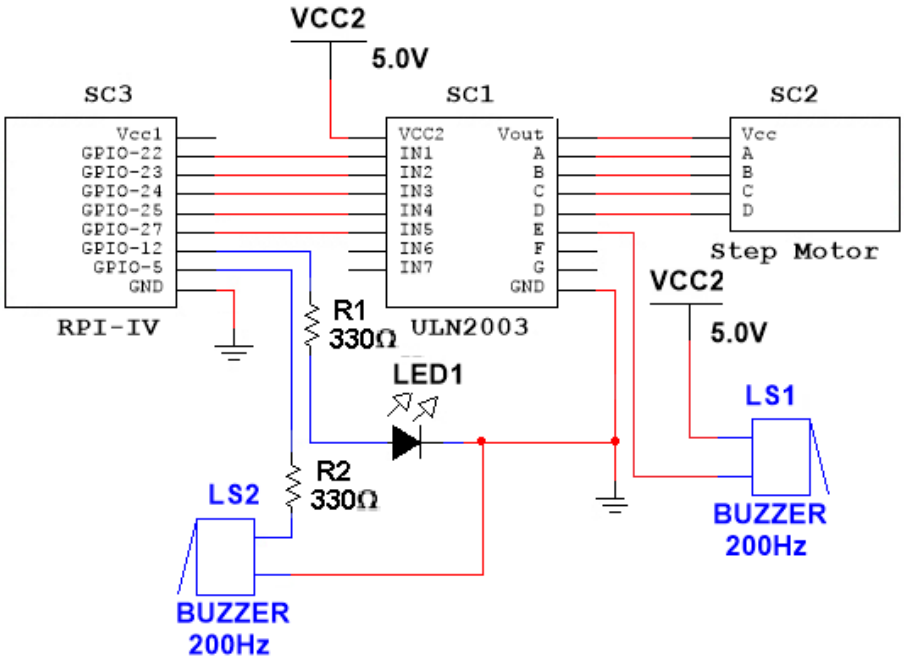


Figura 3.5: Diagrama de conexiones de los dispositivos cableados al ULN2003 y Raspberry Pi.

Para controlar todos los dispositivos conectados por cable se utiliza la función

control_cable(device, on, dat)

Donde:

- `device`: selecciona el dispositivo (1: Motor, 2: LED, 3: Buzzer pasivo, 4: Buzzer activo).
- `on`: Variable booleana que indica si se debe encender (`True`) o apagar (`False`) el dispositivo.
- `dat`: valor adicional según el dispositivo: Puede ser un entero:
  - Motor: ángulo a mover.
  - LED: ciclo de trabajo de la señal PWM.
  - Buzzer pasivo: frecuencia de la señal PWM.

  O ser un string de valor "test"
  En donde :
  - LED: variar el ciclo de trabajo de 0 % a 100 % en incrementos de 5 %.
  - Buzzer pasivo: variar la frecuencia de 200 Hz a 4000 Hz en incrementos de 200 Hz.

A continuación se detallan los dispositivos conectados y cómo se controlan mediante la función de control.

### Fuente externa

La Raspberry Pi tiene una limitación en la corriente que puede suministrar por sus pines GPIO. Por ello, para controlar los dispositivos sin depender de la capacidad de la Raspberry Pi, se utilizó un power bank de 5 V con las siguientes especificaciones:

Cuadro 3.3: Especificaciones del Power Bank utilizado

| **Parámetro** | **Valor** |
|---|---|
| Capacidad | 5400 mAh |
| Corriente máxima | 2.1 A |
| Voltaje | 5 V |

En la figura 3.5 se muestran todas las conexiones entre la Raspberry Pi, los pines del ULN2003, el motor paso a paso, los buzzer y el LED. En este esquema, LS1 corresponde al buzzer activo y LS2 al buzzer pasivo.

Este se utiliza como voltaje de alimentación del ULN2003, y comparte la tierra con el Raspberry Pi

### Control del Motor

El parámetro `dat` representa el ángulo deseado, que puede ser positivo o negativo. La cantidad de pasos a mover se calcula como:

$$steps = dat \cdot \frac{512}{360} \tag{3.2}$$

Para mover el motor, las bobinas se activan en un orden específico mediante el ULN2003. Este patrón permite que el motor gire al ángulo indicado por el usuario. Actualmente el control se realiza ingresando directamente el ángulo, aunque el sistema podría adaptarse para limitar movimientos en aplicaciones como camas para personas con movilidad reducida.

### Control PWM para LED

El LED se controla mediante PWM. Variando el ciclo de trabajo de la señal PWM se ajusta la potencia promedio consumida por el LED, modificando así su intensidad luminosa y emulando un dimmer.

### Control PWM para Buzzer pasivo

El buzzer pasivo se controla variando la frecuencia de la señal PWM. La frecuencia se ajusta dentro del rango audible (20 Hz – 20 kHz), permitiendo al usuario generar diferentes tonos según sea necesario.

### Control Buzzer activo

El buzzer activo se controla con el GPIO. Cuando el GPIO está en alto, el ULN2003 conecta el buzzer a tierra y suena. Cuando el GPIO está en bajo, la salida queda abierta y el buzzer se apaga.

### 3.0.6. Servidor HTTP

El objetivo del servidor HTTP es proporcionar un punto de comunicación centralizado para controlar los dispositivos conectados a la Raspberry Pi mediante solicitudes HTTP enviadas desde un cliente. La Raspberry Pi actúa como servidor, mientras que la computadora portátil funciona como cliente que envía comandos y subcomandos en formato `dispositivo/subcomando`. Por ejemplo, para encender un LED tipo dimmer con un ciclo de trabajo de 30 %, se envía la solicitud `cab_led/30`. El servidor interpreta la solicitud, determina el dispositivo y el subcomando, y ejecuta la función de control correspondiente, por ejemplo: `control_cable(device=3, on=True, dat=30)`.

Esto permite que la tablet controle todos los dispositivos conectados a la Raspberry Pi de forma centralizada y en tiempo real, sin necesidad de manipular los GPIO o interfaces de hardware directamente.

#### Comandos y subcomandos

El servidor reconoce los siguientes comandos:

- `inf_tel`: Control del televisor por infrarrojo
- `inf_vent`: Control del ventilador por infrarrojo
- `blue_par`: Control del parlante Bluetooth
- `cab_buz`: Control de los buzzers conectados por cable
- `cab_mot`: Control del motor paso a paso
- `cab_led`: Control del LED tipo dimmer
- `wifi_luz`: Control de la luz simulada por Wi-Fi

Cada comando admite subcomandos específicos según el dispositivo como se resumen en el cuadro 3.4 :

Cuadro 3.4: Comandos y Subcomandos del Servidor HTTP

| **Comando** | **Subcomandos** |
|---|---|
| `inf_tel` | `on/off`, `return`, `vol_down`, `vol_up`, `left`, `right`, `down`, `exit` |
| `inf_vent` | `timer`, `mode`, `osc`, `off`, `onspeed` |
| `blue_par` | `connect`, `disconnect`, `+5%`, `-5%`, `vol_set` |
| `cab_buz` | `act_on`, `act_off`, `freq`, `pas_test`, `pas_off` |
| `cab_mot` | `ang`, `mot_off` |
| `cab_led` | `duty`, `led_test`, `led_off` |
| `wifi_led` | `on`, `off` |

El servidor se implementa utilizando la librería de Flask y se encarga de recibir y procesar las solicitudes enviadas por el cliente HTTP. Se importan todas las funciones necesarias para controlar los dispositivos conectados a la Raspberry Pi, como `control_led`, `control_cable` y `control_led_wifi`.

Se define la variable global `connect_blue` para mantener el estado de conexión del parlante Bluetooth y se crea `mapa_botones`, que contiene los comandos de los controles infrarrojos para el televisor y el ventilador, el cual se utiliza cuando es necesario enviar órdenes IR.

Cuando el servidor recibe una solicitud, valida que el comando esté dentro de los comandos permitidos, identifica el dispositivo y el subcomando correspondiente, y llama a la función de control adecuada. Por ejemplo, si se recibe la solicitud:

/cab_buz/act_on

se ejecuta la función:

```
1   control_cable(device=3, on=True, dat=0)
```

El servidor se pone a la escucha de solicitudes HTTP desde cualquier dispositivo de la red mediante:

```
2   app.run(host="0.0.0.0", port=8000, threaded=False)
```

quedando listo para controlar todos los dispositivos conectados a la Raspberry Pi en tiempo real.

#### Cliente HTTP

El cliente HTTP se ejecuta en la computadora portátil y utiliza la librería `requests` de Python para enviar comandos y subcomandos al servidor. Su función principal es proporcionar un punto de comunicación que permita al usuario controlar los dispositivos sin interactuar directamente con el hardware.

El cliente permanece activo en un ciclo `while` hasta que el usuario decide salir. La IP del servidor se almacena en una variable, y mediante un menú en la terminal se selecciona el dispositivo y el subcomando que se desea ejecutar. Por ejemplo, al elegir el parlante Bluetooth, se pueden conectar o desconectar, cambiar el volumen en incrementos de 5 %, o establecer un valor específico. El cliente construye la URL en el formato:

```
3   http://IP_DEL_SERVIDOR:PUERTO/dispositivo/subcomando
```

y envía la solicitud con `requests.get()`. La respuesta del servidor se muestra en la terminal, confirmando la ejecución de la acción.

### 3.0.7. Módulo Interfaz

El objetivo de este módulo es proporcionar al usuario una interfaz gráfica e interactiva para controlar los dispositivos conectados a la Raspberry Pi con un software de seguimiento ocular y detección de parpadeo. La interfaz se desarrolló utilizando la librería `pygame`. Cada dispositivo que puede controlarse está representado por un botón.

Al seleccionar un dispositivo, se abre una subpantalla con los subcomandos asociados a dicho dispositivo. Cuando el usuario selecciona una acción, la interfaz construye automáticamente el comando y subcomando correspondiente, que se envía al cliente HTTP para su transmisión al servidor, tal como se describe en las secciones dedicadas al cliente y servidor HTTP.

#### Adaptación al control por seguimiento ocular

El método principal de control es mediante un software de seguimiento ocular (eye tracker). Para ello, se implementó un mecanismo que sustituye las coordenadas del ratón por las coordenadas `x` y `y` proporcionadas por el eye tracker, de manera que todos los elementos de la interfaz puedan seleccionarse únicamente con la mirada. De ahí que se define un cursor del eye tracker que representa estas coordenadas.

### Estructura general de la interfaz

La interfaz se organiza en varias pantallas o pestañas. En la pantalla principal se presentan los botones de acceso a cada dispositivo controlable: televisor, ventilador, buzzer, motor, dimmer, luz y parlante, además se tienen 3 botones de dispositivos que se pueden añadir en el futuro, junto con la opción de salir. Se puede observar el menú principal de la interfaz en la figura 3.6, el punto rojo depende de las coordenadas x y y del eye tracker y el punto azul es un puntero promedio que tiene función de asistencia de puntería. Como se ve, cuando el puntero promedio esta encima de un botón el botón cambia de color a un color más oscuro. Al seleccionar un botón, se abre una subpantalla con los subcomandos específicos de ese dispositivo.

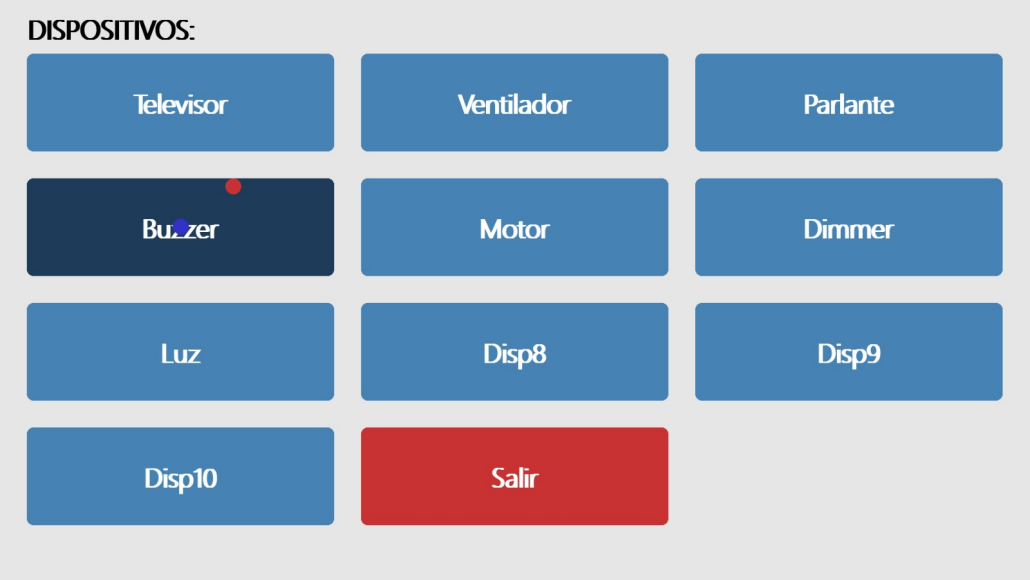


Figura 3.6: Menú principal de la interfaz de control.

Las opciones disponibles en cada subpantalla corresponden a las funciones definidas en la sección del cliente y servidor HTTP. En la figura 3.7 se ve el ejemplo de como se observa una pestaña al seleccionar el dispositivo parlante, pero los botones son proporcionales a la pantalla, de ahí que si hay 3 botones entonces los botones serán mucho mas grandes que si fueran 5. Para acciones que requieren ingresar un valor numérico, como el volumen del parlante, la frecuencia del buzzer pasivo, el nivel de iluminación del dimmer (0-100) o el ángulo del motor, se despliega un teclado numérico dentro de la interfaz. Este se observa en la figura 3.8. En este teclado se puede seleccionar números enteros y al haber ingresado el valor se da enter, o, en caso de querer devolverse, entonces se utiliza Volver.


Figura 3.7: Interfaz de control del parlante Bluetooth. Permite la conexión, desconexión y ajuste del volumen mediante botones grandes.

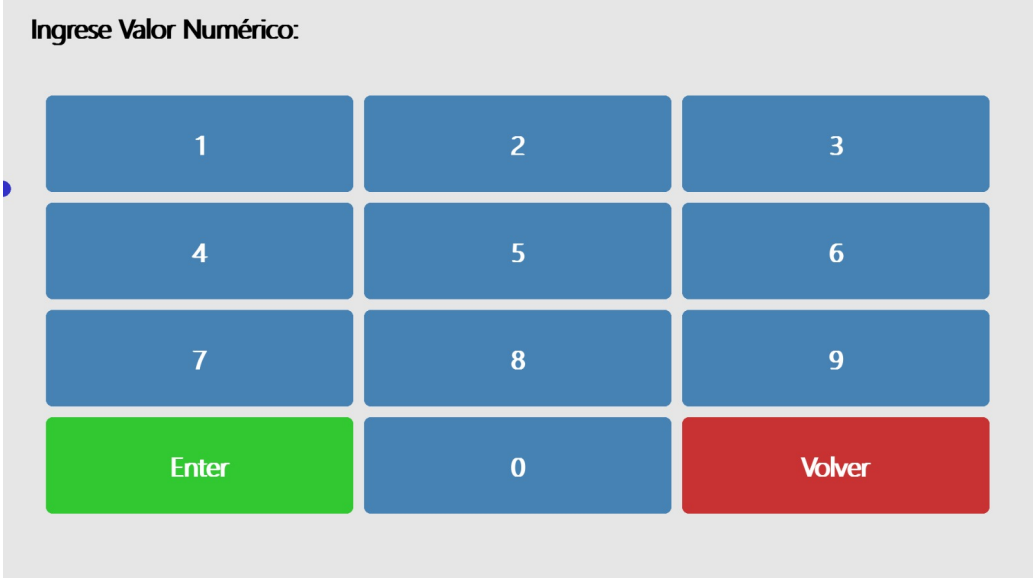


Figura 3.8: Interfaz de control numérico. Se utiliza para el ingreso de números, como en el caso del televisor o para ingresar direcciones IP en el módulo Wi-Fi.

Cada subpantalla incluye un botón para regresar al menú principal. En el menú principal, el botón de salida del programa incluye una ventana de confirmación para prevenir cierres accidentales, considerando que el sistema está diseñado para usuarios con movilidad limitada. En futuras versiones, se podría configurar para que solo el cuidador pueda cerrar el programa y evitar que el usuario con movilidad limitada lo cierre por accidente.

### Integración con el software de seguimiento ocular

Al iniciar el programa, se ejecuta automáticamente el software de seguimiento ocular *Gaze Follower* [32]. Se inicializa automáticamente la calibración del Gaze Follower y luego la interfaz se despliegue en pantalla completa.

Todos los elementos visuales se adaptan proporcionalmente al tamaño de la pantalla, asegurando que los botones y elementos interactivos mantengan dimensiones adecuadas para la selección ocular.

#### Asistencia de puntería y cursor auxiliar

Para ayudar a mitigar los errores en las coordenadas del software de seguimiento ocular y facilitar la selección de los botones de la interfaz se realizaron las siguientes mejoras:

- Se implementó un **filtro exponencial** sobre las coordenadas del eye tracker para suavizar los movimientos del cursor auxiliar, con un `buffer` de 10 muestras y un factor de suavizado $\alpha = 0{,}3$. Este procesamiento reduce el ruido y los saltos bruscos en la posición del cursor, aunque no elimina completamente los errores de precisión del eye tracker.
- Para facilitar la interacción mediante seguimiento ocular se definió un **cursor auxiliar**, que representa la posición en la interfaz relacionada con la mirada del usuario. Este cursor no siempre coincide directamente con las coordenadas crudas del eye tracker.

  El comportamiento del cursor auxiliar se define de la siguiente manera:

  - Inicialmente, el cursor auxiliar sigue las coordenadas filtradas `x` y `y` enviadas por el eye tracker.
  - Cuando el cursor se aproxima a un botón, el sistema de asistencia de puntería ajusta su posición automáticamente, desplazándolo al centro del botón. Esto facilita la selección y reduce errores causados por la imprecisión de las coordenadas.
  - Si el eye tracker detecta un parpadeo que indica una selección, el cursor se bloquea temporalmente durante 2 segundos, de manera que no se actualiza con nuevas coordenadas hasta que finaliza este período. Esto evita cambios bruscos por movimientos involuntarios de la mirada.

  De esta forma, el cursor auxiliar actúa como un intermediario entre las coordenadas crudas del eye tracker y la interfaz, permitiendo seleccionar botones de manera más precisa y confiable. Su comportamiento dinámico asegura que la posición del cursor se ajuste solo cuando sea necesario para mejorar la interacción, sin perder correspondencia con la mirada del usuario.

#### Cumplimiento parcial de estándares de accesibilidad

Se aplicaron las recomendaciones más relevantes de las **Apple Human Interface Guidelines (HIG)** [4] y las **Web Content Accessibility Guidelines (WCAG 2.1)** [30]. Se garantizó que los elementos interactivos tuvieran un tamaño mínimo de 44x44 píxeles y que el contraste de color entre texto, imágenes y fondo alcanzara al menos 4.5:1. Esto facilita la interacción de usuarios con baja visión, daltonismo o movilidad limitada.

Aunque la interfaz no sigue estrictamente todos los lineamientos de Apple HIG o WCAG 2.1, se priorizó la usabilidad y accesibilidad, asegurando que los botones sean lo suficientemente grandes para minimizar errores de interacción y que la información visual sea clara y fácilmente identificable.

### 3.0.8. Módulo Cursor (Eye Tracker)

El objetivo de este módulo es permitir que el usuario seleccione dispositivos y acciones en la interfaz únicamente mediante la mirada, facilitando el control del sistema para personas con movilidad reducida. Este módulo actúa como un controlador de cursor basado en seguimiento ocular y detección de parpadeo en donde se interpretan como movimientos o selecciones en la interfaz, que posteriormente se traducen en comandos hacia la Raspberry Pi.

Inicialmente, se consideraba el uso de un sistema de seguimiento ocular como una *caja negra*, ejecutado en la tablet (Microsoft o Android), el cual debía proporcionar la posición de la mirada en coordenadas y eventos de selección sin requerir modificaciones internas.

Durante el desarrollo se optó por emplear el software *Gaze Follower* de Gancheng Zhu y colaboradores [32], un sistema de código abierto capaz de realizar seguimiento ocular en tiempo real utilizando cámaras web comunes. Su API en Python permite integrarlo directamente con `pygame` y facilita la calibración, el registro de datos y la detección de eventos de selección.

### Adaptaciones implementadas

Aunque *Gaze Follower* proporciona coordenadas de mirada en tiempo real, no incluye mecanismos nativos de selección. Para solventarlo, se implementó un módulo de *detección de parpadeo* que permite realizar selecciones mediante movimientos oculares.

Se utilizan las variables internas `left_openness` y `right_openness`, que representan el grado de apertura de cada ojo, con valores típicos entre 30 y 130. A partir de ellas se definen parámetros y variables auxiliares:

- `threshold_cerrado` = 40
- `threshold_abierto` = 80
- `blink_times`: contador de parpadeos
- `ultimo_abierto`: estado previo del ojo

Un parpadeo se detecta cuando ambos ojos descienden de un valor superior a 80 a uno inferior a 40 y luego vuelven a superar 80, registrando el estado previo mediante `ultimo_abierto`. Si se producen dos parpadeos consecutivos en un intervalo corto, el sistema lo interpreta como doble parpadeo, equivalente a una acción de confirmación o selección dentro de la interfaz.

### Proceso de calibración

Para un seguimiento ocular preciso es necesario realizar la calibración de manera correcta. Para esto usuario debe ubicarse a aproximadamente 50 cm de la cámara, con buena iluminación y sin reflejos. El software inicia con:

gaze_follower = GazeFollower() gaze_follower.preview()

La función `preview()` muestra la imagen capturada por la cámara, junto con los datos números que indican posición detectada de cada ojo, valores de apertura, orientación y muchos más.

Para iniciar la calibración, el usuario presiona la barra espaciadora, de ahí se finaliza la función de preview y luego se ejecuta:

gaze_follower.calibrate()

Durante unos 30 segundos, se muestran puntos en la pantalla que el usuario debe observar, permitiendo que el software ajuste el modelo de predicción de la mirada. Al finalizar, se comparan los puntos detectados y los calculados; si la diferencia es grande, se recomienda repetir la calibración.

### Integración con la interfaz

Una vez calibrado, el módulo emite en tiempo real las coordenadas `x` e `y` de la mirada. Estas se utilizan para mover un **cursor auxiliar** dentro de `pygame`. El cursor auxiliar actúa como intermediario entre las coordenadas crudas del eye tracker y la interfaz, garantizando precisión y usabilidad.

### 3.0.9. Conexiones

La figura 3.9 muestra las conexiones de todos los dispositivos controlados por la Raspberry Pi.

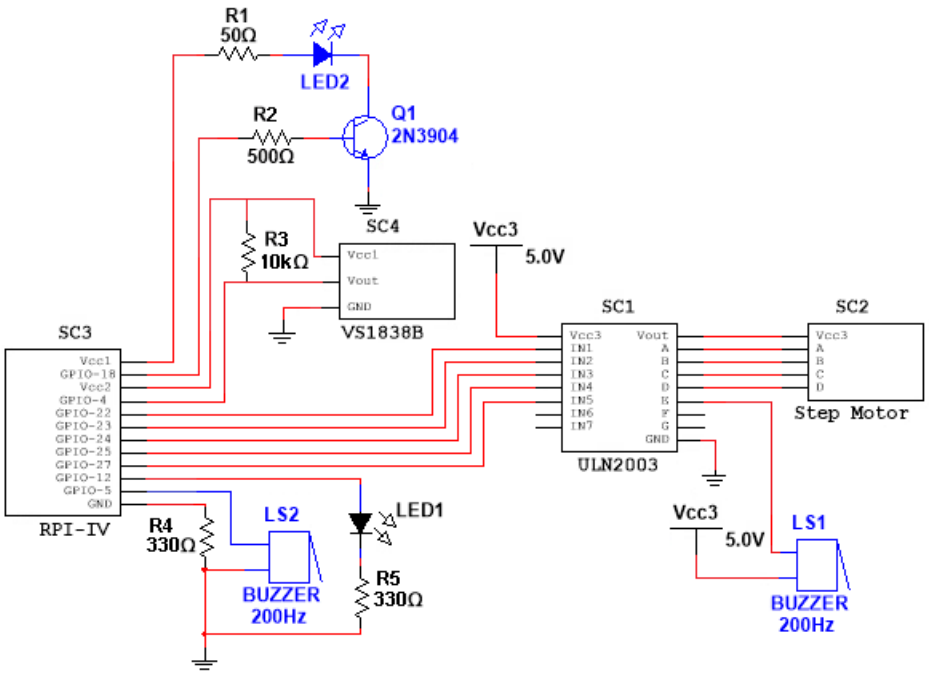


Figura 3.9: Diagrama de conexiones entre la Raspberry Pi, ULN2003, motor, LEDs y buzzer

# Capítulo 4

# Resultados Experimentales

Las pruebas experimentales se llevaron a cabo para verificar la funcionalidad, estabilidad y rendimiento de cada módulo del sistema de control, siguiendo un protocolo de $\approx 100$ ejecuciones por comando o dispositivo.

### 4.0.1. Módulo Infrarrojo

El módulo infrarrojo permitió la captura y reproducción de tramas de control.

#### Pruebas con Ventilador

Se realizaron aproximadamente 100 pruebas con los botones del ventilador, sin registrar errores en los patrones. Esto confirmó que el patrón infrarrojo capturado se enviaba consistentemente.

- **Configuración Física:** El LED IR se colocó a una distancia de aproximadamente 1 metro del receptor del ventilador *Basic Living*. Dado que el receptor está en la parte superior, el LED se orientó diagonalmente hacia abajo de manera lo más recta posible al receptor. Se buscó aprovechar la dirección en que la señal infrarroja es lo más fuerte (parte superior del LED), en donde este tipo de LED suele tener un alcance de alrededor de 6 metros.
- **Tramas de Control:** El control del ventilador utiliza un código propietario desconocido, con tramas que contienen aproximadamente 200 transiciones de datos de estado y nivel.
- **Transmisión:** El envío de la trama mediante chunks de datos no resultó en ninguna pérdida de información ni retardos que causen que el ventilador no interprete el comando de manera correcta.

#### Pruebas con Televisor

Se realizaron aproximadamente 100 pruebas con diferentes botones del televisor, el cual se encontró que utiliza el protocolo de NEC extendido.

- **Configuración Física:** El televisor se ubicaba en una sala con una mayor cantidad de luz natural, de ahí que para evitar interferencias se acercó el emisor a aproximadamente 10 cm, apuntando directamente al receptor del televisor, y se intentó tapar parte de la luz de las ventanas.
- **Comportamiento de Atajos:** Los los televisores implementan atajos para acciones cuando el botón se mantiene presionado, por ejemplo para subir bajar el volumen, entonces el control envía una trama completa al inicio y luego una trama más corta y repetitiva, indicando que el botón se mantiene presionado. El sistema desarrollado no replica este comportamiento, lo que hace que acciones como subir el volumen sean más lentas que con el control original.

- **Latencia:** No se observó **latencia perceptible** entre la interfaz y el control, ya que los patrones se envían en pocos milisegundos.

### 4.0.2. Módulo Bluetooth

El sistema se probó con un parlante *JBL Flip 4* a una distancia de aproximadamente 30-40 cm de la Raspberry Pi, enviando aproximadamente 100 comandos diferentes y utilizándolo por un tiempo prolongado.

- **Funcionalidad:** La conexión fue estable y el control de volumen operó correctamente.
- **Inconvenientes:**
    1. **Modo Reposo:** El parlante entraba en un estado de reposo si no se enviaba un comando dentro de un umbral de tiempo, requiriendo reiniciar el dispositivo para poder conectarse.
    2. **Exclusividad de Conexión:** Otros dispositivos conectados al parlante bloqueaban la conexión de la Raspberry Pi, por lo que para hacer las pruebas era necesario que ningún dispositivo estuviera conectado al parlante a excepción de la Raspberry pi.
- **Retardo de Conexión:** Se identificó un retardo (delay) de aproximadamente 5 o 10 segundos entre la recepción de la señal de control y su ejecución. Una vez conectado, el audio se reprodujo sin inestabilidad.

### 4.0.3. Módulo Stream

El stream se mantuvo por varias horas. La inestabilidad se relaciona principalmente con la conexión a internet.

- Retardo de Conexión: La función `stream(delay)` fue la que tomó más tiempo en ejecutarse (tiempo en que `VLC` se conecta al stream).
- Retardo de Audio: Existe un retardo constante de 5-10 segundos entre la señal de audio del dispositivo fuente y su reproducción en el parlante (latencia de red, decodificación y reproducción). Esto no afecta la escucha de música, pero sí causa desincronización en archivos de video, a menos que el video se observe directamente en la `Raspberry Pi`.

### 4.0.4. Módulo Wi-Fi

La comunicación Wi-Fi se evaluó con un dispositivo simulado (ESP32) y se mantuvo estable en las condiciones controladas.

- Pruebas: Se realizaron aproximadamente 100 pruebas de encendido y apagado del LED.
- Configuración Física: La Raspberry Pi y el ESP32 se ubicaron a 30-40 cm del router para asegurar una conexión directa y fuerte.
- Problema de IP Dinámica: El principal inconveniente inicial fue la dirección IP cambiante del ESP32, lo cual se solucionó configurando una IP estática.
- Latencia: La señal fue muy rápida, con la luz encendiéndose en menos de 1 segundo después de enviar el comando, sin latencia visible.
- Inestabilidad: Como es esperable con una aplicación que depende del wifi, depende mucho de la estabilidad del internet, el cual es un factor que depende del proveedor del internet.

### 4.0.5. Módulo por Cable

El control mediante pines `GPIO` y el driver `ULN2003` fue exitoso. Se realizaron 100 pruebas para cada dispositivo/comando. Se confirmó que el `ULN2003` no es compatible con señales PWM, por lo que el LED dimmer y el buzzer pasivo se conectaron directamente a la Raspberry Pi.

#### Dimmer LED

La función de test se ejecutó 100 veces, realizando transiciones de 0 % a 100 % del ciclo de trabajo de la señal PWM (duty cycle) con incrementos del 5 %.

- Ejecución: Al ser una conexión por cable, la ejecución fue directa y sin problemas.
- Percepción de Brillo: El brillo a diferentes valores se puede observar en las figuras 4.1, 4.2 y 4.3, por lo que es visible el cambio de brillo cambiando el valor del ciclo de trabajo. El brillo es poco visible para el ojo humano después de los 50 % pero al utilizar la cámara como se ve en las figuras 4.2 y 4.3 se puede observar una diferencia de brillo evidente.

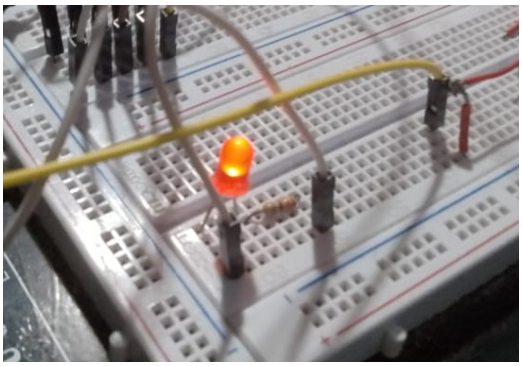

Figura 4.1: Brillo del LED cuando se tiene un ciclo de trabajo (*duty cycle*) del 1 %

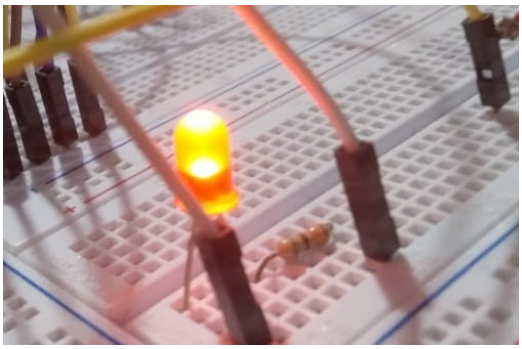

Figura 4.2: Brillo del LED cuando se tiene un ciclo de trabajo (*duty cycle*) del 50 %

#### Buzzer Activo

El buzzer activo se alimentó con un power bank de 9000-10000 mAh, que asegura una larga autonomía debido a su bajo consumo. Por lo que se puede mantener encendido por muchas horas y El sonido fue estable y no se registró ningún error despues de encender y apagar el buzzer unas 100 veces seguidas.

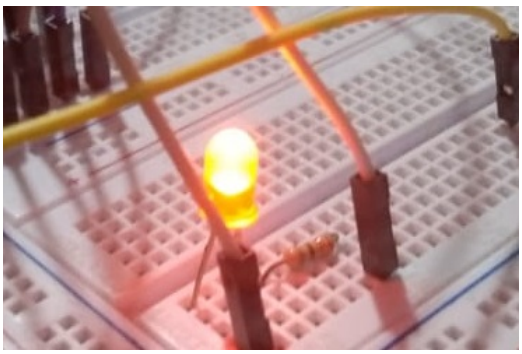

Figura 4.3: Brillo del LED cuando se tiene un ciclo de trabajo (*duty cycle*) del 100 %.

**Buzzer Pasivo**

El buzzer pasivo, conectado directamente a la `Raspberry Pi`, presentó una inestabilidad notable.

- Interferencia: Se observó un fenómeno de ruido eléctrico donde valores bajos de duty cycle del dimmer causaban un sonido débil en el buzzer pasivo (baja frecuencia). Este ruido se propaga a través del nodo de tierra en común, a pesar de tener los pines de voltaje de alimentación separados.
- Pruebas: Para simplificar la evaluación, el comando de test se ejecutó 100 veces, realizando incrementos de 200 Hz desde 200 Hz hasta 4000 Hz. Aunque el oido humano puede escuchar de 20 a 20 KHz, los sonidos de alarma deben de ser de frecuencias más agudas, por lo que estas frecuencias son más que suficientes para el propósito del buzzer.

**Motor**

El motor fue controlado a través del `ULN2003` mediante comandos de control para ángulos positivos y negativos (principalmente múltiplos de $45°$ y $-45°$).

- Visualización: Para hacer el movimiento más visible, se adhirió un papel con una marca a la patilla del motor.
- Funcionalidad: El motor funciona correctamente, el cual gira de manera lenta debido a su diseño. La función solo realiza incrementos/decrementos al ángulo, sin seguimiento de posición, lo que dificulta regresar a un estado inicial sin registrar cuanto se ha girado.

### 4.0.6. Módulo Servidor HTTP

El servidor `Flask` se ejecutó en la `Raspberry Pi`, implementando subprocesos para evitar bloqueos durante la ejecución de tareas de control de dispositivos.

- Pruebas sin *Eye Tracker*: Se realizaron 100 pruebas enviando comandos directamente con el cliente.
- Tasa de Éxito: El 95 % de las pruebas fueron exitosas.
- Causas de Fallo: Los fallos se concentraron en: cambio de IP del `ESP32` (resuelto con IP estática), inestabilidad de la conexión a internet (factor externo al dispositivo) o cuando el parlante Bluetooth entraba en modo de reposo o estaba conectado a otro dispositivo.

- Dependencia de la Red: La comunicación `HTTP` es dependiente de una conexión a internet estable.
- Retroalimentación: El servidor siempre devuelve retroalimentación al cliente indicando el estado de la solicitud.
- Pruebas con *Eye Tracker*: Se realizaron 100 pruebas con el eye tracker, y el módulo funcionó correctamente. Su uso no impuso una carga significativa de memoria o procesamiento a la computadora a la hora de ejecutar el programa.

### 4.0.7. Módulo Cliente HTTP

Se enviaron comandos 100 veces sin el eye tracker y 100 veces con él. Los problemas observados en el cliente fueron los mismos que se registraron en el servidor, ya que son dependientes de los mismos factores como el internet y la disponibilidad de los dispositivos.

### 4.0.8. Módulo Interfaz

La interfaz gráfica (`Pygame`) se diseñó con botones grandes y colores neutros para facilitar la lectura, la selección y reducir la fatiga visual.

- Problemas de Selección (Eye Tracker): Las selecciones con el eye tracker fueron complicadas.
  - Cuando se quería seleccionar los botones cercanos al borde de la pantalla era necesario mirar muy encima o muy por debajo del botón objetivo, forzando la vista y causando fatiga visual.
  - La inestabilidad del eye tracker podía causar descalibración, requiriendo mantener la cara lo más quieta posible y con menores movimientos posibles.
- Efectividad de Apuntado (100 Pruebas):
  - 60-70 selecciones lograron apuntar al botón después de varios intentos.
  - 20-30 selecciones lograron apuntar al botón al primer intento.
- Distribución de Dificultad: Los botones en el centro de la pantalla fueron los más fáciles de seleccionar. Los botones centrales superiores e inferiores requirieron forzar la vista y mirar $\approx 10$ unidades por encima. Los botones en las esquinas (superior-derecha, inferior-izquierda, etc.) fueron más sencillos de apuntar que los centrales extremos.

### 4.0.9. Módulo Cursor

Las pruebas revelaron que la efectividad del sistema es altamente dependiente de la calidad y estabilidad de la calibración del sistema, a pesar de que se añadieron formas de asistencia de puntería en la interfaz. El factor más relevante es la exactitud de los datos del eye tracker.

- Precisión Limitada: Las coordenadas originales tienen una precisión limitada, haciendo muy difícil apuntar sin la asistencia de puntería. Las variaciones ligeras entre el punto real y el predicho causan una gran inestabilidad del cursor.
- Calibración: Para la calibración se deben de tomar en cuenta las siguientes consideraciones:

- Setup: Para las pruebas, la cámara se situó a nivel de los ojos y la cara a 30-50 cm. Se utilizó una silla con soporte para evitar la inestabilidad. Como se observa en la figura 4.4

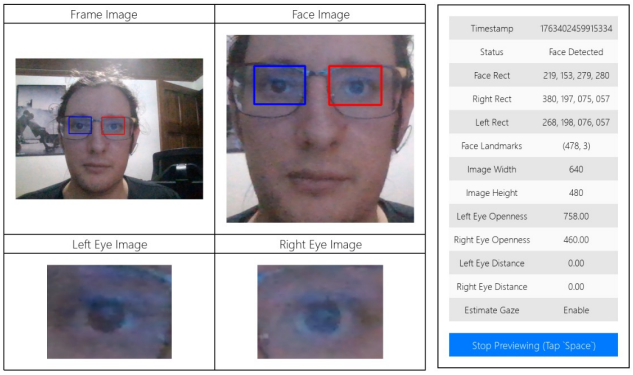


Figura 4.4: Pestaña de previsualización del GazeFollower

- Iluminación: Iluminación adecuada (luz blanca de 100 Watts) ubicada arriba y atrás en diagonal del usuario.
- Problemas Adicionales: El uso de lentes para las pruebas por parte del usuario y su brillo asociado fueron causas de errores, pero eran necesarios para hacer las pruebas.
- Resultado: En 100 pruebas con diferentes calibraciones, nunca se logró una calibración con errores de menores a 1 cm en todos los puntos. Los puntos centrales eran más precisos, pero los puntos en los borde presentaban errores altos de mayores de 2-3 cm. Esto se puede observar en la figura 4.5, en donde los puntos rojos puntos en la pantalla y los puntos verde son donde el programa de seguimiento ocular predice que el usuario está observando. Como se puede notar, hay diferencias claras entre los dos puntos.

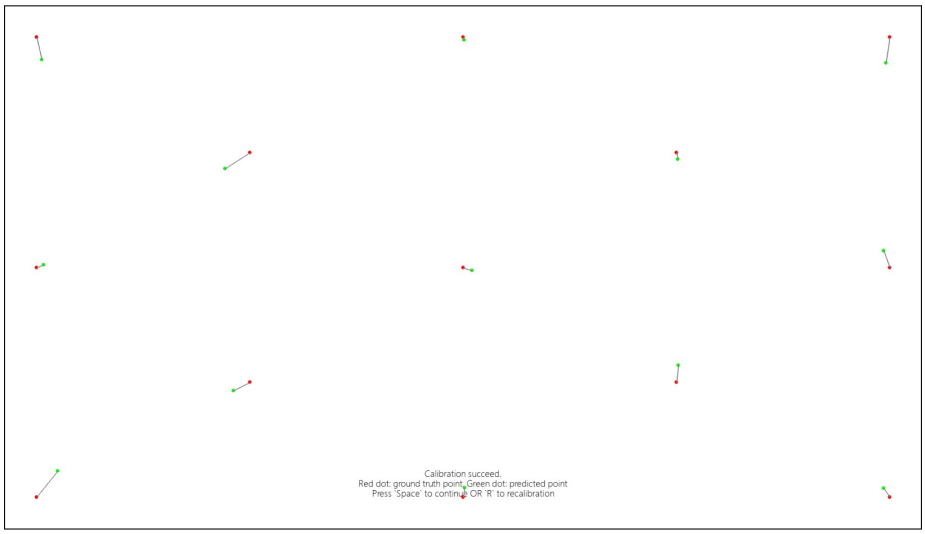


Figura 4.5: Calibración del seguidor ocular: en rojo los puntos reales y en verde los predichos.

- Inestabilidad: El cursor se pierde y la calibración se anula si la cámara pierde vista directa a los ojos, incluso en tiempos cortos. Por ejemplo, si se mueve la cabeza a la izquierda para observar algo, el sistema muchas veces perdía la calibración y había que detener

el programa y luego calibrarlo desde cero. Aunque personas con movilidad severamente limitada no pueden mover la cabeza, es posible que la calibración se pierda en caso de que tengan cama hospitalaria eléctrica.

- Detección de Parpadeo: La función de parpadeo, esencial para la selección, fue más sencilla de ejecutar.
  - Efectividad: 80 % de las selecciones por parpadeo fueron exitosas. El 20 % restante falló por no detectar el parpadeo o por detecciones erróneas de parpadeo.
  - Causas de Error: Es posible que el brillo de los lentes cause detecciones erróneas al hacer que la variable left_openness o right_openness caigan por debajo del umbral de cerrado y luego regrese al umbral de abierto.
- Variables Nativas: La interpretación de `left_openness` y `right_openness` (valores entre 30-130) resultó poco intuitiva para la alteración del código.

# Capítulo 5

# Análisis de resultados

El objetivo del proyecto es desarrollar un sistema de asistencia de bajo costo (menos de 500 USD), accesible y escalable. El costo total del prototipo de hardware se mantuvo significativamente por debajo de los 500 $. El análisis de resultados confirma que el objetivo de bajo costo se cumplió, si bien las limitaciones de precisión y escalabilidad observadas son un reflejo directo de esta elección de hardware accesible. A continuación, se analizan los resultados de cada módulo, destacando fortalezas, limitaciones y aspectos de viabilidad práctica.

### 5.0.1. Módulo Infrarrojo

El control de dispositivos con esta tecnología presenta desafíos significativos para usuarios con movilidad limitada y cuidadores, debido a la complejidad técnica de la captura y reproducción de señales. causada por los siguientes factores:

#### Variabilidad de las señales

Los dispositivos infrarrojos pueden operar con distintas frecuencias portadoras y protocolos propietarios. En las pruebas, tanto el ventilador como el televisor utilizaron una frecuencia de 38 kHz, compatible con el sistema actual, sin embargo, si se presentaran frecuencias diferentes (36 kHz o 40 kHz.), sería necesario cambiar el LED transmisor y el receptor, y en casos complejos incluso emplear un osciloscopio para analizar y capturar los patrones de la señal modulada. Algunos controles repiten los patrones, generan variaciones de botones o utilizan bits de dirección específicos, lo que exige capturar la señal correctamente y asociar cada comando con su respectivo dispositivo. Esto incrementa la dificultad de uso y configuración, especialmente para usuarios sin conocimientos técnicos, pero crear un receptor capaz de manejar todos los protocolos de comunicación infrarroja existentes y que sea altamente configurable requeriría probablemente un proyecto de mucho más tiempo y con mayores recursos de hardware y software.

#### Configuración del cliente

Actualmente, el código se diseñó mapeando manualmente los botones de cada dispositivo, creando archivos de texto con los patrones de control correspondientes y el cliente para controlar los dispositivos o se mapeó manualmente como subcomandos, lo que significa que es muy poco configurable. ya que un ventilador o televisor diferente podría requerir un mapeo de botones y subcomandos distintos.

Para mejorar la accesibilidad y escalabilidad, sería conveniente desarrollar un cliente más configurable, que permita al usuario o cuidador definir botones útiles o incorporar librerías predefinidas de controles de distintos dispositivos. Sin embargo,esto necesitaría contaar con más tiempo de desarrollo, por lo cual no se realizó en este proyecto.

#### Limitaciones físicas y ambientales

Algunos factores externos pueden afectar la operación del infrarrojo:

- Luz ambiental intensa que interfiera con la señal. Este problema puede mitigarse acercando el LED al dispositivo o utilizando cubiertas que concentren la señal.
- Alcance limitado del LED (aproximadamente 6 m). Para distancias mayores se podrían usar espejos o sistemas de guía óptica.
- Visibilidad del LED: la luz infrarroja no es perceptible al ojo humano, por lo que puede ser complicado saber si el sistema funciona o no, pero la cámara de los teléfonos móvil puede permitir observar la luz infrarroja si el LED funciona, si se mantiene la salida en alto.

#### Consideraciones técnicas

El `VS1838` invierte la señal de salida: entrega un nivel bajo cuando recibe luz infrarroja y un nivel alto en reposo. Esta característica requiere invertir los patrones capturados para una correcta reproducción.

El sistema permite configurar la frecuencia portadora en el software, facilitando la adaptación a distintos LEDs, aunque el cambio físico de hardware sigue siendo necesario si la frecuencia del dispositivo no coincide. Para cada nuevo dispositivo se deben capturar los patrones de cada botón varias veces, promediarlos y almacenarlos en archivos de texto. Sería ideal que esto se haga automáticamente con el cliente.

#### Viabilidad práctica

A pesar de estas dificultades, se comprobó que el control de dispositivos infrarrojos es posible. El ventilador y televisor se controló correctamente. La configuración a diferentes dispositivos requiere conocimientos técnicos, lo que limita la accesibilidad del prototipo para usuarios finales sin asistencia. Tal como se indicó anteriormente, crear una librería extensa con múltiples patrones que se puedan tener disponibles, tener múltiplos emisores o receptores que emitan o reciban señales a diferente longitudes y frecuencia de onda tomaría más tiempo y recursos de los que se tienen disponibles.

### 5.0.2. Módulo Bluetooth

Aunque la tecnología permite la comunicación inalámbrica entre dispositivos, existen limitaciones prácticas significativas que afectan la accesibilidad y la escalabilidad del sistema.

#### Dependencia de conexión exclusiva

Muchos dispositivos Bluetooth comerciales no permiten ser controlados simultáneamente por más de un cliente. Por ejemplo, un parlante conectado a una tablet no puede ser controlado desde la Raspberry Pi mientras mantiene la conexión activa con la tableta. Esto obliga a implementar soluciones alternativas, como transmitir audio mediante RTMP y reproducirlo con VLC en la Raspberry Pi, lo que añade complejidad al sistema. Lo ideal es utilizar un parlante que pueda ser controlado con la Raspberry sin importar cuantos dispositivos estén conectados al parlante Bluetooth.

#### Dirección MAC y control de dispositivos

Cada dispositivo Bluetooth tiene una dirección MAC única, que debe ser considerada en el software para establecer la comunicación. El cliente actual está configurado para funcionar únicamente con direcciones MAC previamente definidas. Para ampliar la compatibilidad, sería conveniente:

- Permitir la detección automática de nuevas direcciones MAC desde la interfaz para que se guarden automáticamente.

### Variabilidad de controles

Los controles Bluetooth pueden variar ampliamente entre dispositivos. En el prototipo actual, solo se implementaron funciones básicas: conectar, desconectar y cambiar volumen. Para soportar otras funciones sería necesario:

- Extender el cliente y servidor para incluir nuevos comandos.
- Adaptar la interfaz para que permita configurar subcomandos según las características de cada dispositivo.

### Limitaciones de transmisión de audio

Cuando se utiliza la `Raspberry Pi` para recibir audio de un dispositivo, se presentan posibles retrasos o desincronización, especialmente si el contenido es audiovisual. Esto se debe al retardo introducido por la transmisión RTMP. Las pruebas evidenciaron un retardo constante de 5 a 10 segundos en las funciones de conexión/desconexión y durante el streaming de audio.

Otra limitación es que en diferentes sistemas operativos se pueden dar restricciones a la hora de transmitir el audio. En Android o en iOS, el audio del sistema solo se puede controlar si se tiene un control de administrador, de ahí que se hace una dependencia a aplicaciones ya aprobadas por IOS o Android. Estas tienen que poder transmitir por internet en una forma en que la Raspberry Pi en Linux pueda recibir los datos. Por esta, se utilizó VLC, el cual se puede instalar en Windows y Linux, y permite conectarse y transmitir datos por stream.

### Inestabilidad Operacional

El modo de auto-reposo del parlante JBL 4 obliga a la presión de botones para salirse del modo reposo si no hay comandos en un umbral de tiempo, lo que limita el diseño actual y obliga a la intervención del cuidador, pero se puede corregir fácilmente si el parlante siempre esta conectado y escuchando a la Raspberry Pi. Esto podría consumir más energía, pero permite contar con un sistema más práctico para personas con movilidad limitada.

### Viabilidad práctica

A pesar de las limitaciones, el sistema permite establecer conexión y ejecutar funciones básicas de control Bluetooth. La accesibilidad para usuarios finales depende de que los dispositivos sean compatibles y permitan control desde la `Raspberry Pi`. La creación de un cliente más configurable y una interfaz que permita añadir dispositivos y comandos facilitaría la escalabilidad y el uso para personas con movilidad limitada.

El análisis evidencia que, para lograr un control completo de dispositivos Bluetooth de manera abierta y accesible, se requiere:

- Seleccionar dispositivos que permitan control concurrente o remoto.
- Diseñar software que gestione direcciones MAC dinámicamente o utilizar las herramientas nativas del Raspberry Pi, esto requeriría una conexión inicial de parte del cuidador de todos los dispositivos Bluetooth que se quieran controlar.
- Prever posibles limitaciones de transmisión de audio y video, especialmente en entornos con dispositivos Android o iOS.

### 5.0.3. Módulo Wi-Fi

El control de dispositivos mediante esta tecnología enfrenta limitaciones significativas, principalmente debido a aplicaciones propietarias y restricciones de seguridad implementadas por los fabricantes.

#### Dependencia de software propietario

Muchos dispositivos Wi-Fi comerciales solo pueden ser controlados mediante aplicaciones específicas de sus fabricantes. Esto implica que:

- No es posible acceder directamente al dispositivo desde la `Raspberry Pi` sin ingeniería inversa o licencias.
- La obtención de claves o protocolos de comunicación suele ser compleja y, en algunos casos, ilegal según los términos de uso.
- El control directo del dispositivo requiere aplicaciones compatibles con Linux, lo que limita la escalabilidad del proyecto.

#### Solución mediante dispositivos propios

Para sortear estas limitaciones, se implementaron dispositivos Wi-Fi basados en `ESP32`, controlables mediante `HTTP`. Esto permitió:

- Un control completo y programable de los dispositivos sin depender de aplicaciones propietarias.
- La posibilidad de adaptar la seguridad de la comunicación según las necesidades del usuario o cuidador.

#### Consideraciones de seguridad

Aunque los dispositivos `ESP32` controlados por el proyecto funcionan correctamente, el análisis muestra que:

- Para diseñar dispositivos propios de Wi-Fi se se deben considerar medidas de cifrado y autenticación para tener una mayor seguridad informática.
- Añade un inconveniente ya que los cuidadores deben de tener conocimientos mínimos de seguridad informática para configurar contraseñas y protocolos de forma adecuada. Además, deben poder adquirir dispositivos que no sean propietarios y deben de poder diferenciar entre los que son seguros y no lo son.

#### Viabilidad práctica

A pesar de estas limitaciones, el control de dispositivos Wi-Fi mediante hardware propio demuestra la viabilidad de la solución. Para su escalabilidad y accesibilidad en un entorno real:

- Sería recomendable desarrollar una interfaz que permita añadir nuevos dispositivos Wi-Fi y configurar comandos sin modificar el código.
- La selección de dispositivos abiertos es crítica para garantizar que el sistema sea funcional.
- Es importante establecer procedimientos claros de configuración y seguridad para cuidadores.

### 5.0.4. Módulo Cable

Esta tecnología es la más estable y confiable para el control de dispositivos, debido a la ausencia de interferencias inalámbricas y la simplicidad en la transmisión de señales.

#### Control de actuadores

Los dispositivos conectados por cable, como LEDs, buzzers y motores, funcionaron según lo esperado:

- Los LEDs y buzzer pasivos permitieron un control preciso de intensidad y frecuencia. Para el LED, el control del ciclo de trabajo permitió simular un dimmer.
- Los buzzers respondieron correctamente a los comandos de encendido y apagado, sin presentar errores.
- Los motores, como el stepper, pudieron moverse controlando individualmente las bobinas. El código actual solo permite mover el motor a un ángulo específico, sin llevar un seguimiento del ángulo actual, lo que limita aplicaciones más complejas como simulaciones de camas reclinables.

#### Limitaciones técnicas

Algunas restricciones técnicas fueron identificadas:

- El driver ULN2003 no permitió controlar LEDs o buzzer pasivo mediante PWM, lo que requirió conexiones separadas para estos actuadores.
- La potencia disponible de la Raspberry Pi limita el uso directo del motor, por lo que fue necesario utilizar la fuente (power bank) y el ULN2003 para controlar el motor.
- Para motores de mayor potencia sería necesario utilizar fuentes externas y considerar medidas de seguridad adicionales, ya que la corriente y voltaje requeridos superarían los 5V proporcionados por la `Raspberry Pi`.

#### Viabilidad práctica

La solución demuestra que el control físico mediante cable utilizado es estable, seguro y adecuado para sistemas de asistencia accesibles y de bajo costo.

### 5.0.5. Módulo Servidor

El funcionamiento del servidor es confiable, aunque presenta limitaciones propias de su arquitectura y del entorno Linux en la `Raspberry Pi`.

#### Funcionamiento y arquitectura

El servidor corre en múltiples hilos (multi-threaded) para atender solicitudes concurrentes, lo que permite que varios clientes puedan enviar comandos sin bloquear la ejecución del sistema. Sin embargo:

- Las funciones que bloquean el flujo de ejecución, como comandos que esperan finalización de procesos externos (Bluetooth o streaming de audio), deben ser ejecutadas de manera asíncrona o mediante `subprocess` para evitar que `Flask` detenga la atención de nuevas solicitudes.

- La arquitectura cliente-servidor facilita el control remoto y la escalabilidad del sistema, pero requiere atención en el diseño del flujo de datos para evitar bloqueos y retrasos.

**Limitaciones y consideraciones**

- La configuración actual permite manejar comandos básicos de manera eficiente, pero cualquier ampliación de funciones requiere modificar el servidor lo que requeriría de conocimientos técnicos.
- El servidor depende de la conectividad eléctrica e internet, de ahí que sin estos servicios el sistema completo no funciona.

**Viabilidad práctica**

El servidor `HTTP` demuestra ser un medio confiable para la comunicación entre la `Raspberry Pi` y clientes externos siempre y cuando haya una conexión de internet estable, permitiendo la ejecución de comandos de control de forma estructurada y escalable. La separación de funciones en bibliotecas simplifica la extensión y mantenimiento del código.

### 5.0.6. Módulo Cliente

El módulo cliente muestra que su implementación cumple con las funciones de control previstas, aunque con ciertas limitaciones en configurabilidad y escalabilidad.

**Funcionamiento**

El cliente se encarga de enviar solicitudes `HTTP` al servidor, activando funciones específicas de los dispositivos conectados. El diseño modular permite:

- Configurar subcomandos de manera simple mediante la carga de mapas de botones y archivos de texto predefinidos.
- Mantener el código del servidor limpio, reduciendo la complejidad en la gestión de comandos.

**Limitaciones**

- El cliente está actualmente limitado a los comandos y subcomandos predefinidos, lo que restringe la compatibilidad con nuevos dispositivos o funciones.
- Para cada dispositivo adicional, es necesario crear o modificar archivos de configuración, lo que requiere conocimiento técnico del usuario o cuidador.
- La robustez del control depende de la correcta comunicación con el servidor, y de la estabilidad de la conexión eléctrica e internet.

**Viabilidad práctica**

A pesar de las limitaciones, el cliente `HTTP` permite controlar dispositivos de manera ordenada y estructurada, facilitando la interacción del usuario con el sistema. La modularidad del diseño y el uso de archivos de configuración simplifica la extensión futura y la adaptación a nuevos dispositivos.

### 5.0.7. Módulo Interfaz

La interfaz gráfica desarrollada para el control de los dispositivos conectados a la `Raspberry Pi` funcionó correctamente durante las pruebas. Todos los botones respondieron adecuadamente y enviaron las órdenes al cliente `HTTP` y al servidor.

Se observó que la selección mediante seguimiento ocular era funcional gracias al cursor auxiliar, el sistema de asistencia de puntería y el bloqueo tras parpadeo. Estas soluciones permitieron seleccionar los botones en alrededor de 80 de las pruebas, aunque en algunos casos fue necesario dirigir la mirada ligeramente por encima o por debajo del botón deseado debido a la imprecisión del eye tracker, lo que causa fatiga en el ojo por forzar la vista.

La interfaz se percibió cómoda para la vista: los botones grandes y los colores neutros facilitaron la legibilidad y redujeron la fatiga visual.

#### Viabilidad práctica

El sistema resultó viable para pruebas experimentales y tareas de selección de baja a media precisión. La interfaz es adecuada para usuarios con movilidad reducida, aunque la exactitud del eye tracker impone un límite para tareas que requieren alta precisión.

### 5.0.8. Módulo Cursor

Durante las pruebas experimentales se observaron las siguientes limitaciones:

- Baja precisión absoluta del cursor, dificultando apuntar a zonas pequeñas. Los puntos periféricos presentaron errores de 2 a 3 cm.
- Valores de apertura de ojos poco intuitivos (aproximadamente de 30 a 130 ) lo que hace que no sea tan fácil añadir funciones de parpadeo.
- La detección de parpadeos no siempre funcionaba correctamente (tasa de fallo 20 %).
- Inestabilidad Crítica: La calibración se anula inmediatamente ante interrupciones momentáneas de la cámara o movimientos bruscos del usuario, lo que obliga a tener que calibrar manualmente.
- Incluso tras calibración, los puntos predictivos y reales podían presentar errores considerables con la configuración actual.

#### Impacto de Factores Externos

La sensibilidad a factores externos es alta. Las pruebas evidenciaron que el brillo de los lentes (gafas) del usuario introduce ruido, afectando las variables de openness y causando detecciones erróneas de parpadeo. Es necesario mantener la cabeza extremadamente quieta para mantener la estabilidad del cursor.

#### Viabilidad práctica

El eye tracker por sí solo tiene limitaciones severas para selección autónoma de elementos de la interfaz. Cumple su función después del procesamiento que realiza la interfaz, pero aún con este procesamiento a veces es complicado hacer las selecciones.

# Capítulo 6

# Conclusiones y Recomendaciones

### 6.0.1. Conclusiones

El sistema de control multiprotocolo basado en seguimiento ocular permitió controlar dispositivos mediante Wi-Fi, Bluetooth, infrarrojo y cableado usando una interfaz gráfica adaptada. Se logró controlar correctamente los siete dispositivos implementados, ejecutando comandos con éxito y verificando la respuesta física de cada uno. La interfaz, combinada con el cursor auxiliar, el filtro exponencial, la asistencia de puntería y el bloqueo tras parpadeo, permitió realizar selecciones funcionales, compensando en gran medida las limitaciones de precisión del eye tracker.

La implementación del sistema demostró que es viable construir un controlador multiprotocolo mediante seguimiento ocular con un presupuesto inferior a 500 USD. Las limitaciones de precisión y estabilidad del cursor provienen del software de seguimiento ocular (gaze follower) el cual fue desarrollado principalmente como herramienta de recolección de datos de mirada en estudios de psicología y ciencias cognitivas y no para tareas de interacción directa y de alta precisión con interfaces. Sin embargo, mediante las soluciones de software (cursor auxiliar y asistencia de puntería) es posible adaptar esta herramienta para su uso como un cursor para tareas de control simple.

Se concluye que la precisión absoluta del eye tracker es limitada y sensible lo que hace que no sea confiable. El control por Bluetooth genera un retraso constante de 5 a 10 segundos debido al streaming, lo que dificulta el manejo de contenido en tiempo real como video, aunque escuchar música funciona correctamente. El control por Wi-Fi se ve afectado por aplicaciones propietarias y protocolos encriptados, lo que reduce la practicidad para controlar hardware directamente. El módulo de infrarrojo cumple con el control.

El sistema funciona para los dispositivos que se buscaban controlar pero no permite integrar nuevos dispositivos o protocolos sin ajustes técnicos. La configuración limitada reduce la flexibilidad y la accesibilidad para usuarios sin conocimientos técnicos. Los resultados muestran que un sistema de control multiprotocolo mediante seguimiento ocular es funcional, pero todavía necesita mejoras importantes para ser práctico y accesible en escenarios reales.

### 6.0.2. Recomendaciones

A partir de los hallazgos y limitaciones observadas, se proponen las siguientes recomendaciones para futuras mejoras:

- **Software de seguimiento ocular más avanzado:** Utilizar eye trackers con mayor precisión, estabilidad y calibración dinámica para reducir la dependencia de calibraciones frecuentes y mejorar la exactitud de selección.
- **Disposición Óptica del Usuario:** En el caso de usuarios con problemas de visión, se recomienda el uso de lentes de contacto en lugar de gafas. Esto elimina el ruido introducido por los reflejos y el brillo de los cristales, lo cual mejora la calibración del sistema.

- **Pruebas con usuarios finales:** Realizar estudios de usabilidad con personas con movilidad limitada o con síndrome de enclaustramiento para evaluar la efectividad y comodidad del sistema en entornos reales de asistencia. Realizar estudios con personas promedio para determinar cuán fácil es hacer la configuración inicial del sistema para una persona sin muchos conocimientos técnicos.

- **Ampliación librerías:** Incorporar más patrones de control o desarrollar un servidor más configurable que permita añadir nuevos comandos, similar a los controles universales, para aumentar la compatibilidad con los diferentes dispositivos que las personas con movilidad limitada pueden tener. Crear librearías para diferentes dispositivos de Bluetooth, infarrrojo o Wifi podría facilitar mucho el control de dispositivos domésticos.

- **Dispositivos de audio compatibles:** Priorizar parlantes con control directo y conexión abierta, evitando depender de la función de transmisión de audio.

- **Dispositivos Wi-Fi abiertos o control por software:** Emplear dispositivos Wi-Fi de código abierto o con protocolos accesibles; en caso de dispositivos encriptados o con aplicaciones propietarias, considerar desarrollar soluciones de control puramente basadas en software, en donde se controlan directamente las aplicaciones de estas empresas con el software de seguimiento ocular.

- **Control de Posición de Actuadores:** Implementar un seguimiento de ángulo para el motor paso a paso (módulo por cable). Esto permitiría mantener un estado angular conocido y ejecutar funciones más avanzadas, como para camas reclinables), incrementando la utilidad del módulo de control por cable.

- **Utilizar lenguajes optimizados para interfaces:** Como recomendación para el diseño, se sugiere adoptar un lenguaje de programación optimizado para la interactividad de bajo nivel con interfaces, como JavaScript. Esto permitiría mejorar la comunicación entre interfaces y disminuir la latencia del sistema.

# Bibliografía


[1] A. Cespedes Sandí, *Desarrollo de un sistema de control de bajo costo para la asistencia de personas con movilidad severamente limitada*, Trabajo final de graduación, Escuela de Electrónica, Tecnológico de Costa Rica, Primer semestre 2025.

[2] Abyz.me.uk, pigpio: A C library for the Raspberry Pi's GPIO. Disponible en: `https://abyz.me.uk/rpi/pigpio/`. [Accedido: Nov. 2025].

[3] Abyz.me.uk, pigpiod - daemon. Disponible en: `https://abyz.me.uk/rpi/pigpio/pigpiod.html`. [Accedido: Nov. 2025].

[4] Apple Inc., *Human Interface Guidelines*. [En linea]. Disponible en: `https://developer.apple.com/design/human-interface-guidelines`.

[5] Arch Linux Manuals, bluetoothctl (1) manual page. Disponible en: `https://man.archlinux.org/man/bluetoothctl.1`. [Accedido: Nov. 2025].

[6] BlueZ, The Linux Bluetooth Protocol Stack. Disponible en: `https://www.bluez.org/`. [Accedido: Nov. 2025].

[7] ETC2, VS1838: Infrared Remote Control Receiver Module. Datasheet, ETC2. Disponible en: `https://www.alldatasheet.com/html-pdf/1132466/ETC2/VS1838/110/1/VS1838.html`. [Accedido: Nov. 2025].

[8] IEEE 802.11 Working Group, *IEEE Standard for Information Technology—Telecommunications and Information Exchange Between Systems—Local and Metropolitan Area Networks—Specific Requirements—Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE Std 802.11-2020, approved Dec. 3, 2020. [En linea]. Disponible en: `https://standards.ieee.org/ieee/802.11/7028/`.

[9] IEEE 802.15 Working Group, *IEEE Standard for Telecommunications and Information Exchange Between Systems—Local and Metropolitan Area Networks—Specific Requirements—Part 15: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Wireless Personal Area Networks (WPANs)*, IEEE Std 802.15.1-2002, approved Apr. 15, 2002. [En linea]. Disponible en: `https://standards.ieee.org/ieee/802.15.1/1180/`.

[10] J. Reinoso Guzmán, *Receptor con PIC para mandos infrarrojos tipo NEC*, Electrónica y Ciencia, May 7, 2010. [En linea]. Disponible en: `https://www.electronicayciencia.com/2010/05/receptor-con-pic-para-mandos.html`.

[11] John Uke, Aim Assist (Asistencia de puntería). [Artículo de GameTree]. Publicado: 2024-12-19. Disponible en: `https://gametree.me/gaming-terms/aim-assist/`. [Accedido: Nov. 2025].

[12] K. Maiese, M. C. Levin, *Síndrome de enclaustramiento (Locked-In Syndrome)*, Manuales MSD versión para profesionales. [En linea]. Disponible en: https://www.msdmanuals.com/es/professional/trastornos-neurolÃşgicos/coma-y-deterioro-de-la-consciencia/sÃŋndrome-de-enclaustramiento-locked-in-syndrome.

[13] Mustahsin Zarif, Procesamiento de señales: Filtro de media móvil exponencial (EMA). [Blog de Digikey]. Publicado: 2025-06-25. Disponible en: https://www.digikey.com.mx/es/blog/signal-processing-exponentially-moving-average-filter. [Accedido: Nov. 2025].

[14] ON Semiconductor, 2N3904: NPN General Purpose Amplifier. Datasheet, ON Semiconductor. Disponible en: https://www.alldatasheet.com/datasheet-pdf/pdf/11470/ONSEMI/2N3904.html. [Accedido: Nov. 2025].

[15] Page Laubheimer, Dragging and Dropping: A Common UI Pattern. [Artículo de NN Group]. Publicado: 2020-02-23. Disponible en: https://www.nngroup.com/articles/drag-drop/. [Accedido: Nov. 2025].

[16] Pallets, Flask Documentation. Disponible en: https://flask.palletsprojects.com/en/stable/. [Accedido: Nov. 2025].

[17] Pygame, *Noticias de Pygame*. [En linea]. Disponible en: https://www.pygame.org/news.

[18] Pygame, Pygame Documentation. Disponible en: https://www.pygame.org/docs/. [Accedido: Nov. 2025].

[19] Python Software Foundation, Requests: HTTP for Humans. [Paquete PyPI]. Disponible en: https://pypi.org/project/requests/. [Accedido: Nov. 2025].

[20] Pygame, Pygame Documentation. Disponible en: https://www.pygame.org/docs/. [Accedido: Nov. 2025].

[21] Python Software Foundation, Módulo subprocess. [Documentación de Python 3]. Disponible en: https://docs.python.org/es/3/library/subprocess.html. [Accedido: Nov. 2025].

[22] Python Software Foundation, Módulo time. [Documentación de Python 3]. Disponible en: https://docs.python.org/3/library/time.html. [Accedido: Nov. 2025].

[23] R. T. Fielding and J. Reschke, *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*, RFC 7231, June 2014. [En linea]. Disponible en: https://datatracker.ietf.org/doc/html/rfc7231.

[24] Raspberry Pi Foundation, Raspberry Pi Documentation: Getting Started. [En linea]. Disponible en: https://www.raspberrypi.com/documentation/computers/getting-started.html. [Accedido: Nov. 2025].

[25] RPi.GPIO, RPi.GPIO: A module to control the GPIO on a Raspberry Pi. [Paquete PyPI]. Disponible en: https://pypi.org/project/RPi.GPIO/. [Accedido: Nov. 2025].

[26] Texas Instruments, ULN2003A/ULN2003D/ULN2003APW/ULN2003AFW: High-Voltage, High-Current Darlington Transistor Arrays. Datasheet. Disponible en: https://www.alldatasheet.com/datasheet-pdf/pdf/29520/TI/ULN2003A.html. [Accedido: Nov. 2025].

[27] Tobii Dynavox, *TD I-Series*.[En linea]. Disponible en: https://us.tobiidynavox.com/pages/td-i-series.

[28] Tobii Gaming, *Tobii Eye Tracker 5*. [En linea]. Disponible en: https://gaming.tobii.com/product/eye-tracker-5/.

[29] VideoLAN, The VideoLAN Project. Disponible en: https://www.videolan.org/. [Accedido: Nov. 2025].

[30] World Wide Web Consortium (W3C), *Directrices de Accesibilidad para el Contenido Web (WCAG) 2.1*, W3C Recommendation, Jun. 5, 2018. [En linea]. Disponible en: https://www.w3.org/TR/WCAG21/

[31] Yuan Sheng, YSL-R531FR1C-F1: Infrared LED. Datasheet, SparkFun Electronics. Disponible en: https://cdn.sparkfun.com/datasheets/Components/LED/YSL-R531FR1C-F1.pdf. [Accedido: Nov. 2025].

[32] Zhu, G. gaze-follower. [Repositorio de GitHub]. Disponible en: https://github.com/ganchengzhu/gazefollower. [Accedido: Nov. 2025].

# Anexo A

### 6.0.3. Supervisión del Proyecto

El presente anteproyecto para el trabajo final de graduación está siendo supervisado y acompañado por la profesora Dra. Paola Vega Castillo, docente de la Escuela de Ingeniería Electrónica del Instituto Tecnológico de Costa Rica (TEC). La Dra. Paola ha brindado apoyo técnico, orientación metodológica y seguimiento continuo durante la planificación y desarrollo del sistema propuesto. Su experiencia en proyectos de integración tecnológica y su conocimiento en diseño electrónico han sido clave para validar la factibilidad y pertinencia de la solución desarrollada. Datos de contacto de la profesora supervisora: Nombre: Paola Vega Castillo

Correo electrónico: pvega@itcr.ac.cr

Departamento: Escuela de Ingeniería Electrónica, Instituto Tecnológico de Costa Rica

Se agradece profundamente su colaboración, la cual ha sido fundamental para el progreso y solidez del proyecto.

# Anexo B

### 6.0.4. Código fuente

#### Codigo servidor

Él código del servidor en servidor.py es:

```
from flask import Flask
from .Bluetooth.bluetooth_commands import dic_blue_commands
from .WiFi.led_control_wifi import control_led_wifi
from .Cable.control_ULN import control_cable
from .Infrared.Control_LEDs.controlled import cargar_botones,
   control_led
connect_blue=False
botones_map=cargar_botones()
app= Flask(__name__)

comandos_validos=[
        "inf_tel",
        "inf_vent",
        "blue_par",
        "cab_buz",
        "cab_mot",
        "cab_led",
        "wifi_luz",
        "comando8",
        "comando9",
        "comando10"
        ]
@app.route("/<comando>/<subcomando>",methods=["GET"])
def manejar_comando(comando,subcomando):
        global connect_blue
        if comando in comandos_validos:
                print(f"Comando recibido:{comando}",flush=True)
                if comando == "inf_tel":
                        print(f"subcomando recibido:{subcomando}",flush
                           =True)
                        control_led("tel",subcomando,botones_map)

                elif comando == "inf_vent":
                        print(f"subcomando recibido:{subcomando}",flush
                           =True)
                        control_led("vent",subcomando,botones_map)
                elif comando == "blue_par":
                        print(f"subcomando recibido:{subcomando}",flush
                           =True)
```

```
36                          if subcomando=="connect":
37                                  if connect_blue==False:
38                                          connect_blue=dic_blue_commands(
                                               subcomando,connect_blue)
39
40                          else:
41                                  if connect_blue==True:
42                                          connect_blue=dic_blue_commands(
                                               subcomando,connect_blue)
43
44
45
46                  elif comando == "cab_buz":
47                          print(f"subcomando recibido: {subcomando}",
                               flush=True)
48                          if subcomando=="act_on":
49                                  control_cable(3,True,0)
50                          elif subcomando=="act_off":
51                                  control_cable(3,False,0)
52                          elif subcomando=="pas_test":
53                                  control_cable(4,True,"test")
54                          elif subcomando=="pas_off":
55                                  control_cable(4,False,0)
56                          else:
57                                  freq=int(subcomando)
58                                  control_cable(4,True,freq)
59                  elif comando == "cab_mot":
60                          print(f"subcomando recibido: {subcomando}",
                               flush=True)
61                          if subcomando=="mot_off":
62                                  control_cable(1,False,0)
63                          else:
64                                  ang=int(subcomando)
65                                  control_cable(1, True, ang)
66                  elif comando == "cab_led":
67                          print(f"subcomando recibido:{subcomando}",flush
                               =True)
68                          if subcomando=="led_off":
69                                  control_cable(2,False,0)
70                          elif subcomando=="led_test":
71                                  control_cable(2,True,"test")
72                          else:
73                                  control_cable(2,True,subcomando)
74                  elif comando == "wifi_luz":
75                          print(f"Subcomando recibido:{subcomando}",flush
                               =True)
76                          control_led_wifi(subcomando)
77
78                  else:
79                          return f"Comando '{comando} recibido
                               correctamente."
80                  return f"Comando '{comando}' recibido correctamente."
81
```

```
        else:
                print(f"Comando inv lido: {comando}",flush=True)
                return f"Comando '{comando}' no reconocido.",404
if __name__== "__main__":
        app.run(host="0.0.0.0",port=8000,threaded=False)
```

### Módulo infrarrojo

El código de python del emisor es:

```
import pigpio
import time

GPIO_TX=18
GPIO_RX=4
pi=pigpio.pi()
duty=50/100
if not pi.connected:
        exit()
BASE_DIR="/home/pi/Proyecto/Infrared/patrones_inf"
ARCHIVOS={
1:"comando_botones_tel_promedio.txt",
2:"comando_botones_vent_promedio.txt"}
freq=38000
pi.set_mode(GPIO_RX,pigpio.INPUT)
pi.set_mode(GPIO_TX,pigpio.OUTPUT)


def cargar_botones():
        path_tel=f"{BASE_DIR}/{ARCHIVOS[1]}"
        path_vent=f"{BASE_DIR}/{ARCHIVOS[2]}"
        botones_tel={}
        botones_vent={}
        with open(path_tel,"r") as f:
                for linea in f:
                        if ":" in linea:
                                boton_tel, patron_tel=linea.strip().
                                   split(":",1)
                                botones_tel[boton_tel.strip()]=
                                   patron_tel.strip()
        with open(path_vent,"r") as g:
                for linea in g:
                        if ":" in linea:
                                boton_vent,patron_vent=linea.strip().
                                   split(":",1)
                                botones_vent[boton_vent.strip()]=
                                   patron_vent.strip()
        botones_map={"tel": botones_tel,
                "vent": botones_vent}
        return botones_map
def parsear_patron(patron_str):
        patron_str=patron_str.strip()
        patron_str=patron_str.strip("[]")
```

```
        partes=patron_str.split("),")
        patron=[]
        for p in partes:
                p=p.strip().replace("(","").replace(")","")
                if not p:
                        continue
                nums=p.split(",")
                if len(nums)==2:
                        estado=int(nums[0].strip())
                        duracion=int(nums[1].strip())
                        patron.append((estado,duracion))
        return patron

def crear_pulsos_modulados(patron,carrier_freq):
        wf=[]
        T=int(1000000//carrier_freq)
        t_on=int(T//2)
        t_off=T-t_on
        for estado, duracion in patron:
                if estado==1:
                        num_ciclos=max(1,int(duracion)//T)
                        for _ in range(num_ciclos):
                                wf.append(pigpio.pulse(1<<GPIO_TX,0,
                                   t_on))
                                wf.append(pigpio.pulse(0,1<<GPIO_TX,
                                   t_off))
                        resto=duracion -(num_ciclos * T)
                        if resto>0:
                                wf.append(pigpio.pulse(1<<GPIO_TX,0,int
                                   (resto//2)))
                                wf.append(pigpio.pulse(0,1<<GPIO_TX,int
                                   (resto-resto//2)))
                else:
                        wf.append(pigpio.pulse(0,1<<GPIO_TX,duracion))
        return wf
def enviar_patron(patron):
        wf_mod=crear_pulsos_modulados(patron,freq)
        chunk_size=1000
        for i in range(0,len(wf_mod),chunk_size):
                pi.wave_clear()
                chunk=wf_mod[i:i+chunk_size]
                pi.wave_add_generic(chunk)
                wid=pi.wave_create()
                if wid>=0:
                        pi.wave_send_once(wid)
                        while pi.wave_tx_busy():
                                time.sleep(0.001)
                        pi.wave_delete(wid)
def control_led(disp,boton,map_botones):
        botones=map_botones.get(disp)
        patron_str=botones.get(boton)
        patron_pars=parsear_patron(patron_str)
        enviar_patron(patron_pars)
```

El código de python de un ejemplo de receptor es:

```
import pigpio
import time
from collections import Counter
IR_PIN=4

pi = pigpio.pi()
if not pi.connected:
        exit()
last_tick= 0
last_level=0
pulses=[]
trama_capturada=[]
capturando=False

def cbf(gpio,level,tick):
        global last_tick,last_level,pulses,trama_capturada,capturando
        level=1-level #Esto se hace por que el receptor VS1838 esta en alto al no detectar se al y en bajo al detectar se al.
    #Por lo que hay que invertir los bits que detecta el GPIO
        if not capturando:
                return
        if last_tick==0:
                last_tick=tick
                last_level=level
                return
        delta = pigpio.tickDiff(last_tick,tick)
        last_tick = tick
        if level !=last_level:
                pulses.append((last_level,delta))
                last_level = level
        if delta > 10000 and pulses:
                trama_capturada=pulses
                pulses=[]
pi.set_mode(IR_PIN,pigpio.INPUT)
pi.set_pull_up_down(IR_PIN,pigpio.PUD_UP)
cb= pi.callback(IR_PIN, pigpio.EITHER_EDGE,cbf)

botones = {
        1: "timer",
        2: "mode",
        3: "osc",
        4: "off",
        5: "on/speed"
}

archivo="nombre_archivo_para_botones.txt"
with open(archivo,"w") as f:
        f.write("")

print("Registro de botones")


```

```
try:
        while True:
                boton_input=int(input( "Ingrese el n mero del boton a
                   estripar: \n1. TIMER \n2. MODE\n3. OSC\n4. OFF \n5.
                   ON/SPEED\n"))
                if boton_input not in botones:
                        print("Bot n inv lido. Intente de nuevo")
                        continue

                boton = botones[boton_input]
                capturando=True
                input(f"Apunte el control remoto y presione el boton {
                   boton} y presione enter cuando haya terminado...")
                #print("trama", trama_capturada)
                capturando=False
                if not trama_capturada:
                        print("No se detecto ninguna trama. Intente de
                           nuevo.")
                        continue
                print(f"{boton} : {trama_capturada}")
                with open(archivo, "a") as f:
                        f.write(f"{boton } :{trama_capturada}\n")
                print(f"Boton {boton} registado correctamente!\n")
                trama_capturada=[]
except KeyboardInterrupt:
        cb.cancel()
        pi.stop()
```

Este código esta hecho específicamente para guardar los patrones de los botones del ventilador, de ahí que al correr el programa abre una interfaz que le indica al usuario cual botón del ventilador debe de presionar. Si se tiene un dispositivo diferente se recomienda parametrizar el código. En donde se puede crear una lista con los comandos, y luego llamar uno por uno. Por lo que sería más sencillo si se quieren guardar patrones de infrarrojo para otros dispositivos.

El código para obtener el valor promedio de cada uno de los botones y luego guardarlo en un archivo de texto es:

```
archivo = "nombre_archivo_para_botones.txt"
archivo_salida="nombre_archivo_para_botones_promediado.txt"
UMBRAL_CORTE = 10000
botones={}
with open(archivo, "r") as f:
        for linea in f:
                if ":" not in linea:
                        continue
                boton,trama_str=linea.split(":",1)

                trama_str = trama_str.strip()

                trama_str = trama_str.replace("[","").replace("]","").
                   replace("(","").replace(")","")
                numeros=[x.strip() for x in trama_str.split(",") if x.
                   strip()]

                trama=[]
```

```
                for i in range(0,len(numeros)-1,2):
                        try:
                                bit=int(numeros[i])
                                tiempo=int(numeros[i+1])
                                trama.append((bit, tiempo))

                        except:
                                continue

                if boton not in botones:
                        botones[boton]=[]
                botones[boton].append(trama)
        tramas_promedio={}
        for boton, tramas in botones.items():

                if not tramas:
                        continue
                max_len=max(len(t) for t in tramas)
                print(max_len)
                tramas_validas=[t for t in tramas if len(t) ==max_len]
                if not tramas_validas:
                        continue
                promedio_con_bits = []
                for i in range(max_len):
                        bit=tramas_validas[0][i][0]
#                       print(tramas_validas)
                        sum_tiempos=sum(t[i][1] for t in tramas_validas
                           )
                        tiempo_prom = int(round(sum_tiempos/len(
                           tramas_validas)))
                        promedio_con_bits.append((bit,tiempo_prom))
                tramas_promedio[boton] = promedio_con_bits
        with open(archivo_salida,"w") as f:
                for boton, trama in tramas_promedio.items():
                        f.write(f"{boton}:{trama}\n")
        print(f"tramas promedios guardadas en '{archivo_salida}'")
```

Esto daría una salida en el archivo de texto varios botones con el siguiente formato si es NEC extendido:

```
on/off:[(1, 8972), (0, 4478), (1, 564), (0, 552), (1, 569), (0, 1675), (1, 564),
(0, 555), (1, 568), (0, 1678), (1, 561), (0, 555), (1, 569), (0, 1675), (1, 562),
(0, 1676), (1, 563), (0, 1677), (1, 562), (0, 1676), (1, 562), (0, 1676), (1, 563),
(0, 1678), (1, 560), (0, 557), (1, 565), (0, 560), (1, 566), (0, 557), (1, 563),
(0, 1677), (1, 562), (0, 1678), (1, 562),
(0, 1680), (1, 558), (0, 1678), (1, 561),
(0, 1678), (1, 561), (0, 560), (1, 563), (0, 1678), (1, 559), (0, 561), (1, 562),
(0, 562), (1, 561), (0, 1680), (1, 559), (0, 561), (1, 561), (0, 561), (1, 562),
(0, 562), (1, 560), (0, 1688), (1, 551), (0, 562), (1, 560),
(0, 1686), (1, 552), (0, 1690), (1, 549), (0, 559), (1, 560), (0, 37600)]
```

Pero, existen muchos formatos diferentes asi que no deben de seguir exactamente este formato. (El último dato de 0,37600 es cuando el programa termina de detectar, el tiempo de este dato no va a afectar el control)

### Módulo Bluetooth

El código para controlar el parlante Bluetooth es:

```
from .connect_bluet import connect_bluetooth
from .vol_bluet import vol_bluetooth
from ..Stream.connect_stream import stream

def dic_blue_commands(command,connect):
        if command=="connect":
                connect=connect_bluetooth(True)
                connect2=stream(True)
                if connect and connect2:

                        print("Parlante conectado.")

                else:

                        if connect:
                                connect=connect_bluetooth(False)
                        elif connect2:
                                connect2=stream(False)

                        print("Error de conexi n")
        elif command=="disconnect":
                connect=connect_bluetooth(False)
                connect2=stream(False)
                if connect==False and connect2==False:
                        print("Parlante desconectado.")

                else:
                        print("Error de desconexi n")

        elif command=="vol_up":
                if connect==True:
                        vol_bluetooth("+5%")
                else:
                        print("Error parlante no conectado:Conecte el
                           parlante primero")
        elif command=="vol_down":
                if connect==True:

                        vol_bluetooth("-5%")
                else:
                        print("Error parlante no conectado: Conecte el
                           parlante primero")
        else:
                if connect==True:
                        vol_bluetooth(command)
                else:
                        print("Error parlante no conectado:conecte el
                           parlante primero")
        return connect
```

El código para conectarse al parlante Bluetooth es:

```
import subprocess
import shlex

BLUETOOTH_MAC="04:FE:xx:xx:xx:xx" #Direccion MAC del parlante Bluetooth


def  connect_bluetooth(connect):
        try:

                if connect:
                        subprocess.run(
                                f"bluetoothctl connect {BLUETOOTH_MAC}"
                                   ,
                                shell=True,
                                check=True
                        )
                        subprocess.run(
                                f"pactl set-default-sink bluez_sink.{
                                   BLUETOOTH_MAC.replace(':','_')}.
                                   a2dp_sink",
                                shell=True,
                                check=True
                        )
                        return True
                else:
                        subprocess.run(
                                f"bluetoothctl disconnect {
                                   BLUETOOTH_MAC}",
                                shell=True,
                                check=True
                        )
                        return False
        except subprocess.CalledProcessError:
                if connect:
                        print("Error al conectar parlante")
                        return False
                else:
                        print("Error al desconectar parlante")
                        return True
```

El código para cambiar el volumen del parlante Bluetooth es:

```
import subprocess
import shlex
def vol_bluetooth(vol):
        if vol=="+5%":
                subprocess.run(shlex.split("pactl set-sink-volume
                   @DEFAULT_SINK@ +5%"))
        elif vol=="-5%":
                subprocess.run(shlex.split("pactl set-sink-volume
                   @DEFAULT_SINK@ -5%"))
        else:
                subprocess.run(shlex.split(f"pactl set-sink-volume
                   @DEFAULT_SINK@ {vol}%"))
```

### Módulo Stream

El código para conectarse al stream es:

```
import subprocess
import time

process=None
url="rtmp://127.x.x.x.x:1935/live/stream" #<stream> en este caso es la
    contrase a de la transmisi n

def stream(connect):
        global process
        if connect:
                try:

                        process = subprocess.Popen(["cvlc",url],
                                stdout=subprocess.PIPE,
                                stderr=subprocess.PIPE,
                                text=True
                        )
                        time.sleep(3)
                        if process.poll() is not None:
                                process = None
                                return False
                        return True
                except Exception:
                        process=None
                        return False
                else:
                        if process is not None:
                                process.terminate()
                                process=None
                                return False
                        else:
                                return True



        if connect == False:
                process.terminate()
                return False
        return True
```

### Módulo Wi-fi

El código para controlar el led Wifi es:

```
import requests

ESP32_IP="192.xxx.xxx.xxx" #Direcci n IP del ESP32
```

```
4
5  def control_led_wifi(control):
6          if control=="on":
7                  requests.get(f"http://{ESP32_IP}/led_on")
8          else:
9                  requests.get(f"http://{ESP32_IP}/led_off")
10 #control_led_wifi("on")
```

El código para emular una luz wifi utilizando el ESP32 en arduino en donde se utilizo la board ESP32 Dev Module en el IDE de Arduino es:

```
1  #include <WiFi.h>
2  #include <WebServer.h>
3
4  const char* ssid = "nombre_wifi";
5  const char* password = "contrase a_wifi";
6
7  WebServer server(80); // Servidor HTTP en el puerto 80
8  const int ledPin = 2; // Pin donde tienes el LED
9
10 // Configuraci n de IP fija
11 IPAddress local_IP(192,x,x,x);
12 IPAddress gateway(192,x,x,x);    // Tu router, revisa que sea correcto
13 IPAddress subnet(255,255,255,0);   // M scara de subred t pica
14
15 void setup() {
16   Serial.begin(115200);
17   pinMode(ledPin, OUTPUT);
18
19   // Configurar IP est tica antes de conectarse
20   if (!WiFi.config(local_IP, gateway, subnet)) {
21     Serial.println("Error al configurar IP est tica");
22   }
23
24   WiFi.begin(ssid, password);
25   Serial.print("Conectando al WiFi");
26   while (WiFi.status() != WL_CONNECTED) {
27     delay(500);
28     Serial.print(".");
29   }
30   Serial.println("\nWiFi conectado!");
31   Serial.print("IP fija: ");
32   Serial.println(WiFi.localIP());
33
34   // Definir rutas HTTP
35   server.on("/led_on", []() {
36     digitalWrite(ledPin, HIGH);
37     server.send(200, "text/plain", "LED ENCENDIDO");
38   });
39
40   server.on("/led_off", []() {
41     digitalWrite(ledPin, LOW);
42     server.send(200, "text/plain", "LED APAGADO");
43   });
```

```
44
45    server.begin();
46    Serial.println("Servidor listo");
47  }
48
49  void loop() {
50    server.handleClient(); // Manejar solicitudes entrantes
51  }
```

### Módulo cableado

El código para control de los componentes por cable es:

```
1   import RPi.GPIO as GPIO
2   import time
3
4   motor_pins=[22,23,24,25]
5   led_pin=12
6   buzzer_activo_pin=27
7   buzzer_pasivo_pin=5
8
9   GPIO.setmode(GPIO.BCM)
10  GPIO.setwarnings(False)
11
12  for pin in motor_pins + [buzzer_activo_pin]:
13          GPIO.setup(pin,GPIO.OUT)
14          GPIO.output(pin,GPIO.LOW)
15
16  GPIO.setup(buzzer_pasivo_pin,GPIO.OUT)
17  GPIO.setup(led_pin,GPIO.OUT)
18  led_pwm=GPIO.PWM(led_pin,1000)
19  led_pwm.start(0)
20  buzzer_pwm=GPIO.PWM(buzzer_pasivo_pin,2000)
21  buzzer_pwm.start(0)
22
23  def control_cable(device,on,dat):
24          if device==1:
25                  dat=int(dat)
26                  steps=int(abs(dat*512/360))
27                  if on:
28                          seq= [
29                          [1,0,0,0],
30                          [1,1,0,0],
31                          [0,1,0,0],
32                          [0,1,1,0],
33                          [0,0,1,1],
34                          [0,0,0,1],
35                          [1,0,0,1]
36                          ]
37                          if dat<0:
38                                  seq=list(reversed(seq))
39
40                          for _ in range(steps):
```

```
                                for pattern in seq:
                                        for pin, val in zip(motor_pins,
                                           pattern):
                                             GPIO.output(pin,val)
                                        time.sleep(0.002)
                else:
                        for pin in motor_pins:
                                GPIO.output(pin,GPIO.LOW)
        elif device==2:
                if on:
                        led_pwm.ChangeDutyCycle(0)
                        if dat=="test":
                                for d in range(0,101,5):
                                        print(d)
                                        led_pwm.ChangeDutyCycle(d)
                                        time.sleep(0.25)
                                led_pwm.ChangeDutyCycle(0)
                        else:
                                dat=int(dat)
                                led_pwm.ChangeDutyCycle(dat)
                else:
                        led_pwm.ChangeDutyCycle(0)
        elif device==3:
                if on:
                        GPIO.output(buzzer_activo_pin,GPIO.HIGH)
                else:
                        GPIO.output(buzzer_activo_pin,GPIO.LOW)
        elif device==4:
                if on:
                        if dat=="test":
                                buzzer_pwm.start(50)
                                for f in range(200,4001,200):
                                        buzzer_pwm.ChangeFrequency(f)
                                        time.sleep(0.5)
                                buzzer_pwm.start(0)
                        else:
                                dat=int(dat)
                                buzzer_pwm.start(50)
                                buzzer_pwm.ChangeFrequency(dat)


                else:
                        buzzer_pwm.ChangeDutyCycle(0)
```

### 6.0.5. Módulo Cliente

El código del cliente se tiene:

```
import requests

ip_servidor="192.xxx.x.xx"
puerto=8000

```

```
def enviar_request(comando,subcomando):
    url = f"http://{ip_servidor}:{puerto}/{comando}/{subcomando}"
    try:
        r = requests.get(url, timeout=10)
        print(f"Respuesta del servidor: {r.text}")
    except requests.exceptions.RequestException as e:
        print(f"No se pudo enviar el comando: {e}")
```

### 6.0.6. Módulo Interfaz y cursor

La inicialización del GazeFollower y la calibración del mismo y la interfaz se desarrolla con este código:

```
    import pygame
import sys
import time
from gazefollower import GazeFollower
from Cliente.cliente import enviar_request

#Configuracion del eye tracker
gaze_follower = GazeFollower()
gaze_follower.preview()
gaze_follower.calibrate()
pygame.display.quit()
pygame.quit()
gaze_follower.start_sampling()

#Configuracion de la interfaz
pygame.init()
pantalla = pygame.display.set_mode((0, 0), pygame.FULLSCREEN)
ANCHO, ALTO = pantalla.get_size()
pygame.display.set_caption("Interfaz de Control")

#Colores
BLANCO = (255, 255, 255)
NEGRO = (0, 0, 0)
AZUL = (70, 130, 180)
AZUL_OSCURO = (30, 60, 90)
ROJO = (200, 50, 50)
VERDE = (50, 200, 50)
GRIS = (230, 230, 230)
AZUL_CURSOR = (50, 50, 200)

fuente = pygame.font.SysFont("segoeui", 50)
fuente.set_bold(True)

#Cursor de prueba, ayuda a ver los cursores del eye tracker y el
   promedio para pruebas
TEST = True

# Dispositivos, que son los nombres de los botones.
comandos_amigables = {
```

```
    "inf_tel": "Televisor", "inf_vent": "Ventilador", "blue_par": "Parlante",
    "cab_buz": "Buzzer", "cab_mot": "Motor", "cab_led": "Dimmer",
    "wifi_luz": "Luz", "comando8": "Disp8", "comando9": "Disp9", "comando10": "Disp10",
}

#Subcomandos, que son el nonmbre de botones y el subcomando

subcomandos_amigables = {
    "inf_tel": {"on/off": "Encender/Apagar","return": "Volver","vol_down": "Volumen -","vol_up": "Volumen +",
                "left": "Izquierda","right": "Derecha","up": "Arriba","down": "Abajo","exit": "Salir TV"},
    "inf_vent": {"timer": "Temporizador","mode": "Modo","osc": "Oscilaci n","off": "Apagar ventilador","onspeed": "Encender/Velocidad"},
    "blue_par": {"connect": "Conectar parlante","+5%": "Subir volumen","-5%": "Bajar volumen",
                "vol": "Ajustar volumen espec fico","disconnect": "Desconectar parlante"},
    "cab_buz": {"act_on": "Buzzer activo ON","act_off": "Buzzer activo OFF","freq": "Frecuencia buzzer pasivo",
                "pas_test": "Prueba buzzer pasivo","pas_off": "Buzzer pasivo OFF"},
    "cab_mot": {"ang": " ngulo  del motor","mot_off": "Motor apagado"},
    "cab_led": {"duty": "Brillo del LED","led_test": "Prueba brillo LED","led_off": "LED apagado"},
    "wifi_luz": {"on": "Luz encendida","off": "Luz apagada"}
}

# Claves que envian dat a la funcion de control_uln. O sea se envian como int, por lo que se abre n teclado numerico para el control
SUBCOMANDOS_NUMERICOS = ["vol", "freq", "ang", "duty"]

# Ayuda para detecci n de parpadeos
blink_times = []
ultimo_abierto = True

#Bloqueo del puntero promedio cuando se parpadea
block_pointer_until = 0.0 # Tiempo hasta el cual el puntero debe permanecer bloqueado.

# Caonfiguracion del filtro exponencial para suavizado del gazefollowwer
gaze_buffer = []
BUFFER_SIZE = 10
ALPHA = 0.3
ALPHA_POINTER = 0.15
pointer_x, pointer_y = ANCHO // 2, ALTO // 2

#Umbral en donde se considera que se cerraron los ojos, ayuda a bloquear el puntero promedio
```

```
77  THRESHOLD_CERRADO_BLOQUEO = 60
78
79  #Seleccion por doble parpadeo
80  def detectar_doble_parpadeo(info, threshold_cerrado=40,
        threshold_abierto=80, intervalo=3):
81      global blink_times, ultimo_abierto
82      if info is None or not hasattr(info, 'left_openness'): return False
83      left, right = info.left_openness, info.right_openness
84      ahora = time.time()
85      abierto = (left > threshold_abierto and right > threshold_abierto)
            if left > threshold_cerrado or right > threshold_cerrado else
            ultimo_abierto
86
87      if not ultimo_abierto and abierto:
88          blink_times = [t for t in blink_times if ahora - t < intervalo]
89          blink_times.append(ahora)
90          if len(blink_times) >= 2:
91              blink_times.clear()
92              ultimo_abierto = abierto
93              return True
94
95      ultimo_abierto = abierto
96      return False
97
98  # Filtro exponencial
99  def obtener_coordenadas_gaze(info, ancho, alto):
100     global gaze_buffer
101
102     # Verifica si left o right openess son None o un int.
103     if info is None or not hasattr(info, 'filtered_gaze_coordinates')
            or info.filtered_gaze_coordinates is None:
104         if not hasattr(obtener_coordenadas_gaze, "last_gaze"):
105             return -1, -1
106         # Si no hay nueva data, devolvemos la  ltima  posici n
                suavizada conocida
107         smoothed_x, smoothed_y = obtener_coordenadas_gaze.last_gaze
108         return smoothed_x, smoothed_y
109
110     gx, gy = info.filtered_gaze_coordinates
111
112     # Suavizado de los datps crudos del Buffer
113     gaze_buffer.append((gx, gy))
114     if len(gaze_buffer) > BUFFER_SIZE:
115         gaze_buffer.pop(0)
116
117     avg_x = sum(x for x, _ in gaze_buffer) / len(gaze_buffer)
118     avg_y = sum(y for _, y in gaze_buffer) / len(gaze_buffer)
119
120     # Inicializaci n de la posici n suavizada
121     if not hasattr(obtener_coordenadas_gaze, "last_gaze"):
122         obtener_coordenadas_gaze.last_gaze = (avg_x, avg_y)
123
124     # Aplicaci n de suavizado exponencial
```

```
125     last_x, last_y = obtener_coordenadas_gaze.last_gaze
126     smoothed_x = int(ALPHA * avg_x + (1 - ALPHA) * last_x)
127     smoothed_y = int(ALPHA * avg_y + (1 - ALPHA) * last_y)
128     obtener_coordenadas_gaze.last_gaze = (smoothed_x, smoothed_y) #
            Actualiza la  ltima  posici n
129
130     actual_x, actual_y = smoothed_x, smoothed_y
131
132     x = max(0, min(ancho, actual_x))
133     y = max(0, min(alto, actual_y))
134     return x, y
135
136 # Funcion que actualiza el puntero promedio basicamente el aim assist
137 def actualizar_puntero_promedio(gaze_x, gaze_y, botones, info):
138     global pointer_x, pointer_y, block_pointer_until
139
140     if info is None or gaze_x < 0 or gaze_y < 0:
141         return
142
143     ojos_cerrados = info.left_openness < THRESHOLD_CERRADO_BLOQUEO or
            info.right_openness < THRESHOLD_CERRADO_BLOQUEO
144     ahora = time.time()
145
146     # L gica de bloqueo cuando hay un parpadeo, se bloquea por 2 s
147     if ojos_cerrados:
148
149         block_pointer_until = ahora + 2.0
150         return
151
152     if ahora < block_pointer_until:
153         return
154
155     #L gica de Aim Assist
156     gaze_target_x, gaze_target_y = gaze_x, gaze_y
157
158     # Aim assist para la salida del problema, basicamente divide la
            pantalla en dos y si se observa despues de un umbral neutral se
            centra en el boton.
159     if len(botones) == 2 and botones[0].clave in ["si", "no"]:
160
161         # Definici n del umbral neutral (75px a cada lado del centro)
162         threshold_x = 75
163         center_x = ANCHO // 2
164
165         # Coordenadas de la zona neutral
166         zona_neutral_min_x = center_x - threshold_x
167         zona_neutral_max_x = center_x + threshold_x
168
169         # Chequeo de zona Neutral
170         if zona_neutral_min_x <= gaze_x <= zona_neutral_max_x:
171             # Si est  en la zona neutral, target_x/y se mantienen como
                    gaze_x/y
172             pass
```

```
173
174         # Chequeo de zona Confirmar (Izquierda)
175         elif gaze_x < zona_neutral_min_x:
176             # Snap al centro del bot n "Confirmar" (botones[0] si es "si")
177             if botones[0].clave == "si":
178                 gaze_target_x = botones[0].rect.centerx
179                 gaze_target_y = botones[0].rect.centery
180
181         #  Chequeo de zona Cancelar (Derecha)
182         elif gaze_x > zona_neutral_max_x:
183             # Snap al centro del bot n "Cancelar" (botones[1] si es "no")
184             if botones[1].clave == "no":
185                 gaze_target_x = botones[1].rect.centerx
186                 gaze_target_y = botones[1].rect.centery
187
188     else:
189         # Snap al centro del bot n para el men  principal, submen s y teclado num rico.
190         for b in botones:
191             if b.rect.collidepoint(gaze_x, gaze_y):
192                 gaze_target_x = b.rect.centerx
193                 gaze_target_y = b.rect.centery
194                 break
195
196     # Suavizado LERP para el puntero final, basicamente evita saltos tan bruzcos
197     pointer_x += ALPHA_POINTER * (gaze_target_x - pointer_x)
198     pointer_y += ALPHA_POINTER * (gaze_target_y - pointer_y)
199
200 # inicializacion de los botones de la interfaz
201 class Boton:
202     def __init__(self, x_min, x_max, y_min, y_max, texto, clave=None, color=AZUL, color_hover=AZUL_OSCURO):
203         self.x_min, self.x_max, self.y_min, self.y_max = x_min, x_max, y_min, y_max
204         self.rect = pygame.Rect(x_min, y_min, x_max - x_min, y_max - y_min)
205         self.texto, self.clave = texto, clave
206         self.color, self.color_hover = color, color_hover
207
208     def dibujar(self, pantalla, puntero_x, puntero_y):
209         color = self.color
210         if puntero_x is not None and puntero_y is not None:
211             if self.rect.collidepoint(puntero_x, puntero_y):
212                 color = self.color_hover
213         pygame.draw.rect(pantalla, color, self.rect, border_radius=12)
214         texto_surf = fuente.render(self.texto, True, BLANCO)
215         pantalla.blit(texto_surf, texto_surf.get_rect(center=self.rect.center))
216
217     def clic(self, puntero_x, puntero_y, info):
```

```
        if puntero_x is not None and puntero_y is not None:
            if self.rect.collidepoint(puntero_x, puntero_y):
                if detectar_doble_parpadeo(info):
                    return True
        return False


notificaciones = []
def mostrar_notificacion(texto, duracion=1000, color=GRIS):
    color_texto = NEGRO if color == GRIS else BLANCO
    notificaciones.append({"texto": texto, "inicio": pygame.time.
        get_ticks(), "duracion": duracion, "color": color, "color_texto"
        : color_texto})

def dibujar_notificaciones():
    ahora = pygame.time.get_ticks()
    for n in notificaciones[:]:
        if ahora - n["inicio"] > n["duracion"]:
            notificaciones.remove(n)
        else:
            surf = fuente.render(n["texto"], True, n["color_texto"])
            rect = surf.get_rect(center=(ANCHO // 2, ALTO // 2))
            pygame.draw.rect(pantalla, n["color"], rect.inflate(20, 20)
                , border_radius=12)
            pantalla.blit(surf, rect)

def confirmar_salida():
    # Par metros ajustados para abarcar casi toda la pantalla
        horizontalmente
    w = ANCHO // 2 - 100
    h = 150
    sep = 200

    # Calcular posiciones para el umbral "neutral" grande en el centro.
    center_x = ANCHO // 2
    y_b = ALTO // 2 + 50

    x_si_max = center_x - sep // 2
    x_si_min = x_si_max - w

    x_no_min = center_x + sep // 2
    x_no_max = x_no_min + w

    btn_si = Boton(x_si_min, x_si_max, y_b, y_b + h, "Confirmar", clave
        ="si", color=VERDE, color_hover=(0, 150, 0))
    btn_no = Boton(x_no_min, x_no_max, y_b, y_b + h, "Cancelar", clave=
        "no", color=ROJO, color_hover=(150, 0, 0))

    botones = [btn_si, btn_no]

    mensaje = fuente.render(" Seguro  que quieres salir? Doble parpadeo
        para seleccionar.", True, NEGRO)
    rect = mensaje.get_rect(center=(ANCHO // 2, ALTO // 2 - 50))
```

```
264
265     while True:
266         pantalla.fill(GRIS)
267         pantalla.blit(mensaje, rect)
268
269         info = gaze_follower.get_gaze_info()
270         gaze_x, gaze_y = obtener_coordenadas_gaze(info, ANCHO, ALTO)
271         actualizar_puntero_promedio(gaze_x, gaze_y, botones, info)
272
273         for b in botones:
274             b.dibujar(pantalla, pointer_x, pointer_y)
275
276         if TEST:
277             # Dibujar la zona neutral para visualizaci n
278             threshold_x = 75
279             center_x = ANCHO // 2
280             pygame.draw.rect(pantalla, (255, 255, 0, 50), (center_x -
                threshold_x, 0, 2 * threshold_x, ALTO), 5) # Amarillo
281
282             # El puntero rojo (gaze_x, gaze_y) sigue la data sin
                bloqueo/snap
283             pygame.draw.circle(pantalla, ROJO, (gaze_x, gaze_y), 15)
284             # El puntero azul (pointer_x, pointer_y) es el que se
                bloquea/snap y se suaviza con LERP
285             pygame.draw.circle(pantalla, AZUL_CURSOR, (int(pointer_x),
                int(pointer_y)), 15)
286
287         pygame.display.flip()
288
289         for evento in pygame.event.get():
290             if evento.type == pygame.QUIT:
291                 pygame.quit(); sys.exit()
292
293         for b in botones:
294             if b.clic(pointer_x, pointer_y, info):
295                 return b.clave == "si"
296
297         pygame.time.delay(10)
298
299 # Generar botones proporcionales a la pantalla
300 def generar_botones_cuadricula(lista_comandos, columnas=3, margen_sup
    =100, margen_lat=50):
301     botones = []
302     total = len(lista_comandos) + 1
303     filas = (total + columnas - 1) // columnas
304
305     w = (ANCHO - (columnas + 1) * margen_lat) // columnas
306     h = (ALTO - margen_sup - (filas + 1) * margen_lat) // filas
307
308     for idx, (cmd, nombre) in enumerate(lista_comandos):
309         fila = idx // columnas
310         col = idx % columnas
311         x_min = margen_lat + col * (w + margen_lat)
```

```
312         y_min = margen_sup + fila * (h + margen_lat)
313         botones.append(Boton(x_min, x_min + w, y_min, y_min + h, nombre
                , clave=cmd))
314
315
316     fila = (total - 1) // columnas
317     col = (total - 1) % columnas
318     x_min = margen_lat + col * (w + margen_lat)
319     y_min = margen_sup + fila * (h + margen_lat)
320     botones.append(Boton(x_min, x_min + w, y_min, y_min + h, "Salir",
            clave="salir", color=ROJO))
321     return botones
322
323 #Men  principal
324 def menu_principal():
325     comandos_lista = list(comandos_amigables.items())
326     botones = generar_botones_cuadricula(comandos_lista, columnas=3)
327
328     while True:
329         pantalla.fill(GRIS)
330         info = gaze_follower.get_gaze_info()
331         gaze_x, gaze_y = obtener_coordenadas_gaze(info, ANCHO, ALTO)
332         actualizar_puntero_promedio(gaze_x, gaze_y, botones, info)
333
334         titulo = fuente.render("DISPOSITIVOS:", True, NEGRO)
335         pantalla.blit(titulo, (50, 20))
336
337         for b in botones:
338             b.dibujar(pantalla, pointer_x, pointer_y)
339
340         if TEST:
341             pygame.draw.circle(pantalla, ROJO, (gaze_x, gaze_y), 15)
342             pygame.draw.circle(pantalla, AZUL_CURSOR, (int(pointer_x),
                    int(pointer_y)), 15)
343
344         dibujar_notificaciones()
345         pygame.display.flip()
346
347         for b in botones:
348             if b.clic(pointer_x, pointer_y, info):
349                 if b.clave == "salir":
350                     if confirmar_salida():
351                         return "EXIT"
352                 else:
353                     resultado = menu_subcomandos(b.clave)
354                     if resultado:
355                         return resultado
356
357         for evento in pygame.event.get():
358             if evento.type == pygame.QUIT:
359                 if confirmar_salida():
360                     return "EXIT"
361
```

```
# Teclado Num rico Gr fico , basicamente es para cuando se necesitan valores numericos
def teclado_numerico():
    #Definici n de la cuadr cula y las dimensiones
    columnas = 3
    filas = 4
    #Se observa como:
    # 1-2-3
    # 4-5-6
    # 7-8-9
    # Enter-0-Volver
    margen_lat, margen_sup = 80, 100
    espacio = 20

    w = (ANCHO - 2 * margen_lat - (columnas - 1) * espacio) // columnas
    h = (ALTO - 2 * margen_sup - (filas - 1) * espacio - 100) // filas # 100 es para dejar espacio al campo de valor

    teclas = [str(i) for i in range(1, 10)]
    teclas.append("Enter")
    teclas.append("0")
    teclas.append("Volver")

    claves = [str(i) for i in range(1, 10)] + ["enter", "0", "volver"]

    botones = []

    # 2. Creaci n de botones
    for i in range(len(teclas)):

        if i < 9:
            num = int(teclas[i])
            fila = (num - 1) // columnas
            col = (num - 1) % columnas
        else:
            fila = 3
            col = i - 9

        x_min = margen_lat + col * (w + espacio)
        y_min = margen_sup + 80 + fila * (h + espacio)
        texto = teclas[i]
        clave = claves[i]
        color = AZUL if texto.isdigit() or clave == "0" else (VERDE if clave == "enter" else ROJO)

        botones.append(Boton(x_min, x_min + w, y_min, y_min + h, texto, clave=clave, color=color))

    valor_ingresado = ""

    while True:
        pantalla.fill(GRIS)
        info = gaze_follower.get_gaze_info()
```

```
411         gaze_x, gaze_y = obtener_coordenadas_gaze(info, ANCHO, ALTO)
412         actualizar_puntero_promedio(gaze_x, gaze_y, botones, info)
413 
414         titulo = fuente.render("Ingrese Valor Num rico:", True, NEGRO)
415         pantalla.blit(titulo, (50, 20))
416 
417         texto_valor = fuente.render(valor_ingresado, True, NEGRO)
418         pantalla.blit(texto_valor, texto_valor.get_rect(center=(ANCHO
                //2, 80)))
419 
420         for b in botones:
421             b.dibujar(pantalla, pointer_x, pointer_y)
422 
423         if TEST:
424             pygame.draw.circle(pantalla, ROJO, (gaze_x, gaze_y), 15)
425             pygame.draw.circle(pantalla, AZUL_CURSOR, (int(pointer_x),
                    int(pointer_y)), 15)
426 
427         dibujar_notificaciones()
428         pygame.display.flip()
429 
430         for b in botones:
431             if b.clic(pointer_x, pointer_y, info):
432                 if b.clave.isdigit(): # N meros del 0 al 9
433                     valor_ingresado += b.clave
434                 elif b.clave == "enter":
435                     if valor_ingresado:
436                         return valor_ingresado
437                     else:
438                         mostrar_notificacion("Ingrese un valor primero"
                                , duracion=800, color=ROJO)
439                 elif b.clave == "volver":
440                     return None
441 
442         for evento in pygame.event.get():
443             if evento.type == pygame.QUIT:
444                 if confirmar_salida():
445                     return "EXIT"
446 
447 #Menu para subcomandos, o sea abre una pesta a cada vez que se
       selecciona un dispositivo/comando
448 def menu_subcomandos(comando):
449     subcomandos_dict = subcomandos_amigables[comando]
450     subcomandos_items = list(subcomandos_dict.items())
451 
452     subcomandos_items.append(("return_main", "Volver al Men  Principal
          "))
453 
454     columnas = 3
455     margen_sup = 100
456     margen_lat = 50
457 
458     total_items = len(subcomandos_items)
```

```
    filas = (total_items + columnas - 1) // columnas
    w = (ANCHO - (columnas + 1) * margen_lat) // columnas
    h = (ALTO - margen_sup - (filas + 1) * margen_lat) // filas

    botones = []

    for idx, (clave, nombre) in enumerate(subcomandos_items):
        fila = idx // columnas
        col = idx % columnas
        x_min = margen_lat + col * (w + margen_lat)
        y_min = margen_sup + fila * (h + margen_lat)

        color = AZUL
        if clave == "return_main":
            color = ROJO
            fila = (total_items - 1) // columnas
            col = (total_items - 1) % columnas
            x_min = margen_lat + col * (w + margen_lat)
            y_min = margen_sup + fila * (h + margen_lat)

        botones.append(Boton(x_min, x_min + w, y_min, y_min + h, nombre
            , clave=clave, color=color))

    while True:
        pantalla.fill(GRIS)
        info = gaze_follower.get_gaze_info()
        gaze_x, gaze_y = obtener_coordenadas_gaze(info, ANCHO, ALTO)
        actualizar_puntero_promedio(gaze_x, gaze_y, botones, info)

        titulo = fuente.render(f"Control {comandos_amigables[comando]}"
            , True, NEGRO)
        pantalla.blit(titulo, (50, 20))

        for b in botones:
            b.dibujar(pantalla, pointer_x, pointer_y)

        if TEST:
            pygame.draw.circle(pantalla, ROJO, (gaze_x, gaze_y), 15)
            pygame.draw.circle(pantalla, AZUL_CURSOR, (int(pointer_x),
                int(pointer_y)), 15)

        dibujar_notificaciones()
        pygame.display.flip()

        for b in botones:
            if b.clic(pointer_x, pointer_y, info):
                if b.clave == "return_main":
                    return None

                if b.clave in SUBCOMANDOS_NUMERICOS:
                    valor_num = teclado_numerico()
                    if valor_num == "EXIT":
                        return "EXIT"
```

```
509                     if valor_num is not None:
510                         return (comando, valor_num)
511
512                 else:
513                     return (comando, b.clave)
514
515         for evento in pygame.event.get():
516             if evento.type == pygame.QUIT:
517                 if confirmar_salida():
518                     return "EXIT"
519
520 # Ejecuci n, este codigo quedo muy desordenado. Estoy considerando
    separar este codigo y el cliente HTTP en 3 archivos de python.
521 #uno de cliente, corro el cliente, e importa la interfaz
522 #uno de interfaz, incluye toda la interfaz grafica
523 #uno de cursor, incluye la inicializacion del gaze follower y los
    metodos de deteccion de parpadeo y las correciones por el aim assist
     y suavizdo con fikltro exponencial/LERP
524 if __name__ == "__main__":
525     try:
526         while True:
527             resultado = menu_principal()
528
529             if resultado == "EXIT":
530                 break
531
532             if resultado:
533                 comando, subcomando = resultado
534                 mostrar_notificacion("Comando enviado correctamente",
                        duracion=1500, color=VERDE)
535
536                 try:
537                     enviar_request(comando, subcomando)
538                 except NameError:
539                     print("Advertencia: Funci n enviar_request no
                            definida.")
540
541                 print("Comando:", comando, "Subcomando:", subcomando)
542
543     except KeyboardInterrupt:
544         print("\nInterrupci n por teclado detectada. Cerrando procesos
                de hardware y Pygame...")
545         pass
546
547     finally:
548         # Cierra todo si se detiene el problema o se utiliza ctrl + c/
                keyboard interrupt, esto es por que si no se detiene al
                parecer el gaze fololower se mantiene encendido,
549         #junto con la camara.
550         if 'gaze_follower' in locals() and gaze_follower is not None:
551             gaze_follower.stop_sampling()
552             print("Gaze follower sampling detenido.")
553
```

```
554             if 'pygame' in sys.modules:
555                 pygame.display.quit()
556                 pygame.quit()
557                 print("Pygame cerrado de forma segura.")
558 
559             sys.exit()
```