



**Universidad de Magallanes
Facultad de Ingeniería
Departamento de Ingeniería en Computación e
Informática**

KubiBot: Informe Técnico

Ivan Mansilla
Ayrton Morrison
Ingeniería Civil en Computación e Informática

Docente guía: Mgt. Eduardo Peña
Curso: Taller de Integración

2025

Resumen

Resumen bien epico carajo

Índice general

Índice de figuras	ii
1. Introducción	1
2. Objetivos	2
2.1. Objetivo Principal	2
2.2. Objetivos Secundarios	2
3. Marco teórico	3
3.1. Robótica móvil	3
3.2. Inteligencia artificial	3
3.3. Servidores y clientes	3
3.4. Arduino UNO	3
3.5. Raspberry PI 5	3
4. Desarrollo	4
4.1. Análisis previo	4
4.1.1. Módulos del prototipo	4
4.1.2. Diseño	5
4.2. Movimiento	6
4.2.1. Prototipo inicial	6
4.2.2. Segundo prototipo	8
4.2.3. Tercer prototipo	8
4.2.4. Prototipo final	10
4.2.5. Código	11
4.3. Comunicacion	14
4.3.1. Arquitectura inicial	14
4.3.2. Primer Prototipo funcional	15
4.3.3. Arquitectura final	16
4.3.4. Implementación del cliente en Raspberry Pi	17
4.4. Comunicación Raspberry-Arduino	18
4.5. Diseño	20
5. Conclusión	22

Índice de figuras

4.1. Puente H L293D	6
4.2. Diagrama del sensor ultrasónico HC-SR04	7
4.3. Conexión de motores al TB6612FNG	9
4.4. Conexiones del motor driver shield L293D	10
4.5. Diagrama de clases del módulo de movimiento	11
4.6. Diagrama de comunicacion Servidor-Cliente	15
4.7. Diagrama de comunicacion final Servidor-Cliente	16
4.8. Boceto inicial hecho por Iván Mansilla	20
4.9. Modelo 3D de la carcasa diseñado por Nicolás Poblete	21

Introducción

Objetivos

2.1. Objetivo Principal

Diseñar y desarrollar un prototipo de robot de compañía móvil, integrado con un modelo de inteligencia artificial conversacional, capaz de interactuar de forma autónoma y asistir al usuario dentro de un entorno doméstico

2.2. Objetivos Secundarios

Se plantean los siguientes objetivos secundarios:

1. Desarrollar el sistema de movimiento autónomo del robot, incluyendo diseño del chasis, integración de motores y programación de sistema de evitación de obstáculos, utilizando un microcontrolador Arduino Uno
2. Implementar inteligencia artificial en la plataforma Raspberry Pi, abarcando la configuración del sistema, creación de API como intermediario entre host de la LLM y Raspberry PI, e integración de modelos de reconocimiento de voz (Speech-to-text)
3. Integrar periféricos de comunicación (entrada y salida) para la interacción usuario-robot, esto incluyendo instalación de micrófono y altavoz
4. Ensamblar los distintos módulos de hardware del prototipo, procurando coordinación entre Arduino y Raspberry
5. Validar la funcionalidad completa del robot mediante pruebas de software y hardware

Marco teórico

3.1. Robótica móvil

3.2. Inteligencia artificial

3.3. Servidores y clientes

3.4. Arduino UNO

El Arduino UNO es una placa de desarrollo basada en el microcontrolador ATmega328P, ampliamente utilizada en proyectos de electrónica y robótica debido a su facilidad de uso y versatilidad. Cuenta con 14 pines digitales de entrada/salida, 6 entradas analógicas, un cristal oscilador de 16 MHz, una conexión USB, un conector de alimentación y un botón de reinicio.

El Arduino UNO permite a los usuarios programar y controlar dispositivos electrónicos mediante el entorno de desarrollo integrado (IDE) de Arduino, que utiliza un lenguaje de programación basado en C/C++.

3.5. Raspberry PI 5

Desarrollo

4.1. Análisis previo

El punto de partida de este proyecto fue el interés en explorar las aplicaciones prácticas de la inteligencia artificial generativa en un formato físico interactuable, con el fin de crear un dispositivo que genere cercanía y acompañamiento al usuario.

El primer concepto consistía en utilizar un hardware disponible en la institución: una cabeza robótica estática. La idea era dotar a esta cabeza de capacidades conversacionales, aprovechando su estructura existente para simular interacción humana (movimiento de ojos o boca).

Sin embargo, tras un análisis preliminar, se descartó la idea, y se prefirió crear un hardware de cero, ya que entregaría mayor libertad de diseño e integración. Esto llevó entonces hacia un concepto nuevo: un prototipo de robot móvil y compacto. Esta nueva dirección se eligió por dos ventajas estratégicas:

1. **Simplicidad de integración:** Aunque la movilidad añade el desafío de crear un sistema de navegación, diseñar un chasis propio desde cero simplifica enormemente la integración y cohesión de los componentes que se quisieran utilizar, ya que no tienen que obligatoriamente adaptarse a un sistema pre existente.
2. **Mayor Atractivo e Interacción:** Se determinó que un robot capaz de moverse por la habitación y reaccionar físicamente a su entorno sería percibido por el público como un dispositivo más dinámico y, en definitiva, más atractivo y cercano como "compañero", cumpliendo así el objetivo inicial del proyecto de una forma más efectiva.

4.1.1. Módulos del prototipo

Una vez la idea general definida, fue fundamental realizar un análisis de los componentes necesarios para el funcionamiento del prototipo. Para un desarrollo en paralelo y modular, se dividió el robot en dos módulos principales e inicialmente independientes entre sí, con el objetivo de ser conectados una vez cada uno estuviese lo suficientemente avanzado:

1. **Movimiento:** Este módulo englobaría toda la locomoción física del robot. Se compondría de un chasis estructural, un sistema de ruedas impulsadas por motores y un sensor ultrasónico para la detección de obstáculos. Para controlar el movimiento se

escogió el microcontrolador Arduino Uno, debido a su simplicidad de programación y su disponibilidad en el departamento.

2. **Comunicacion:** Modulo en donde residiría todo lo relacionado con la comunicación con el usuario. Se determinó inicialmente el levantar localmente una LLM en un Raspberry PI 5, debido a su diseño compacto y potencia. En ella además se conectarían los dispositivos de entrada y salida. Se determinó que la forma de comunicación más cercana y amigable sería a través de voz.

4.1.2. Diseño

Una vez la funcionalidad determinada, se analizó la presentación visual del prototipo. Por las características de los componentes y los recursos limitados, se necesitaba una carcasa ligera y a su misma vez accesible. Se optó entonces por crear o buscar un modelo para imprimir en 3D, lo que entonces implicó que el diseño además debiese ser simple para evitar problemas con el filamento o de ensamblaje.

Finalmente se optó por una carcasa completamente cúbica, ya que no solo resulta sumamente fácil de ensamblar, sino que además le da al robot una apariencia agradable. Este diseño entonces se bosquejó, para luego ser trabajado y adaptado en un modelo 3D por Nicolás Poblete, quien durante todo el transcurso del proyecto prestó apoyo en lo que es diseño e impresión del prototipo.

4.2. Movimiento

El movimiento durante el desarrollo del proyecto fue uno de los aspectos más desafiantes, no tanto por la complejidad técnica, sino por la limitación de recursos y problemas técnicos imprevistos que surgieron durante la implementación.

Como se mencionó anteriormente, este fue programado con el ambiente de Arduino IDE, utilizando un Arduino UNO como microcontrolador principal. El código fue escrito en C++, utilizando librerías estándar de arduino y un enfoque orientado a objeto para el control de cada componente.

4.2.1. Prototipo inicial

Inicialmente se planificó un sistema de locomoción basado en un chasis con dos ruedas a motor, dos ruedas libres y un sensor ultrasónico para la detección de obstáculos, controlados por un Arduino Uno. El puente H utilizado para controlar los motores fue el L293D, componente que se encontraba disponible en el laboratorio; continuación se presenta el diagrama de este en la figura

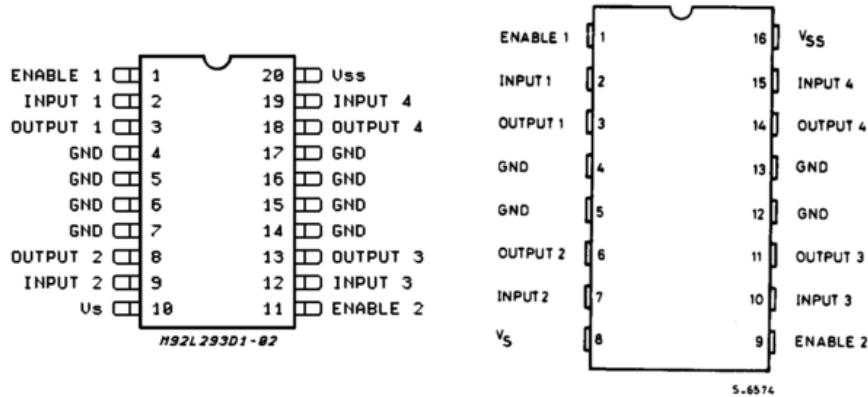


Figura 4.1: Puente H L293D

Los motores DC seleccionados fueron de 6V y 255 RPM, alimentados por una batería externa de 9V para asegurar un suministro de energía estable.

El sensor ultrasónico HC-SR04 se utilizó para medir la distancia a los obstáculos, enviando señales de ultrasonido y midiendo el tiempo que tarda en recibir el eco. Este sensor se conectó al Arduino, que procesaba las lecturas y cortaba el movimiento de los motores si se detectaba un obstáculo. A continuación se presenta el diagrama del sensor

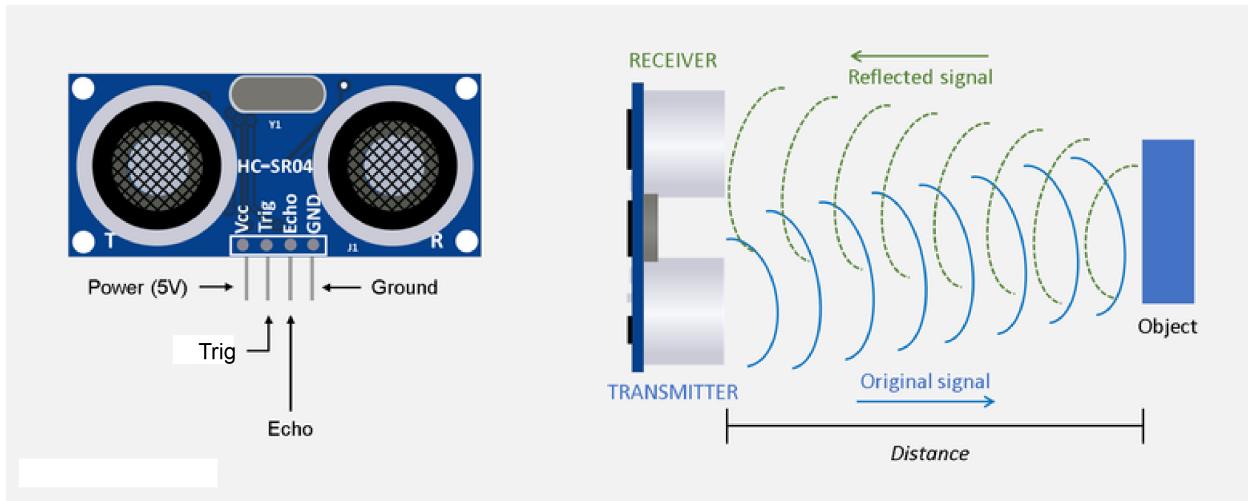


Figura 4.2: Diagrama del sensor ultrasónico HC-SR04

Los problemas comenzaron a surgir durante las primeras pruebas. El primer problema surgió con el sensor ultrasónico, ya que las lecturas eran inconsistentes y a menudo incorrectas. Después de reemplazar el sensor y revisar las conexiones, se descubrió que el problema en realidad estaba en el código de arduino, que no estaba manejando bien los tiempos de espera y las interrupciones. Tras corregir el código, el sensor comenzó a funcionar correctamente.

El siguiente problema fue netamente de diseño, ya que no se consideró adecuadamente el cómo el prototipo giraría al detectar un obstáculo. Ya que el chasis ya se encontraba impreso y era complicado modificarlo para añadir un mecanismo con un servo, se decidió realizar el giro netamente deteniendo un motor y dejando el otro activo.

4.2.2. Segundo prototipo

Algo que no se consideró en el primer prototipo fue la decisión de camino que el robot tomaría al detectar un obstáculo. Es por esto que se decidió implementar al sistema de detección de obstáculos un servomotor analógico conectado al arduino, al que se encontraría ensamblado el sensor ultrasónico.

De esta forma, se ejecutaría el siguiente flujo:

```
1   while(true):
2       if (distancia_obstaculo < umbral de seguridad):
3           1. Detener ambos motores.
4           2. Retroceder hasta una distancia segura.
5           3. Girar el servo a la izquierda y medir distancia.
6           4. Girar el servo a la derecha y medir distancia.
7           5. Comparar ambas distancias.
8           6. Girar en la dirección con mayor distancia libre.
9   else:
10      Continuar moviéndose hacia adelante.
```

Listado 4.1: Flujo de detección de obstáculos con servo

4.2.3. Tercer prototipo

Al probar el giro y el movimiento se encontraron dos problemas graves:

1. **Falta de potencia:** Se empezó a notar que el robot tenía dificultades para moverse, sobre todo en superficies con algo de fricción. Esto se debió a que la batería de 9V se estaba quedando sin carga, lo que resultaba en una potencia insuficiente.
2. **Giro ineficiente:** El sistema de giro implementado en donde se detenía un motor y se dejaba el otro activo no funcionó como se esperaba. El robot prácticamente no giraba, tanto por la fricción del suelo como por la inercia ejercida por la rueda detenida.

Para solucionar el primer problema se decidió cambiar la fuente de alimentación a un portapila de 6 pilas AA, entregando un total de 9V. Esto proporcionó no solo la potencia necesaria, sino que además la posibilidad de cambiar las pilas fácilmente cuando se agotaran.

Para solucionar el segundo problema se tuvo que hacer un cambio importante en el circuito, ya que para combatir la inercia y fricción con una potencia adecuada, se decidió

añadir dos motores adicionales, así el sistema de locomoción convirtiéndose en uno 4x4. Esto implicó rediseñar el chasis para acomodar los nuevos motores y ruedas, lo que fue posible gracias a la impresión 3D.

También se tuvo que cambiar el puente H L293D por uno que soportara mayor corriente, ya que investigando se descubrió que con este habría riesgo de sobrecalentamiento si se utilizaban los 4 motores simultaneamente. El nuevo puente H utilizado fue el TB6612FNG, el cual además de soportar mayor corriente, se encontraba disponible en el laboratorio. A continuación se presenta el diagrama de conexión de este

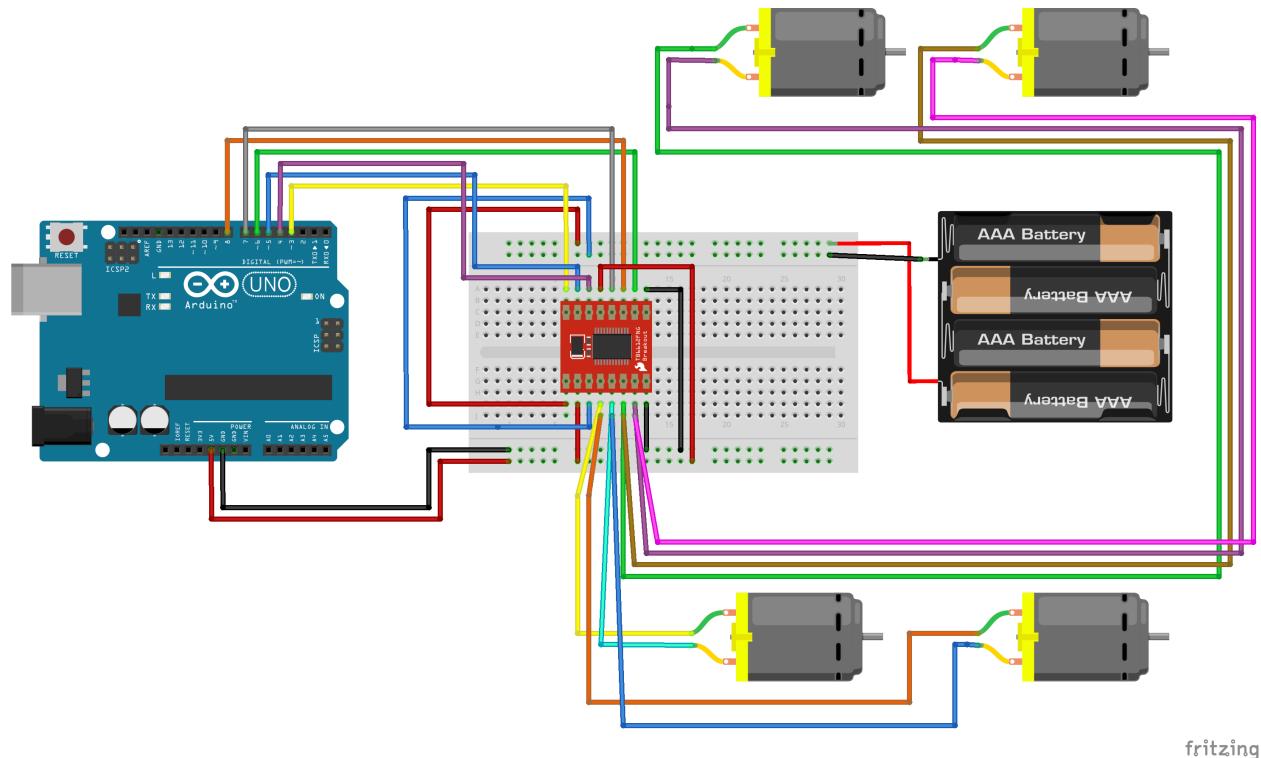


Figura 4.3: Conexión de motores al TB6612FNG

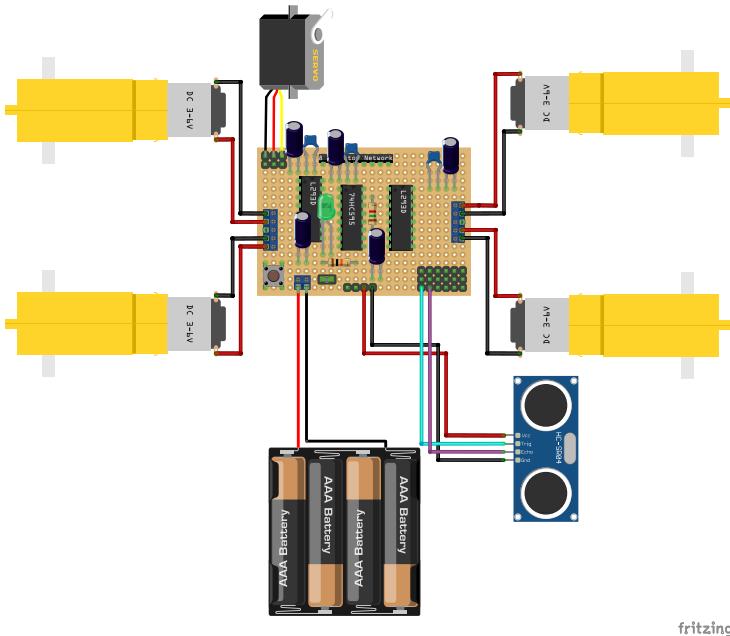
4.2.4. Prototipo final

A pesar de la investigación realizada, el puente H TB6612FNG comenzó a sobrecalentarse de todas formas al utilizar los 4 motores simultáneamente, lo que generó preocupación por la seguridad del prototipo.

Se tuvo que buscar una solución alternativa, y se optó por conseguir un *shield* de motor para Arduino que fuese adecuado para el manejo de 4 motores DC. El *shield* elegido fue el *Motor Driver Shield L293D*, el cual contiene dos puentes H L293D (utilizado en el primer prototipo) integrados.

Este *shield* resultó en un gran avance y beneficio para el prototipo, ya que no solo resolvió el problema de sobrecalentamiento, sino que además simplificó enormemente el cableado y el suministro de energía del circuito.

El shield se conecta directamente al Arduino, y cuenta con terminales tanto para los motores, como para el servo y el sensor ultrasónico, además cuanta con una entrada para una fuente de alimentación externa, que alimenta tanto a los motores como al Arduino y los demás componentes. Entonces, fue posible alimentar todo el sistema con una sola fuente de alimentación y remover uno de los portapilas, además de eliminar el protoboard utilizado para las conexiones, ganando espacio y quitando peso. A continuación se presenta el diagrama de conexión del shield:



4.2.5. Código

El código final se estructuró en torno a dos clases: `ArduinoRobot` y `RaspberryPi`. La primera se encarga de abstraer el control de los cuatro motores, el sensor ultrasónico y el servomotor; la segunda modela el estado de la comunicación con la Raspberry Pi mediante comandos seriales, lo que se detallará más adelante.

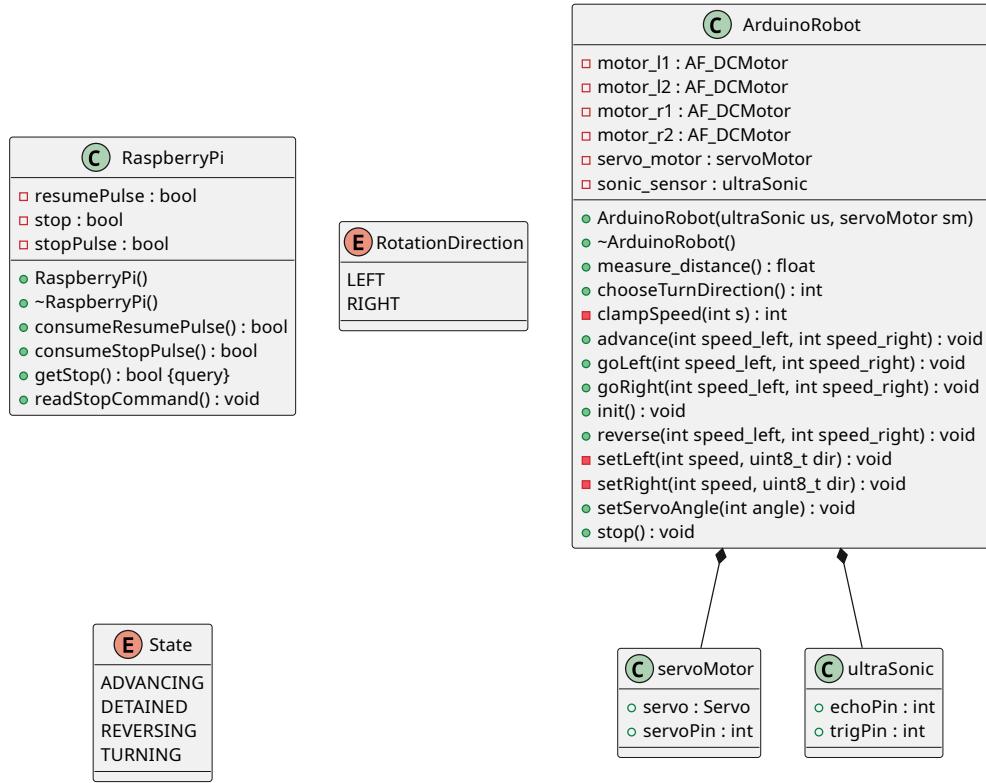


Figura 4.5: Diagrama de clases del módulo de movimiento

Cada par de motores se controla utilizando la librería `AFMotor`. Se definieron métodos para avanzar, retroceder, girar y detener los motores, permitiendo un control sencillo del movimiento del robot. Se aprovechó la programación orientada a objetos para encapsular el control de cada motor, lo que facilitó la gestión del movimiento en conjunto.

El sensor ultrasónico HC-SR04 se gestiona mediante un *struct* llamado `ultraSonic`, hijo de la clase `ArduinoRobot`, y el método `measure_distance()`, que genera un pulso (*TRIG*), mide el tiempo de eco (*ECHO*) mediante la función `pulseIn()` y calcula la distancia en centímetros utilizando la velocidad del sonido.

El servomotor es representado a través del *struct* `servoMotor`, y se controla con la librería

estándar de Arduino Servo. El método `setServoAngle()` es el responsable de rotar el servo a un ángulo específico.

La lógica de navegación se implementó con estados, ADVANCING, REVERSING, TURNING y DETAINED. En ADVANCING, el robot avanza recto mientras la distancia al obstáculo sea mayor que un umbral de seguridad. Si se detecta un obstáculo, se pasa a REVERSING, donde el robot retrocede durante un tiempo determinado hasta quedar a una distancia segura. Luego, en TURNING, el robot gira sobre su propio eje durante un tiempo fijo en la dirección elegida por el método `chooseTurnDirection()`, que compara las distancias medidas cuando el servo mira hacia la izquierda y hacia la derecha y retorna LEFT o RIGHT. El estado DETAINED se utiliza para detener completamente el robot cuando así lo indique la Raspberry Pi.

A continuación se presenta un pseudocódigo del flujo principal del programa:

```
1 inicializar_robot()
2
3 while(true):
4     distancia = medir_distancia_cm()
5
6     if estado_robot == AVANZANDO:
7         if distancia < UMBRAL_SEGURIDAD:
8             estado_robot = RETROCEDIENDO
9         else:
10            avanzar()
11
12    else if estado_robot == RETROCEDIENDO:
13        if distancia < UMBRAL_SEGURIDAD:
14            retroceder()
15        else:
16            // Mirar izquierda y derecha con el servo
17            mover_servo_izquierda()
18            dist_izq = medir_distancia_cm()
19
20            mover_servo_derecha()
21            dist_der = medir_distancia_cm()
22
23            mover_servo_frente()
```

```
25     if dist_izq > dist_der:  
26         direccion_giro = IZQUIERDA  
27     else:  
28         direccion_giro = DERECHA  
29  
30     estado_robot = GIRANDO  
31     tiempo_inicio_giro = tiempo_actual_ms()  
32  
33 else if estado_robot == GIRANDO:  
34     if tiempo_actual_ms() - tiempo_inicio_giro < TIEMPO_GIRO:  
35         girar(direccion_giro)  
36     else:  
37         estado_robot = AVANZANDO
```

Listado 4.2: Flujo principal de control en Arduino

4.3. Comunicacion

Como se establecio previamente en el analisis del problema, se decidió utilizar un Raspberry PI como el núcleo del modulo de comunicación.

Este en un principio se pensó para alojar localmente un modelo de lenguaje, además de manejar la entrada y salida de audio y todo el postprocesamiento. Sin embargo, tras las primeras pruebas, varios problemas fueron encontrados y la arquitectura del sistema tuvo que ser replanteada.

Todo el código desarrollado para este módulo fue escrito en Python, ya que en este lenguaje existe una amplia gama de librerías y prototipos ya hechos para el manejo de IA, audio y comunicación en red, lo que facilita enormemente el desarrollo.

4.3.1. Arquitectura inicial

Para el modelo de lenguaje, se optó utilizar un LLM localmente alojado, debido tanto a la limitación de recursos económicos para utilizar una API comercial, como a la intención de explorar el uso de modelos de lenguaje abiertos. Se seleccionó la herramienta *Ollama*, que permite realizar esto.

Sin embargo, al intentar utilizar diversos modelos, se descubrió que el Raspberry PI 5 no contaba con la potencia suficiente para ejecutar ninguno de ellos de manera fluida. Incluso los modelos más livianos presentaban tiempos de respuesta inaceptables o provocaban que el sistema se congelara, lo que supuso un obstáculo significativo para el desarrollo del proyecto.

Después de considerar diversas opciones, se decidió cambiar la arquitectura del sistema para utilizar el Raspberry PI no como host del modelo del lenguaje, sino como un intermediario entre el usuario y un servidor externo que alojaría el modelo. De esta forma, el Raspberry PI se encargaría de manejar la entrada y salida de audio, mientras que el procesamiento intensivo requerido por el modelo de lenguaje se delegaría a una máquina más potente.

4.3.2. Primer Prototipo funcional

El sistema de comunicación se diseñó adaptando una arquitectura cliente-servidor¹. El Raspberry Pi actúa como cliente, mientras que una máquina externa con mayor capacidad de procesamiento aloja el modelo de lenguaje y actúa como servidor. Esta separación permite que el Raspberry Pi maneje las tareas de entrada y salida de audio, mientras que el servidor se encarga del procesamiento intensivo requerido por el modelo de lenguaje.

En la figura 4.6 se muestra un diagrama de funcionamiento entre los componentes principales del sistema de comunicación.

`client.py` es el script principal que corre en el Raaspberry PI. Este se encarga de detectar constantemente si el usuario ha emitido una entrada de voz específica o Wake Word (en este caso "Hey Bot"). Al detectar esta entrada, el script graba la voz del usuario durante un periodo de tiempo y la envía al servidor a través de una solicitud HTTP POST.

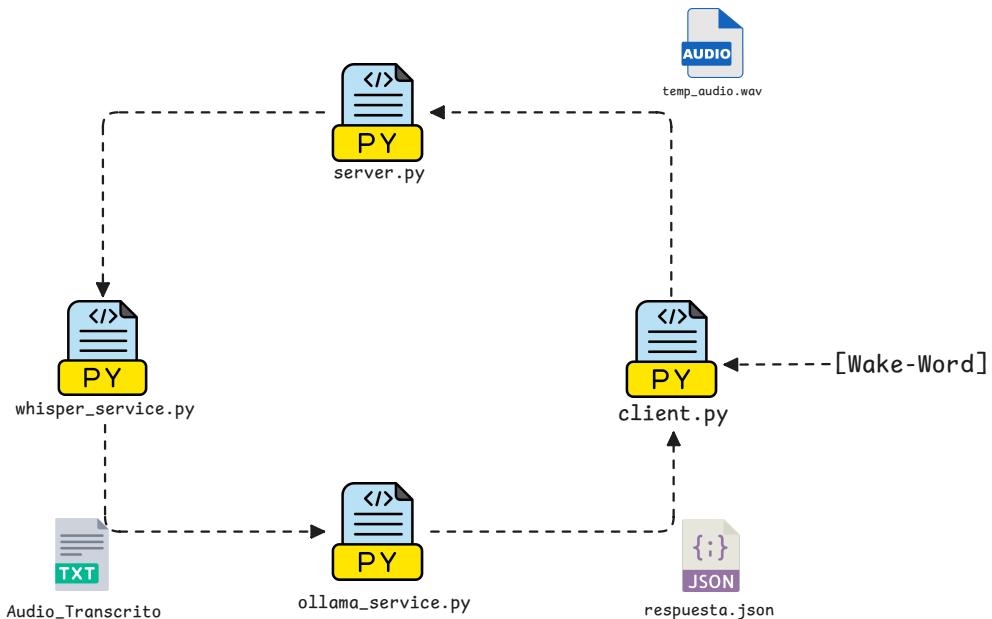


Figura 4.6: Diagrama de comunicacion Servidor-Cliente

¹Modelo de aplicación que distribuye las tareas entre los proveedores de recursos o servicios y solicitantes de servicios. [1]

4.3.3. Arquitectura final

Finalmente, el servidor se implementó utilizando `Flask`[2] junto con `Flask-SocketIO`[3], permitiendo combinar una API web ligera con comunicación en tiempo real mediante Web-Sockets. El script principal `server_api.py` define los eventos que gestionan la recepción y el envío de audio entre el cliente (Raspberry Pi) y el servidor. En adicon a lo anterior, el servidor permite manejar mas de un cliente en caso de que sea necesario.

Para la seguridad básica del sistema, cada cliente debe enviar un *token* de autenticación en la cabecera `Auth`. El servidor valida este valor contra una variable de entorno, si el token es incorrecto, la conexión se rechaza. En caso contrario, se crea un identificador de sesión para el cliente conectado y se procede con el flujo de trabajo.

El flujo de funcionamiento apreciado en al figura 4.7 es el siguiente: al recibir la **wake word**, el cliente comienza a grabar la voz del usuario, enviando fragmentos de audio al servidor mediante un evento específico. Finalizada la grabacion se procesa el audio para obtener la transcripcion del mensaje mediante whisper. Posteriormente el texto resultante se procesa mediante Piper para obtener la respuesta en audio, la cual es enviada junto al texto al cliente para su reproduccion.

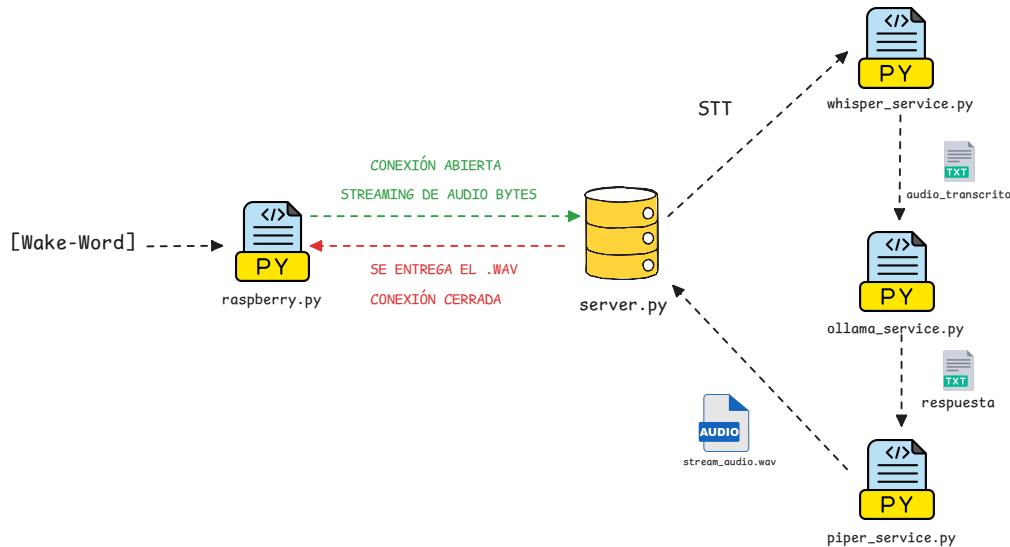


Figura 4.7: Diagrama de comunicacion final Servidor-Cliente

4.3.4. Implementación del cliente en Raspberry Pi

El cliente, implementado en el script `client.py`, se ejecuta en el Raspberry Pi y es responsable de la interacción directa con el usuario. Su función principal es detectar la *wake word*, grabar la voz del usuario, enviarla al servidor y reproducir el audio de respuesta.

Para la detección de la palabra clave se utiliza la librería **Porcupine**, que permite reconocer localmente una *wake word* predefinida sin necesidad de conexión a Internet. El cliente inicializa **Porcupine** con una clave de acceso (`ACCESS_KEY`) y un modelo de palabra clave, y escucha de manera continua el micrófono. Cuando la salida del modelo indica que la palabra ha sido detectada, el sistema reproduce un sonido breve de inicio y pasa al modo de grabación.

La grabación de la voz se realiza con la librería **pvrecorder**, que captura audio en forma de marcos (*frames*) de muestras. Sobre este flujo se implementa un esquema simple de detección de actividad de voz (VAD), basado en un umbral de amplitud y en un contador de silencio. Mientras se detecta voz, el cliente envía cada frame codificado como audio PCM a través del evento `audio_chunk` de Socket.IO. Cuando se detecta un periodo de silencio superior a un límite configurado, o cuando se alcanza una duración máxima, el cliente detiene la captura, reproduce un sonido de fin de grabación y emite el evento `end_of_audio`.

Durante todo este proceso, el cliente mantiene una conexión persistente con el servidor mediante Socket.IO, incluyendo el `API_TOKEN` en las cabeceras para autenticación. Una vez que el servidor procesa la petición, envía de vuelta la respuesta textual y, principalmente, la respuesta en audio a través del evento `audio_response`. El cliente guarda temporalmente este audio en un archivo y lo reproduce usando `aplay`, eliminando el archivo una vez terminado.

De esta forma, el Raspberry Pi actúa como una interfaz conversacional local, que gestiona la experiencia de usuario (*wake word*, sonidos de inicio y fin, reproducción de la voz sintética) mientras delega el procesamiento intensivo de IA al servidor remoto.

4.4. Comunicación Raspberry-Arduino

La comunicación entre el Raspberry PI y el Arduino consiste en enviar una señal desde el Raspberry PI al Arduino para activar o desactivar el movimiento del robot, dependiendo de si el robot está hablando o escuchando al usuario.

El principal obstáculo para lograr esto radica principalmente en la diferencia de voltajes entre ambos dispositivos. El Raspberry PI opera a 3.3V en sus pines GPIO, mientras que el Arduino funciona a 5V. Esta diferencia puede causar daños permanentes en el Raspberry PI si se conecta directamente un pin de salida del Arduino a un pin de entrada del Raspberry. Para resolver esto se consideraron dos opciones principales:

- **Circuito con relay:** Utilizar un relay para aislar eléctricamente ambos dispositivos. El Arduino podría activar el relay para enviar una señal al Raspberry PI sin que haya una conexión directa entre los pines de ambos dispositivos.
- **Cable USB:** Utilizar la comunicación serial a través de un cable USB. El Arduino puede enviar datos al Raspberry PI a través de la conexión USB, y el Raspberry PI puede interpretar estos datos para controlar el movimiento del robot.

Después de evaluar ambas opciones, más que nada por un tema de diseño se optó en primera instancia por el uso de un relay. Sin embargo, tras probar el circuito, no se logró una comunicación estable entre ambos dispositivos. Es por esto, que al final se optó por la comunicación serial a través de un cable USB, que además de ser más simple de implementar, resultó en una comunicación mucho más estable y confiable.

Para implementar la comunicación serial, se utilizó la librería `pyserial` en el Raspberry PI para enviar comandos al Arduino. El Arduino, por su parte, fue programado para escuchar estos comandos y activar o desactivar el movimiento del robot en consecuencia. A continuación se presenta un pseudocódigo del flujo de comunicación:

```
1
2 comando_serial = leer_comando_serial()
3
4 if comando_serial == "S":
5     cambiar_estado_robot("DETENIDO")
6 elif comando_serial == "R" && estado_robot == "DETENIDO":
7     cambiar_estado_robot("MOVIENDOSE")
```

```

8
9 switch(estado_robot):
10     case "MOVIENDOSE":
11         mover_adelante()
12     case "DETENIDO":
13         detener_motores()
14     case "GIRANDO":
15         ejecutar_giro()

```

Listado 4.3: Recepción de comandos seriales en Arduino

```

1 if robot_escuchando || robot_hablando:
2     enviar_comando_serial("S") // "S" para detener
3 else:
4     enviar_comando_serial("R") // "R" para reanudar

```

Listado 4.4: Envio de comandos seriales en Raspberry

En la implementación final, se definieron dos comandos principales enviados desde el Raspberry Pi al Arduino a través del puerto serie: 'S' para detener el robot y 'R' para reanudar el movimiento. El Arduino, mediante la clase `RaspberryPi`, lee continuamente el puerto serie y actualiza un estado interno de parada. Cuando el robot está hablando o escuchando (es decir, mientras se procesa una interacción de voz), el Raspberry Pi envía 'S' para forzar el estado `DETAINED`, lo que detiene inmediatamente los motores. Una vez finalizada la respuesta de voz, se envía 'R', permitiendo al Arduino volver al estado `ADVANCING` y retomar la navegación autónoma.

4.5. Diseño

El diseño físico de Kubibot se abordó con el objetivo de lograr un prototipo compacto, ligero y visualmente amigable, que al mismo tiempo facilitara el acceso a los componentes internos para pruebas y mantenimiento. Para ello se optó por una carcasa cúbica impresa en 3D. Antes del diseño final, se realizó el siguiente bosquejo inicial:



Figura 4.8: Boceto inicial hecho por Iván Mansilla

Con esta idea en mente, se procedió a diseñar el modelo 3D de la carcasa con la ayuda de Nicolás Poblete, quien se encargó de la modelación y la impresión de las piezas. Este diseño fue hecho de tal forma que los componentes internos tuviesen el espacio necesario, además de sus piezas ser modulares y desmontables, o sea, que cada pieza pudiese ser removida individualmente para acceder a los componentes internos con facilidad. A continuación se muestran algunas imágenes del modelo 3D:

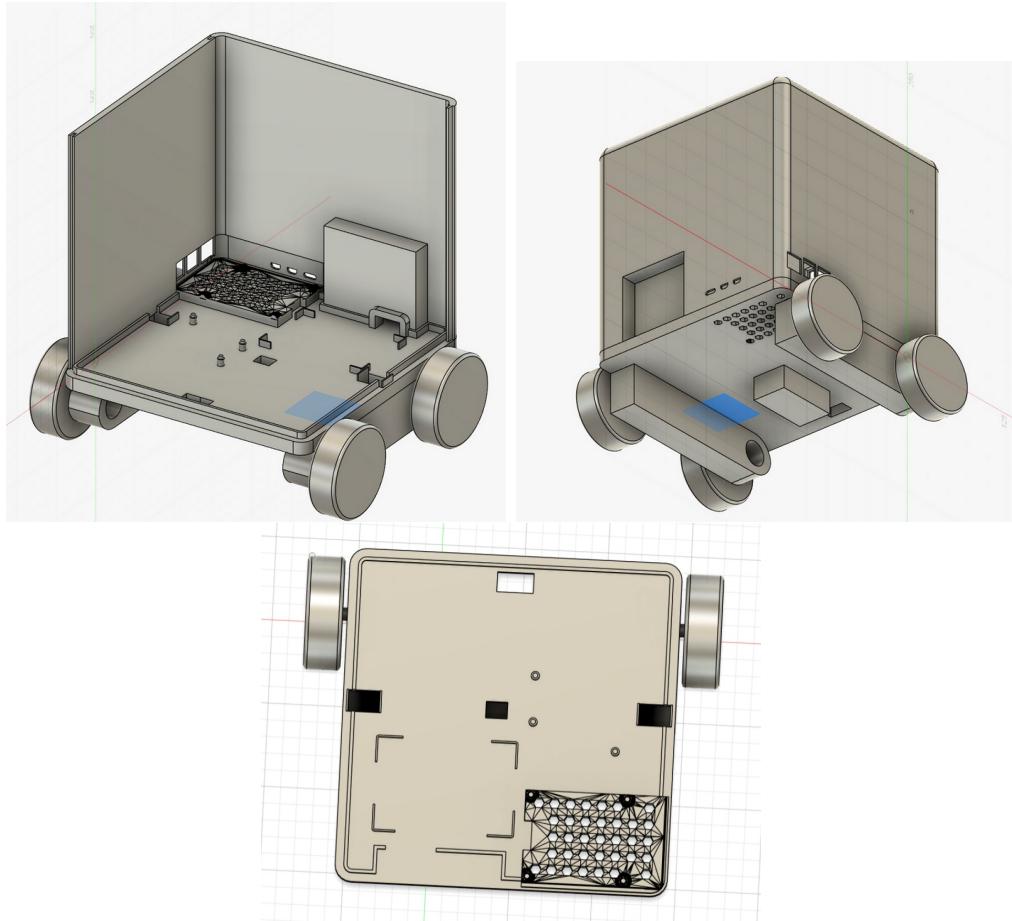


Figura 4.9: Modelo 3D de la carcasa diseñado por Nicolás Poblete

Se puede apreciar como en el diseño se consideraron espacios específicos para cada componente, además de orificios para la ventilación, el acceso a puertos, chasis para los motores y un espacio para la fuente de poder.

Conclusión

Bibliografía

- [1] IBM, *Client/server model*, No publication date. Accessed: 2023-10-27, 2023. dirección: <https://www.ibm.com/docs/en/zos/2.5.0?topic=applications-clientserver-model>.
- [2] Pallets Projects, *Flask Documentation (Stable)*, No publication date. Accessed: 2026-01-03, 2026. dirección: <https://flask.palletsprojects.com/en/stable/>.
- [3] Flask-SocketIO, *Flask-SocketIO Documentation*, No publication date. Accessed: 2026-01-03, 2026. dirección: <https://flask-socketio.readthedocs.io/en/latest/>.