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

En el presente informe técnico se detallan los objetivos y el desarrollo del proyecto KubiBot, un robot móvil potenciado con IA, diseñado como un prototipo de robot de compañía, y como exploración de las capacidades que puede ofrecer la inteligencia artificial en la robótica.

Se describen las tecnologías utilizadas, incluyendo hardware y software, así como los desafíos enfrentados durante el desarrollo. Además, se presentan los resultados obtenidos y se discuten posibles mejoras futuras.

Índice general

Índice de figuras

Introducción

La unión entre la robótica y la inteligencia artificial (IA) ha estado en auge en los últimos años. Si bien no es algo nuevo, la integración de estas dos disciplinas ha cobrado una relevancia significativa, sobre todo con el avance exponencial que ha tenido la IA en el campo generativo y del procesamiento del lenguaje. Esto ha vuelto más accesible la incorporación de la IA en sistemas robóticos, tanto económica como técnicamente, permitiendo que estos sean más inteligentes y capaces de interactuar de manera más natural con los humanos.

En el presente proyecto, llamado "KubiBot", se busca desarrollar un prototipo de robot de compañía móvil potenciado por IA, que sea capaz de interactuar conversacionalmente con los usuarios, respondiendo preguntas, proporcionando información, etc. Se pretende utilizar tecnologías modernas, pero relativamente accesibles para el Departamento de Ingeniería en Computación e Informática de la Universidad de Magallanes, con el fin de explorar las capacidades que puede ofrecer la inteligencia artificial en la robótica.

En este informe técnico se detallarán los objetivos del proyecto, el marco teórico, las tecnologías utilizadas, el proceso de desarrollo y los resultados obtenidos. Además, se discutirán posibles mejoras y futuras líneas de trabajo para continuar explorando esta unión entre la robótica y la inteligencia artificial.

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 el diseño del chasis, la integración de motores y la programación de un sistema de evitación de obstáculos, utilizando un microcontrolador **Arduino Uno**.

2. Implementar inteligencia artificial en la plataforma **Raspberry Pi**, abarcando la configuración del sistema, la creación de una API como intermediaria entre el host del LLM y la **Raspberry Pi**, y la integración de modelos de reconocimiento de voz (speech-to-text).

3. Integrar periféricos de comunicación (entrada y salida) para la interacción usuario-robot, incluyendo la instalación de micrófono y altavoz.

4. Ensamblar los distintos módulos de hardware del prototipo, procurando la coordinación entre **Arduino** y **Raspberry Pi**.

5. Validar la funcionalidad completa del robot mediante pruebas de software y hardware.

Marco teórico

3.1. Robótica móvil

La robótica móvil es la disciplina encargada del estudio y desarrollo de robots que tienen la capacidad física de desplazarse en su entorno, a diferencia de los robots industriales que permanecen anclados a una base fija. El objetivo principal de estos sistemas es realizar tareas en espacios sin intervención humana constante.

3.1.1. Sistemas de locomoción

Para moverse, los robots requieren un mecanismo físico que interactúe con el suelo. La elección de este mecanismo depende del tipo de terreno donde operará el robot:

- **Robots con ruedas:** Son los más populares debido a su simplicidad mecánica, bajo consumo de energía y facilidad de control. Son ideales para superficies lisas o pavimentadas.

3.1.2. Sensores y autonomía

Para que un robot móvil sea "inteligente" o autónomo, necesita percibir el mundo que lo rodea. Esto se logra mediante sensores que actúan como los "sentidos" del robot:

- **Sensores internos:** Permiten al robot conocer su propio estado, como la velocidad de las ruedas o el nivel de batería.
- **Sensores externos:** Recopilan datos del entorno. Los más comunes incluyen cámaras para visión artificial y sensores ultrasónicos para detectar la distancia a objetos cercanos y evitar choques.

Gracias a esta información sensorial, el sistema de control (el "cerebro" del robot) puede tomar decisiones en tiempo real, como detenerse ante un obstáculo o corregir su rumbo hacia un destino.

3.2. Inteligencia artificial

La inteligencia artificial (IA) es un campo de la informática que se centra en la creación de sistemas y programas capaces de realizar tareas que normalmente requieren inteligencia humana. Estas tareas incluyen el aprendizaje, el razonamiento, la percepción, la toma de decisiones y la comprensión del lenguaje natural. La IA puede clasificarse en dos categorías principales: IA débil, que está diseñada para realizar tareas específicas, e IA fuerte, que tiene capacidades cognitivas similares a las humanas.

3.2.1. Modelos de lenguaje (LLM)

El "cerebro" detrás de un asistente moderno es lo que se conoce como un Gran Modelo de Lenguaje (LLM). Estos sistemas funcionan prediciendo palabras y, gracias a su entrenamiento masivo, pueden entender una pregunta en lenguaje natural y construir una respuesta coherente palabra por palabra, manteniendo el contexto de una charla, lo que les da la apariencia de ser conversadores inteligentes.

3.3. 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 prácticamente idéntico a C/C++.

A continuación se presenta un diagrama del **Arduino UNO**, destacando sus principales componentes y conexiones:

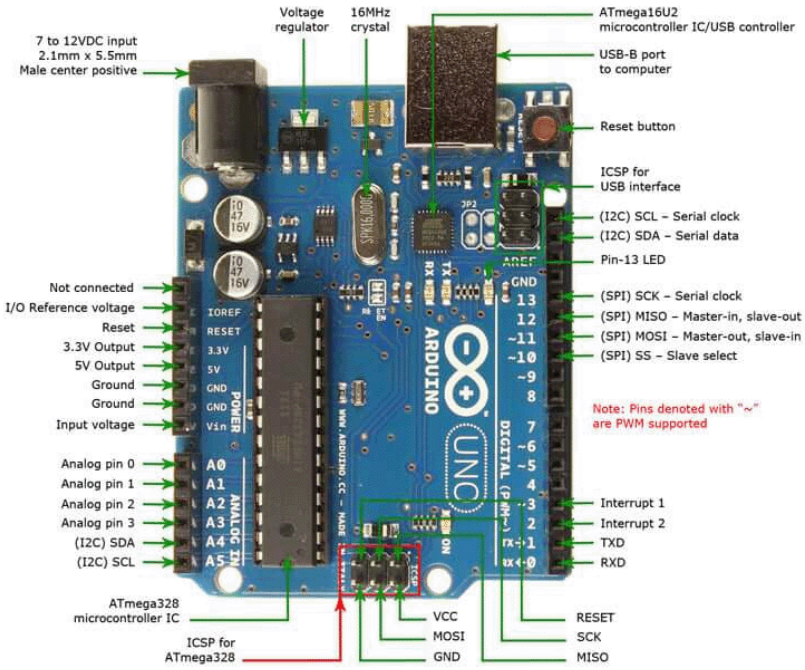

Figura 3.1: Diagrama del Arduino UNO

3.4. Raspberry Pi 5

El **Raspberry Pi 5** es la última versión de las microcomputadoras **Raspberry Pi**, la cual está diseñada para ofrecer tanto rendimiento como versatilidad. No debe ser confundida con un microcontrolador, ya que en realidad es una computadora completa, solo que en un formato muy compacto.

El **Raspberry Pi 5** cuenta con un procesador ARM Cortex-A76 de cuatro núcleos a 1.8 GHz, 4 GB u 8 GB de RAM, conectividad Wi-Fi y Bluetooth integradas, puertos USB 3.0 y USB 2.0, un puerto Ethernet Gigabit, salida HDMI dual para soporte de pantallas 4K y una ranura para tarjeta microSD para almacenamiento.

Este suele ser configurado con un sistema operativo basado en Linux, siendo el más común **Raspberry Pi OS**, aunque de igual forma puede ejecutar otros sistemas operativos siempre y cuando sean compatibles con su arquitectura ARM. Para el acceso a ella, se puede utilizar un monitor, teclado y ratón conectados directamente, o bien acceder de forma remota mediante SSH o escritorio remoto.

A continuación se presenta una imagen de la **Raspberry Pi 5**, destacando sus principales componentes y conexiones:

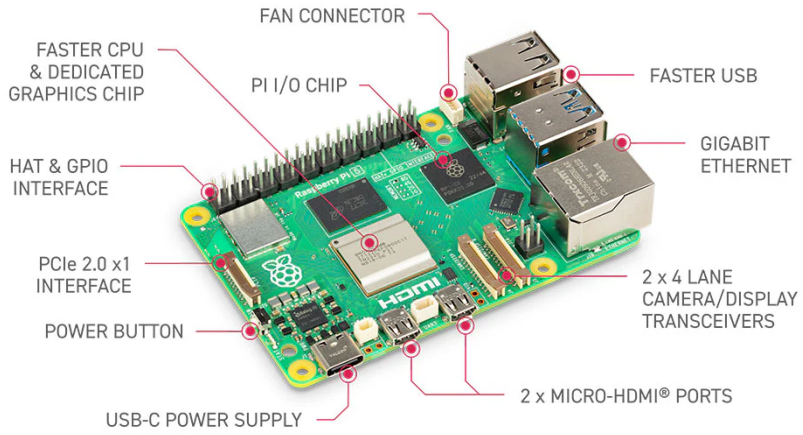

Figura 3.2: Imagen de la **Raspberry Pi 5**

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 desde 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 adaptarse obligatoriamente a un sistema preexistente.

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 definida la idea general, 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 conectarlos una vez que 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. **Comunicación:** Módulo en donde residiría todo lo relacionado con la comunicación con el usuario. Se determinó inicialmente levantar localmente una LLM en una **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 determinada la funcionalidad, 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 vez, accesible. Se optó entonces por crear o buscar un modelo para imprimir en 3D, lo que implicó que el diseño 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 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 entorno 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 objetos 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; a continuación se presenta el diagrama de este en la figura 4.1.

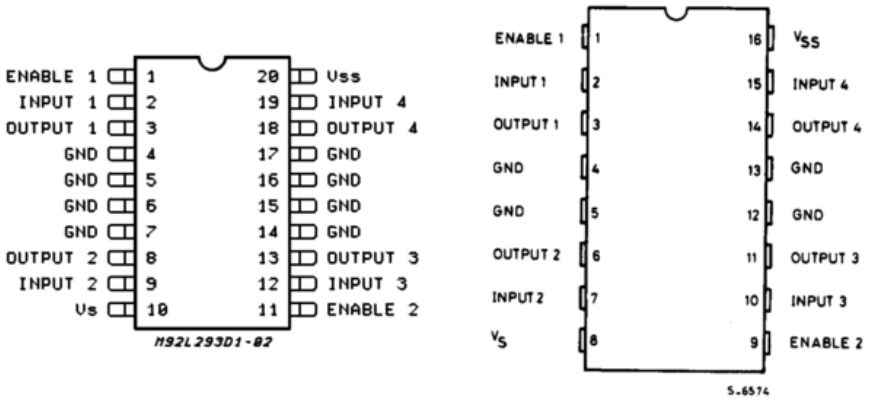

Figura 4.1: Puente H L293D

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.

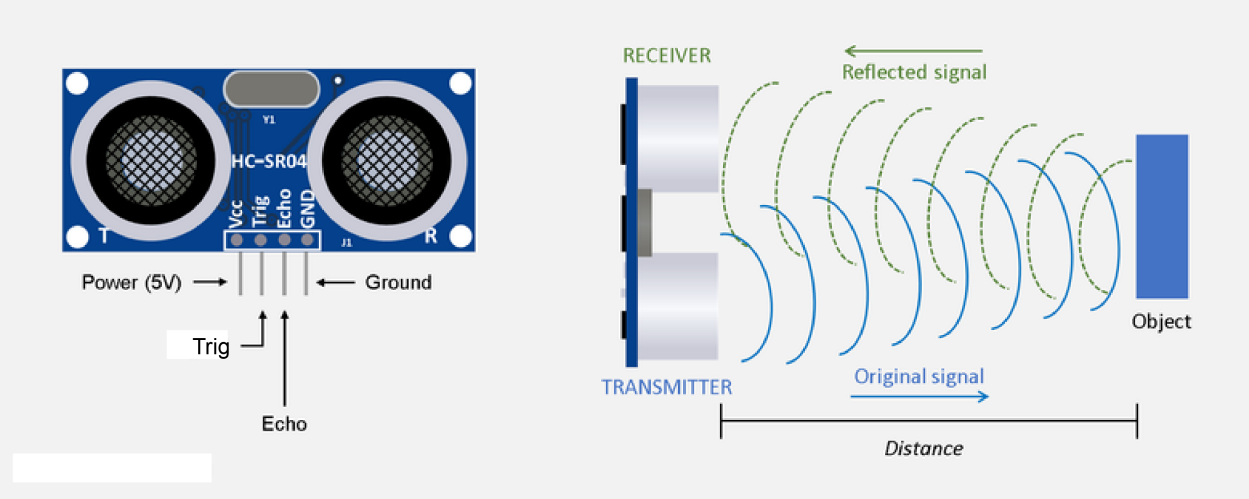

Figura 4.2: Diagrama del sensor ultrasónico HC-SR04

Los problemas comenzaron a surgir durante las primeras pruebas. El primero 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 cómo el prototipo giraría al detectar un obstáculo. Como el chasis ya se encontraba impreso y era complicado modificarlo para añadir un mecanismo con un servo, se decidió realizar el giro 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. Por esto se decidió implementar en el sistema de detección de obstáculos un servomotor analógico conectado al **Arduino**, al cual se encontraría ensamblado el sensor ultrasónico.

De esta forma, se ejecutaría el siguiente flujo:

```
while(true):
    if (distancia_obstaculo < umbral de seguridad):
        1. Detener ambos motores.
        2. Retroceder hasta una distancia segura.
        3. Girar el servo a la izquierda y medir distancia.
        4. Girar el servo a la derecha y medir distancia.
        5. Comparar ambas distancias.
        6. Girar en la direccion con mayor distancia libre.
    else:
        Continuar moviendose 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, 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 portapilas de 6 pilas AA, entregando un total de 9V. Esto proporcionó no solo la potencia necesaria, sino también la posibilidad de cambiar las pilas fácilmente cuando se agoten.

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

dos motores adicionales, convirtiendo el sistema de locomoción 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 al investigar se descubrió que con este habría riesgo de sobrecalentamiento si se utilizaban los 4 motores simultáneamente. 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.

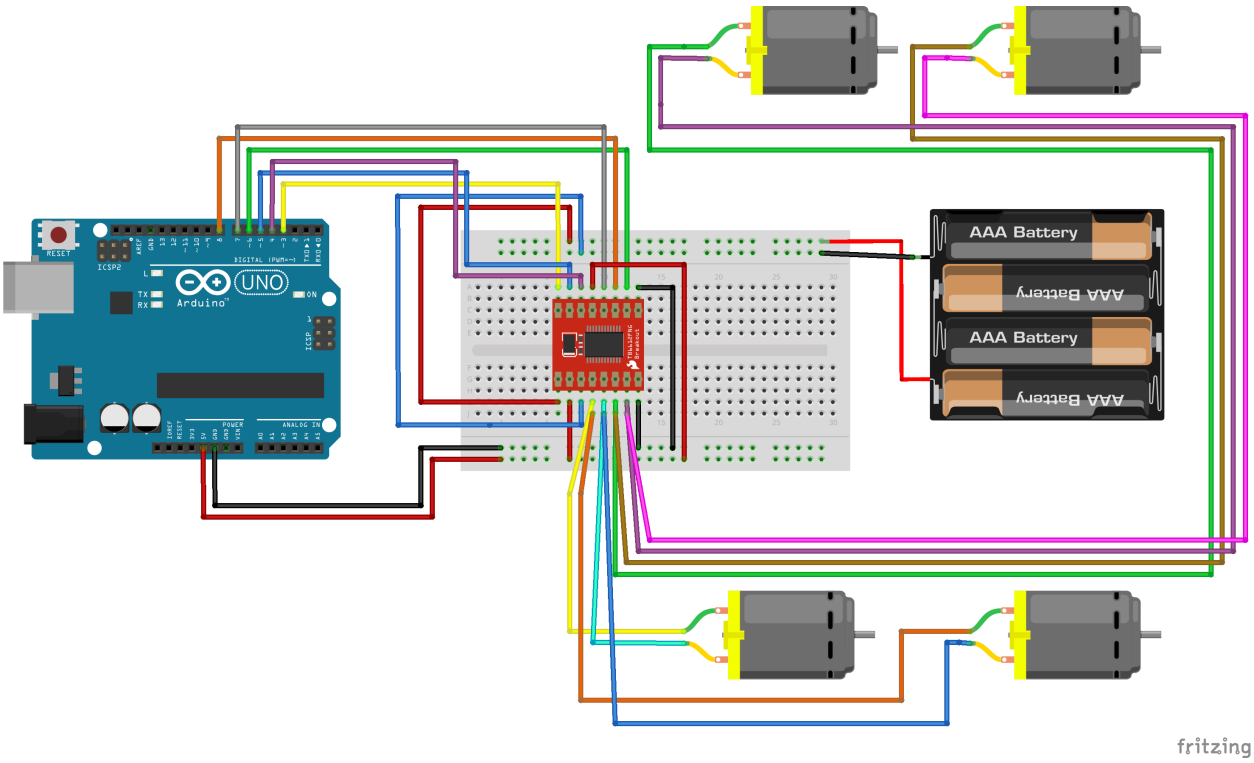

Figura 4.3: Conexión de motores al TB6612FNG

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 (utilizados en el primer prototipo) integrados.

Este *shield* resultó en un gran avance 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, cuenta con una entrada para una fuente de alimentación externa, que alimenta tanto a los motores como al **Arduino** y los demás componentes. Así, 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 reduciendo peso. A continuación se presenta el diagrama de conexión del *shield*:

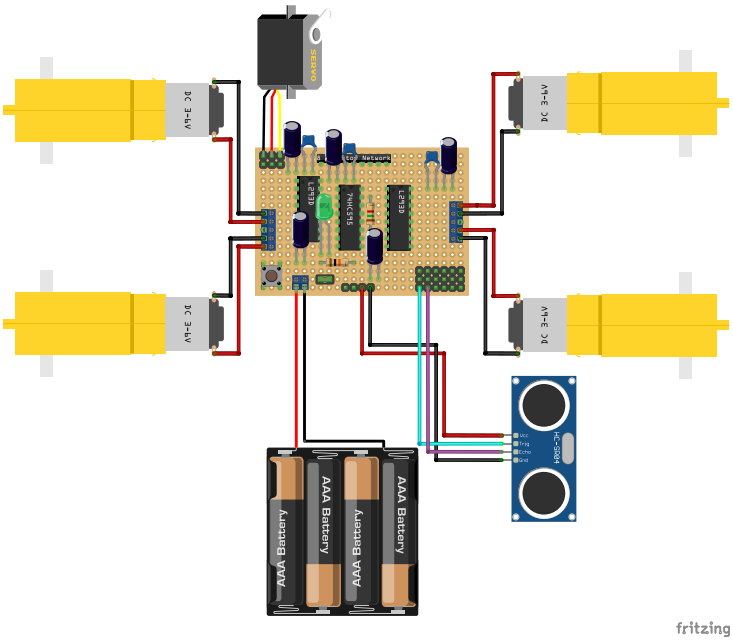

Figura 4.4: Conexiones del motor driver shield L293D

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.

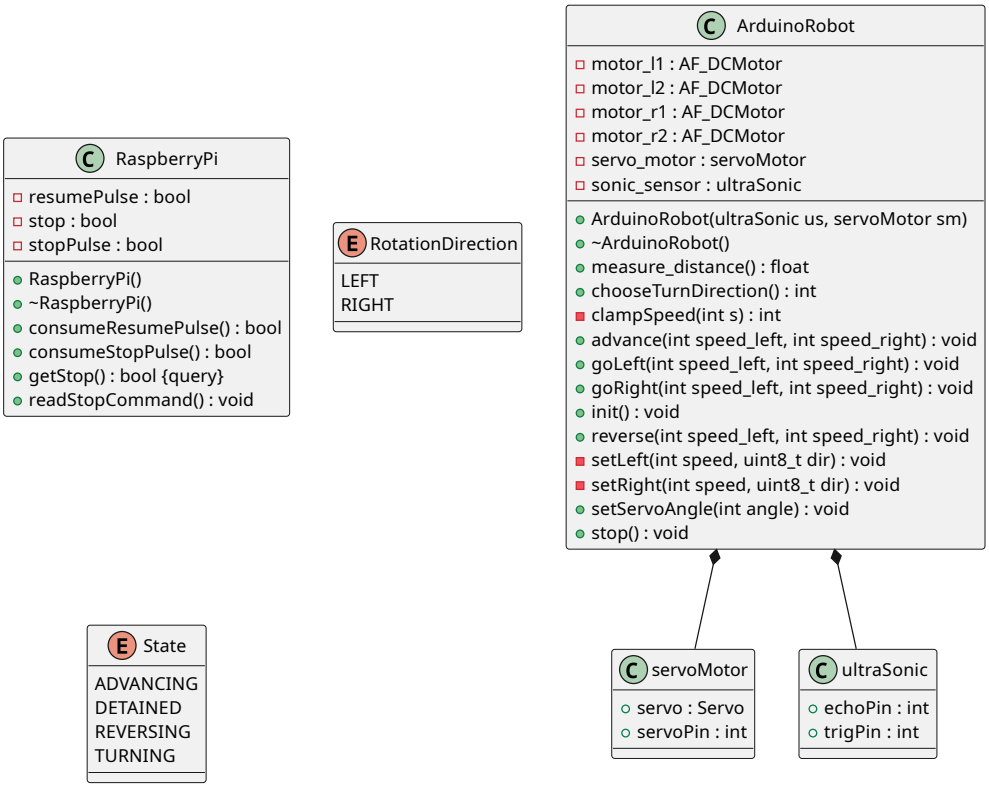

Figura 4.5: Diagrama de clases del módulo de movimiento

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 `ArduinoServo`. 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:

```
inicializar_robot()

while(true):
    distancia = medir_distancia_cm()

    if estado_robot == AVANZANDO:
        if distancia < UMBRAL_SEGURIDAD:
            estado_robot = RETROCEDIENDO
        else:
            avanzar()

    else if estado_robot == RETROCEDIENDO:
        if distancia < UMBRAL_SEGURIDAD:
            retroceder()
        else:
            // Mirar izquierda y derecha con el servo
            mover_servo_izquierda()
            dist_izq = medir_distancia_cm()

            mover_servo_derecha()
            dist_der = medir_distancia_cm()

            mover_servo_frente()

```

```
            if dist_izq > dist_der:
                direccion_giro = IZQUIERDA
            else:
                direccion_giro = DERECHA

            estado_robot = GIRANDO
            tiempo_inicio_giro = tiempo_actual_ms()

    else if estado_robot == GIRANDO:
        if tiempo_actual_ms() - tiempo_inicio_giro < TIEMPO_GIRO:
            girar(direccion_giro)
        else:
            estado_robot = AVANZANDO
```

Listado 4.2: Flujo principal de control en **Arduino**

4.3. Comunicación

Como se estableció previamente en el análisis del problema, se decidió utilizar una **Raspberry Pi** como el núcleo del módulo de comunicación.

En un principio, se pensó para alojar localmente un modelo de lenguaje, además de manejar la entrada y salida de audio y todo el posprocesamiento. Sin embargo, tras las primeras pruebas, se encontraron varios problemas y la arquitectura del sistema tuvo que replantearse.

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 por utilizar un LLM alojado localmente, debido tanto a la limitación de recursos económicos para utilizar una **API** comercial, como a la intención de realizar un sistema local donde no se exponga la información del cliente. Se seleccionó la herramienta *Ollama*, que permite realizar esto.

Sin embargo, al intentar utilizar diversos modelos, se descubrió que la **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 la **Raspberry Pi** no como host del modelo de lenguaje, sino como un intermediario entre el usuario y un servidor externo que alojaría el modelo. De esta forma, la **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-servidor[1]. La **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 la **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.

`raspberry.py` es el script principal que corre en la **Raspberry 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`.

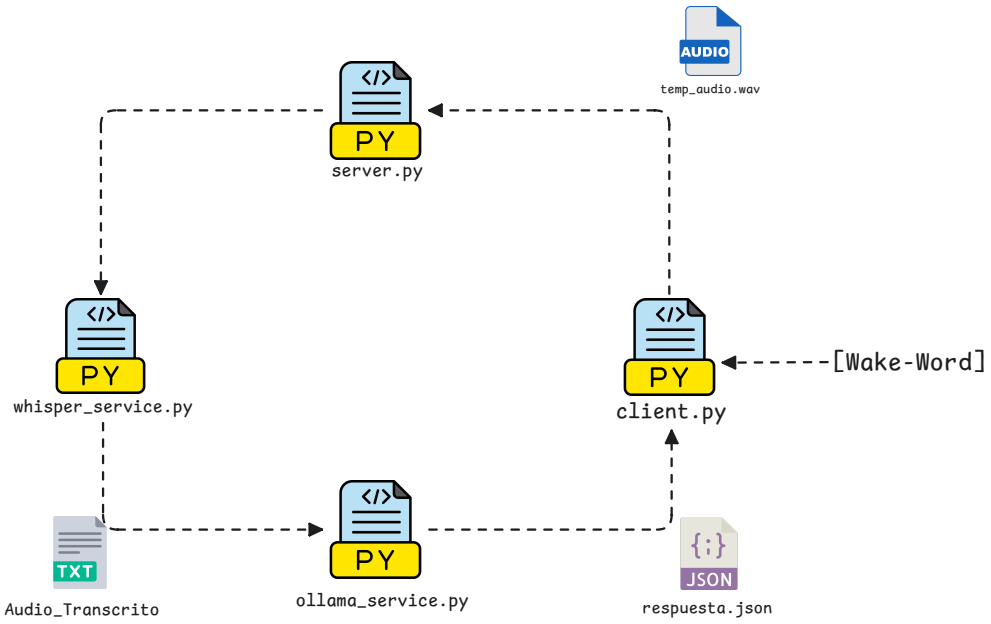

Figura 4.6: Diagrama de comunicación servidor-cliente

[1]Modelo 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 WebSockets. 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. Además de lo anterior, el servidor permite manejar más 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 la 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 grabación, se procesa el audio para obtener la transcripción 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 reproducción.

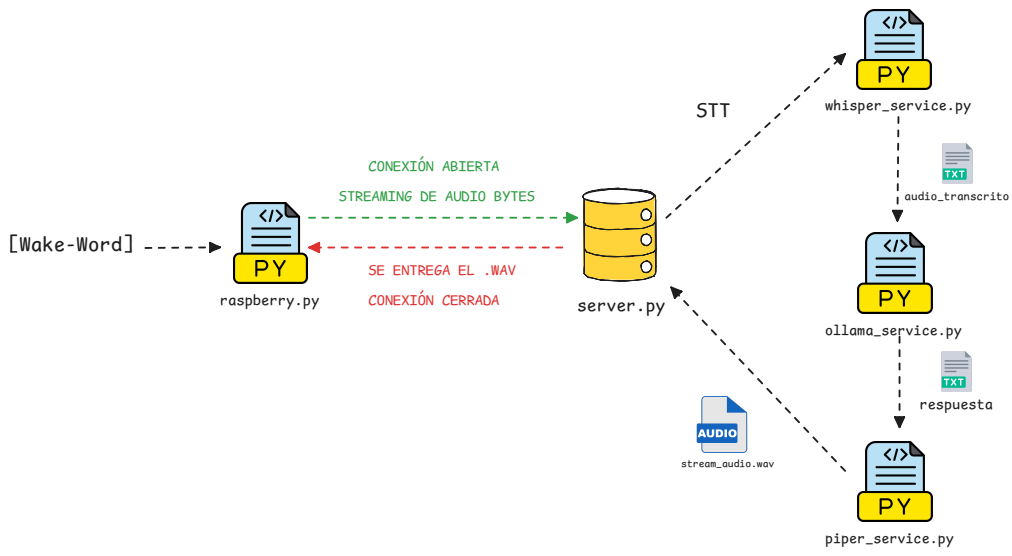

Figura 4.7: Diagrama de comunicación final servidor-cliente

4.3.4. Implementación del cliente en Raspberry Pi

El cliente, implementado en el script `raspberry.py`, se ejecuta en la **Raspberry Pi** y es responsable de la interacción directa con el usuario. Su función principal es detectar la **wake word** (en este caso, «**Hey Bot**»), grabar la voz del usuario, enviarla al servidor y reproducir el audio de respuesta. Esto se logró mediante la implementación de la librería y herramienta `Porcupine`, especializada en la creación de modelos ligeros para la detección de palabras clave. El cliente inicializa `Porcupine` con una clave de acceso 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[2] 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, la **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.

[2]Formato de audio digital sin compresión donde la señal se representa como una secuencia de muestras cuantizadas.

4.4. Comunicación Raspberry-Arduino

La comunicación entre la **Raspberry Pi** y el **Arduino** consiste en enviar una señal desde la **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 en la diferencia de voltajes entre ambos dispositivos. La **Raspberry Pi** opera a 3.3V en sus pines GPIO, mientras que el **Arduino** funciona a 5V. Esta diferencia puede causar daños permanentes en la **Raspberry Pi** si se conecta directamente un pin de salida del **Arduino** a un pin de entrada de la Raspberry Pi. Para resolver esto se consideraron dos opciones principales:

- **Circuito con relé:** Utilizar un relé para aislar eléctricamente ambos dispositivos. El **Arduino** podría activar el relé para enviar una señal a la **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 a la **Raspberry Pi** a través de la conexión USB, y la **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 relé. Sin embargo, tras probar el circuito, no se logró una comunicación estable entre ambos dispositivos. Por esto, finalmente 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.

Para implementar la comunicación serial, se utilizó la librería `pyserial` en la **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:

```
comandoSerial = leer_comando_serial()

if comandoSerial == "S":
    cambiar_estado_robot("DETENIDO")
elif comandoSerial == "R" && estadoRobot == "DETENIDO":
    cambiar_estado_robot("MOVIENDOSE")

switch(estadoRobot):
    case "MOVIENDOSE":
        mover_adelante()
    case "DETENIDO":
        detener_motores()
    case "GIRANDO":
        ejecutar_giro()
```

Listado 4.3: Recepción de comandos seriales en **Arduino**

```
if robot_escuchando || robot_hablando:
    enviar_comando_serial("S") // "S" para detener
    esperar_handshake() // Esperar confirmacion de detencion por parte del
    Arduino, en este caso una "K"
else if (tiempoPasado > tiempoEsperaDespuesDeHablar):
    enviar_comando_serial("R") // "R" para reanudar
```

Listado 4.4: Envío de comandos seriales en Raspberry

En la implementación final, se definieron dos comandos principales enviados desde la **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), la **Raspberry Pi** envía `S` para forzar el estado `DETAINED`, lo que detiene inmediatamente los motores. Para asegurar que la conexión sea exitosa, se envía una señal de confirmación desde el **Arduino**; esta señal es `K`. Una vez finalizada la respuesta de voz, se envía `R`, permitiendo al **Arduino** volver al estado `ADVANCING` y retomar la navegación autónoma.

4.4.1. Alimentación Raspberry

Un problema adicional que surgió durante la implementación de la comunicación fue la alimentación de la **Raspberry Pi**. Inicialmente se intentó alimentar la **Raspberry Pi** directamente desde un *power bank* de 3A/5V conectado al puerto USB. Sin embargo, se descubrió que el *power bank* no suministraba suficiente corriente, lo que provocaba que la **Raspberry Pi** se reiniciara constantemente; por esto, como se observó en la sección anterior, el sistema quedó tan dependiente del servidor.

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 ilustrado por Iván Mansilla

Con esta idea en mente, se procedió a diseñar el modelo 3D de la carcasa con la ayuda de Nicolás Poblete [4], quien se encargó de la modelación y la impresión de las piezas. Este diseño fue realizado de tal forma que los componentes internos tuviesen el espacio necesario, además de que sus piezas fuesen modulares y desmontables; es decir, que cada pieza pudiese ser removida individualmente para acceder a los componentes internos con facilidad. A continuación se muestran algunas imágenes del modelo 3D:

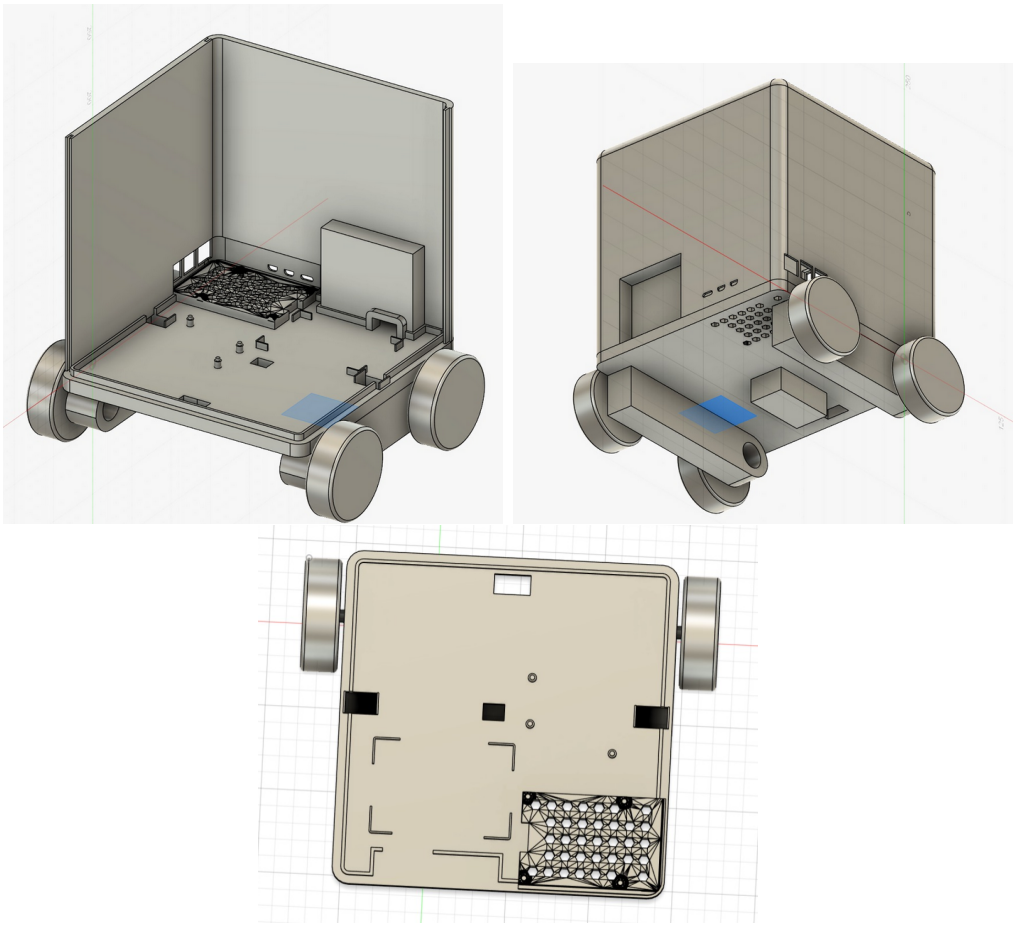

Figura 4.9: Modelo 3D de la carcasa diseñado por Nicolás Poblete

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

4.5.1. Skid steering

En cuanto al diseño del chasis, se optó por un sistema de *skid steering*, que consiste en utilizar ruedas fijas y controlar el giro del robot variando la velocidad de las ruedas en cada lado, similar al sistema que utilizan las máquinas de carga convencionales. Esto permite giros sobre el mismo eje y una mayor estabilidad en superficies irregulares. Este sistema fue elegido para no depender de mecanismos adicionales como un servo para el giro, lo que simplificó el diseño y redujo el peso del prototipo. A continuación se presenta un diagrama ilustrativo del sistema[5] de *skid steering* utilizado:

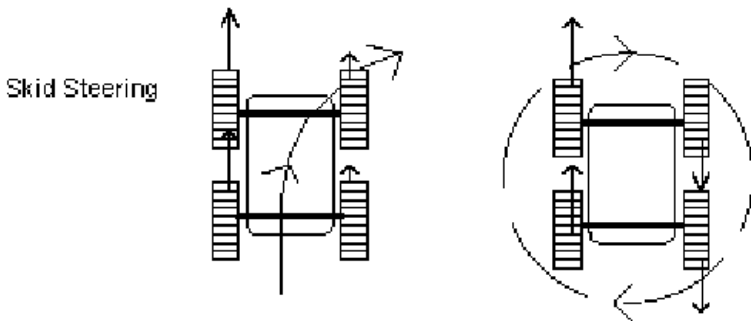

Figura 4.10: Diagrama ilustrativo del sistema de *skid steering*

Este diseño permitió un control preciso del movimiento del robot, soportando el peso de los componentes internos gracias a sus cuatro ruedas motrices y facilitando la navegación en el entorno doméstico para el cual fue diseñado Kubibot.

Conclusión

A raiz de todo lo desarrollado en este proyecto, se puede concluir que efectivamente se cumplio el objetivo de diseñar y desarrollar un prototipo de robot de compañia movil potenciado por inteligencia artificial, capaz de interactuar de forma autónoma y asistir al usuario dentro de un entorno doméstico. Si bien, el prototipo tiene limitaciones en cuanto a su movilidad y capacidades de interacción, se logró demostrar la viabilidad de integrar tecnologías accesibles como el **Arduino UNO** y el **Raspberry Pi 5** con modelos de lenguaje avanzados para crear un sistema funcional. La implementación de un modelo de lenguaje conversacional permitió que el robot respondiera preguntas y mantuviera conversaciones básicas/

Sin embargo, existen evidentes areas de mejoras, como lo pueden ser la optimizacion del sistema de deteccion y evitacion de obstaculos, una mejora en el modelo utilizado (entrenar un modelo especializado para el robot) y la incorporacion de mas sensores para una mejor percepcion del entorno. Ademas, se recomienda explorar la posibilidad de integrar capacidades adicionales, como una pantalla para mostrar inforamcion visual como emociones o respuestas graficas, un disenho mas robusto y estetico para el chasis del robot y por ultimo una mejora en la suministracion de poder para el robot.

En resumen, el proyecto KubiBot demostro el potencial de la combinacion entre robótica e inteligencia artificial, destacando las diversas posibilidades de mejora y desarrollo en futuros proyectos.

Bibliografía

[1] IBM, *Client/server model*, 2023. dirección: https://www.ibm.com/docs/en/zos/2.5.0?topic=applications-clientserver-model.

[2] Pallets Projects, *Flask Documentation (Stable)*, 2026. dirección: https://flask.palletsprojects.com/en/stable/.

[3] Flask-SocketIO, *Flask-SocketIO Documentation*, 2026. dirección: https://flask-socketio.readthedocs.io/en/latest/.

[4] Nicolas Poblete, *maker3d.puq*, 2026. dirección: https://www.instagram.com/maker3d.puq.

[5] G. Shuang, N. C. Cheung, K. W. E. Cheng, D. Lei y L. Xiaozhong, «Skid Steering in 4-Wheel-Drive Electric Vehicle,» *2007 7th International Conference on Power Electronics and Drive Systems*, págs. 1548-1553, 2007. dirección: https://api.semanticscholar.org/CorpusID:2105235.