

Universidad Loyola

FACULTAD DE INGENIERÍA

¿SIRVE UN LLM PARA CONTROLAR UN ROBOT MÓVIL?

Máster en Inteligencia Artificial

Autor:

Jerónimo Manuel Jiménez Mateos

Director:

José María Manzano Crespo



Agradecimientos

Quiero expresar mi más sincero agradecimiento a todas las personas que me han ayudado y apoyado durante la realización de mis estudios de Máster.

A mi tutor José María Manzano, por aceptar el Trabajo de Fin de Máster que le propuse y ayudarme a desarrollarlo. Su experiencia ha sido fundamental a la hora de indicarme qué camino seguir y qué tareas priorizar. Me ha sido muy fácil trabajar con él porque siempre ha estado accesible—sin importar la fecha ni la hora—respondiendo todas mis dudas y corrigiendo mis errores.

A mis compañeros y profesores del Máster, de los cuales he aprendido una gran cantidad de cosas, y junto con los que he pasado muy buenos momentos, en clase y fuera de ella. Ha sido un curso increíble, en el que considero que en general hemos adquirido una gran cantidad de información muy valiosa y que nos va a ser de gran utilidad en nuestra vida profesional.

Por último, pero no menos importante, a mi familia y amigos, que han estado apoyándome desde el principio cuando les conté que quería estudiar un Máster en Inteligencia Artificial, y que me siguen apoyando día a día en todas las decisiones que tomo. Sin ellos, no habría sido posible.

Acknowledgements

I would like to express my gratitude to all the people who have helped and supported me throughout my Master's studies.

To my advisor, José María Manzano, for accepting the Master's Thesis I proposed and guiding me through its development. His experience has been essential in helping me choose the right path and prioritize the most important tasks. Working with him has been very easy, as he has always been available—regardless of the date or time—willing to answer my questions and correct my mistakes.

To my classmates and professors in the Master's program, from whom I have learned a great deal and with whom I have shared many memorable moments, both inside and outside the classroom. It has been an incredible year, in which I believe we have all acquired a wealth of valuable knowledge that will be highly useful in our professional careers.

Last but not least, to my family and friends, who have supported me from the very beginning when I told them I wanted to pursue a Master's degree in Artificial Intelligence, and who continue to support me every day in all the decisions I make. Without them, none of this would have been possible.

Resumen

En el proyecto que se presenta a continuación se propone el uso de un modelo de lenguaje como algoritmo de control de un robot móvil. Para ello, se construye un robot y se diseña un método para que sea un LLM el que lo controle.

Posteriormente, se realizan una serie de experimentos cuyo fin es determinar la viabilidad de usar un LLM para esta función.

Por último, se añaden las conclusiones extraídas de los experimentos y se hace un breve inciso acerca de posibles líneas de investigación futuras relacionadas con la temática.

En este sentido, se trata de un proyecto que abarca por un lado la parte robótica de diseñar y construir el robot, hallar el modelo cinemático del mismo, aplicar técnicas clásicas de visión por computador, etc. y por otro conocimientos de Inteligencia Artificial, usando redes convolucionales y sobre todo LLMs.

Abstract

The following project explores the use of a Large Language Model as the control algorithm for a mobile robot. To achieve this, a physical robot is built and a method is developed to enable an LLM to take control of it.

A series of experiments are then conducted with the goal of assessing the feasibility of employing an LLM for this task.

Finally, the conclusions drawn from these experiments are presented, along with a brief discussion of potential future research directions related to this subject.

In this context, the project bridges two main areas: on one side, the robotics work involving the design and construction of the robot, the derivation of its kinematic model, and the application of classical computer vision techniques; on the other, the artificial intelligence domain, leveraging convolutional neural networks and, most notably, Large Language Models.

Índice general

1. Introducción	11
1.1. Descripción general	11
1.1.1. Robótica	12
1.1.2. Inteligencia Artificial	13
1.2. Justificación y motivación	13
1.2.1. Estado del arte	14
1.3. Estructura de la memoria	15
2. Hardware	16
2.1. Componentes	17
2.1.1. Raspberry Pi	18
2.1.2. Motores	18
2.1.3. Puente H	19
2.1.4. Encoders	20
2.1.5. Acelerómetro y giróscopo	20
2.1.6. Ultrasonido	21
2.1.7. Cámara	21
2.2. Cinemática	23
2.2.1. Introducción a la cinemática del robot	23
2.2.2. Ecuaciones del modelo cinemático	24
3. Software	26
3.1. Introducción al software utilizado	26
3.2. Calibración de una cámara estéreo	28
3.3. Toma y procesamiento de imágenes	30
3.3.1. Primera estrategia	30
3.3.2. Segunda estrategia	33

3.4.	WebSocket	34
3.4.1.	Envío de imágenes por WebSocket	35
3.5.	Introducción al esquema de control	36
3.6.	PID	37
3.6.1.	Breve introducción al control PID	37
3.6.2.	Implementación de esquemas de control tipo PID en el proyecto . .	38
3.7.	LLM	40
3.7.1.	Características de los LLM	40
3.7.2.	Modelos disponibles	41
3.7.3.	Prompt para el modelo	42
3.7.4.	Configuración de los LLM	47
4.	Análisis de resultados	49
4.1.	Comparación de ambas estrategias	49
4.2.	Análisis de tiempos de inferencia	49
4.3.	Análisis de tokens	50
4.4.	Análisis de consistencia	51
4.5.	Búsqueda de objetos	51
5.	Conclusiones	53
5.1.	Ventajas de utilizar un LLM como controlador de un robot	53
5.2.	Ventajas y desventajas de utilizar modelos en la nube frente a modelos en local	54
5.3.	Limitaciones y posibles contribuciones al proyecto	54
5.3.1.	Limitaciones de la primera estrategia	54
5.3.2.	Otras limitaciones del proyecto	55
5.3.3.	Trabajos futuros	56

Índice de figuras

1.1. Esquema del funcionamiento del robot	12
2.1. Foto del robot	17
2.2. Esquema de conexiones del robot	17
2.3. Diagrama de pines de la Raspberry Pi (Pi (2025))	18
2.4. Ejemplo de señal PWM que modula la función seno	19
2.5. Esquema básico del puente H	19
2.6. Representación de un encoder óptico (Verschuren (2006))	20
2.7. Imagen de la cámara GXIVISION-LSM22100	21
2.8. Simplificación del modelo cinemático del robot	23
2.9. Modelo cinemático del robot	23
3.1. Captura del software LMStudio	27
3.2. Ejemplo de patrón típico de calibración	29
3.3. Esquema de la primera estrategia	31
3.4. Ejemplo de un mapa de disparidad (Baeldung (2021))	32
3.5. Recorte del objeto en el mapa de profundidad	33
3.6. Esquema del funcionamiento del protocolo WebSocket basado en Wellons (2013)	35
3.7. Esquema del control del robot	36
3.8. Diagrama del robot con variables a controlar	38
3.9. Representación de texto dividido en tokens	40
5.1. Ejemplo de mapa de profundidad con profundidad estimada incorrecta . .	55

Índice de tablas

3.1. Comparación de características entre LLaVA 1.5 7B, ChatGPT-4o mini y Gemma 3 4B	41
4.1. Comparación de los tiempos de inferencia de los modelos	50
4.2. Comparación de la consistencia en las respuestas de los modelos	51

1. Introducción

En este capítulo se presentará el proyecto realizado, explicando brevemente en qué consiste y aportando un contexto que servirá como base sobre la que parten los capítulos posteriores.

1.1. Descripción general

Este Trabajo de Fin de Máster tiene como objetivo investigar la utilidad de los recientes modelos grandes de lenguaje (LLM, por sus siglas en inglés) para controlar la navegación de robots móviles terrestres, integrando comunicaciones, sensores, software y visión artificial.

Para ello, se ha construido un robot con una placa Raspberry Pi que está conectada por protocolo WebSocket con un ordenador portátil en el que se ejecuta un LLM. Así, el robot envía al LLM imágenes y el LLM determina qué debe hacer el robot. Concretamente, en este proyecto la función final del robot es **encontrar objetos en una habitación**. Esta es una de las muchas aplicaciones que se le puede dar a la combinación de los LLM y los robots móviles.

Para evaluar las prestaciones, se ha realizado una serie de experimentos con distintos modelos de lenguaje y siguiendo diversas estrategias. Esta experimentación permite seleccionar las características y parámetros que dan mejores resultados para finalmente implementarlos en conjunto.

A continuación se presenta un esquema simplificado del proyecto y se procede a explicar, a grandes rasgos, cómo actúa cada una de las partes implicadas.

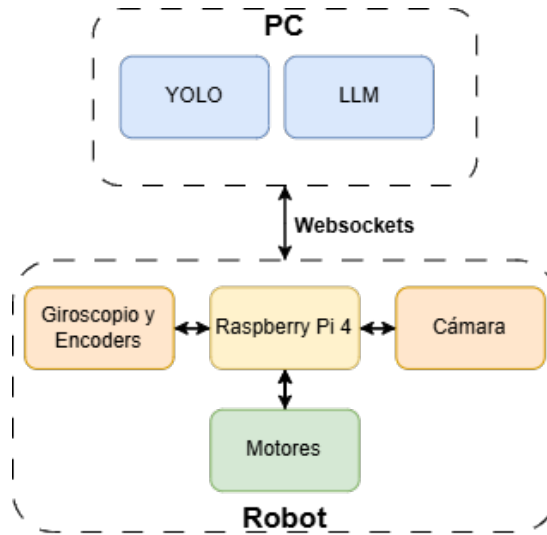


Figura 1.1: Esquema del funcionamiento del robot

A groso modo, el trabajo se puede dividir en dos grandes bloques: robótica e inteligencia artificial, estando estos estrechamente relacionados.

1.1.1. Robótica

Esta parte comprende desde la construcción del robot físico hasta la creación de los bloques de código que se encargan de manejar los sensores y actuadores del mismo.

Aunque más tarde en el Capítulo 2 se entrará en detalle acerca del robot, es un buen momento para introducirlo y presentar sus funcionalidades. Se trata de un robot móvil accionado por 4 motores conectados a 4 ruedas que actuará como plataforma de pruebas para las diferentes estrategias abordadas a lo largo del proyecto. El robot está equipado con una cámara estereoscópica en su parte frontal, lo que le permite obtener información tridimensional del entorno. Además, cuenta con un acelerómetro y un giróscopo, y encoders en las ruedas que proporcionan datos sobre su orientación y desplazamiento, respectivamente. Gracias a estos sensores, el robot puede estimar su posición relativa. Mediante un sensor de ultrasonido y la propia cámara, tiene la capacidad de detectar obstáculos en su trayectoria. En conjunto, estas características hacen del robot una base ideal para implementar y evaluar técnicas de inteligencia artificial orientadas a la navegación autónoma.

El sistema de procesamiento del robot es una Raspberry Pi 4, sobre la que se ejecuta el código principal, desarrollado en Python, y la cual se comunica a su vez con un ordenador portátil en el que se realizan las tareas más pesadas a nivel computacional. Estas son la detección del objeto en la imagen y la ejecución del modelo. El protocolo de comunicación elegido es WebSocket, que se caracteriza por su alta latencia y fiabilidad (Fette and Melnikov (2011)). Esto se analiza más en detalle en la Sección 3.1.

1.1.2. Inteligencia Artificial

El bloque de inteligencia artificial está ligado al planificador del robot. En el proyecto se plantea la pregunta: ¿sirve un modelo de lenguaje para controlar un robot? El LLM actúa como algoritmo de control, interpretando las imágenes capturadas y traduciéndolas en acciones concretas para el robot. Este enfoque permite explorar hasta qué punto un modelo multimodal, entrenado para comprender tanto lenguaje natural como información visual, puede asumir un rol central en la toma de decisiones de un sistema robótico.

A diferencia de las arquitecturas tradicionales, que separan claramente los módulos de percepción, razonamiento y control, este sistema unifica todos estos procesos dentro de un único modelo. El LLM multimodal recibe como entrada imágenes del entorno y, en algunos casos, instrucciones textuales complementarias, y genera como salida comandos específicos que el robot ejecuta (por ejemplo, avanzar, girar, detenerse o explorar).

En la Sección 3.5 se presenta el esquema de control del robot, el cual se trata de un sistema en cascada en el que un primer control formado por el LLM traduce las imágenes en instrucciones para el robot y posteriormente un control PID tradicional las traduce en comandos a los motores. Esta división jerárquica permite separar el razonamiento estratégico, más lento y complejo, del control físico de bajo nivel, que necesariamente debe ser más rápido.

Para ello, se utiliza un LLM con capacidad de visión que, dado un prompt diseñado cuidadosamente, genera una respuesta estructurada en un formato fijo (por ejemplo: **EXPLORA**). De este modo, se garantiza que la salida sea fácilmente interpretable por el sistema robótico.

El diseño del prompt es clave en este proceso, puesto que el desempeño del modelo depende en gran medida del prompt proporcionado. Para hacerlo lo más adecuado posible, se aplican técnicas de ingeniería de prompting como asignar un rol al modelo, definir el contexto en el que se realiza la tarea y dar ejemplos del funcionamiento (*few-shot prompting*).

Además, se han evaluado una serie de modelos multimodales ligeros que comprenden LLaVA 1.5, GPT-4o-mini y Gemma 3.

Desde el punto de vista del robot, se puede entender el LLM como una “caja negra” a la que se le pasa una cierta información y esta devuelve otra información, como la acción que debe tomar el robot a continuación. Dicho de otro modo, el modelo de lenguaje puede ser visto como un módulo independiente del robot. Esta *modularidad* facilita la escalabilidad y permite experimentar con distintos modelos sin afectar a la capa física de control, siendo así el LLM un núcleo “flexible” e inteligente encargado de la toma de decisiones del robot.

1.2. Justificación y motivación

El objetivo de esta sección es aportar un contexto teórico y presentar el estado actual de la investigación en el ámbito de la robótica, y más concretamente del uso de grandes modelos de lenguaje (LLMs, por sus siglas en inglés) como método de control de los robots móviles. Se incluyen las motivaciones, los fundamentos y las contribuciones más recientes en esta materia en el ámbito científico, justificando así la pertinencia del tema abordado.

El creciente uso de los LLMs en un diverso rango de sectores es muestra clara de su polivalencia. La multimodalidad es un objetivo atractivo para los creadores de modelos de lenguaje, que buscan conseguir una Inteligencia Artificial General (AGI).

Una de las tendencias de los últimos años es aplicar estos modelos en el área de la robótica, en constante crecimiento y con gran proyección. Esta nueva herramienta está permitiendo diseñar algoritmos de control más versátiles, adaptables a cambios (Wang et al. (2024)). En definitiva, permite dotar de una mayor autonomía a los sistemas robóticos, facilitando la interacción con el entorno y mejorando su capacidad de aprendizaje en situaciones no previstas durante el proceso de entrenamiento.

1.2.1. Estado del arte

Como se ha mencionado anteriormente, la integración de los LLMs en los robots ha emergido como un área de investigación activa en los últimos años, a pesar de ser una tecnología relativamente reciente. Se han adoptado diversos enfoques en los estudios acerca de este tema: desde la utilización de estos modelos para mejorar las capacidades de navegación autónoma (Shah et al. (2023)), hasta su aplicación a la hora de ayudar al robot a interpretar instrucciones en lenguaje natural (Brohan et al. (2023)). Este incremento en las capacidades viene además acompañado de una menor necesidad de reprogramar continuamente los robots, acelerando en gran medida el progreso.

Una de las ramas más prometedoras es la integración de los modelos de IA con capacidades de visión artificial, aprovechando los avances recientes en modelos multimodales que combinan lenguaje y visión y dotando a los robots de un “cerebro” artificial capaz de procesar lo que están viendo y tomar decisiones en base a ello (Barturen (2023), Zitkovich et al. (2023)). En la última década han surgido enfoques basados en modelos visión-lenguaje (VLM) que mapean imágenes a incrustaciones semánticas. Modelos como CLIP o BLIP procesan la imagen ambiental y generan vectores que representan la relación entre el entorno visual y la descripción textual del objeto buscado (Choi et al. (2024)).

En tareas de **búsqueda de objetos**, como es el caso de este proyecto, la tendencia es dejar que el LLM razone sobre la escena y guíe al robot hacia las áreas donde es más probable encontrar el objeto definido como “objetivo”. Por ejemplo, el marco GMM-Searcher construye un mapa topológico adaptativo y emplea mezclas gaussianas junto con un LLM para enfocar la exploración en zonas prometedoras (Zheng et al. (2025)). Esto optimiza el uso de memoria sin sacrificar la fidelidad del entorno. Además, los GMM permiten almacenar experiencias de búsqueda previas para mejorar el rendimiento en exploraciones futuras.

En síntesis, la literatura reciente muestra cómo la integración de LLMs con visión artificial eleva notablemente las capacidades de los robots móviles en interiores. Al interpretar escenas visuales mediante modelos de lenguaje, los robots pueden seguir instrucciones en lenguaje natural más complejas y adaptarse a objetos o situaciones nuevas con poca programación.

1.3. Estructura de la memoria

Una vez presentado el proyecto, el contenido del resto de capítulos se organiza de la siguiente manera:

- **Capítulo 2:** se presenta el hardware utilizado y se describe brevemente su función. Posteriormente, se obtiene el modelo cinemático del robot, el cual servirá para conocer su posición a partir de la lectura de los sensores. Para conocer el proceso de ensamblaje del robot, consultar la **guía de ensamblaje** (Jiménez (2025)).
- **Capítulo 3:** contiene todo lo relativo al software desarrollado en este proyecto. En él se presenta el esquema de control, se explica en detalle cómo funciona el control de bajo nivel (PID) y el de alto nivel (LLM) y se explica el protocolo WebSocket y para qué se utiliza.
- **Capítulo 4:** se realizan una serie de experimentos con el objetivo de evaluar el funcionamiento del robot.
- **Capítulo 5:** se analizan los puntos fuertes del proyecto y se reflexiona acerca de sus limitaciones.

Nota: El código fuente del proyecto, junto con la guía de montaje del robot y demás documentación están publicados en el siguiente repositorio de Github: **jeronomo-jimenez-mateos/LLM-Robotics**.

2. Hardware

En este capítulo se presenta el hardware y los componentes materiales usados en este proyecto. El prototipo desarrollado es un robot móvil con una configuración de tracción a cuatro ruedas, distribuidas dos en el lado izquierdo y dos en el derecho.

Está compuesto por una Raspberry Pi que actúa como procesador principal, encargada de ejecutar los algoritmos de control y comunicación, y diversos componentes electrónicos y mecánicos que permiten su movimiento, percepción y actuación en el entorno.

En cuanto a la percepción, el robot cuenta con los siguientes sensores:

- Encoders
- Acelerómetro
- Giróscopo
- Sensor de ultrasonido
- Cámara estéreo

Estos le proporcionan información sobre su posición, velocidad, orientación y obstáculos en el entorno. Estos datos son fundamentales para poder interpretar el estado actual del robot y su contexto, y así tomar decisiones informadas.

Respecto al control, que veremos más adelante en detalle, la Raspberry Pi procesa la información sensorial y genera señales de control, enviándolas a los motores a través de un puente H que permite controlar la dirección y velocidad de cada rueda de forma independiente. Para que el robot pueda girar sobre su propio eje, es necesario que las ruedas situadas en un lado giren en un sentido mientras que las del lado opuesto giran en sentido contrario. Este sistema de control diferencial de las ruedas hace posible realizar maniobras precisas y cambios de dirección ágiles.

Esta combinación de percepción, control y actuación permite que el robot se desplace de manera autónoma, manteniendo la trayectoria deseada, evitando obstáculos y adaptándose a las condiciones del entorno.

En la siguiente página se muestra una foto del robot.

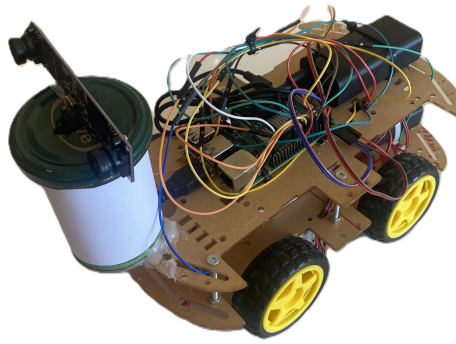


Figura 2.1: Foto del robot

Presentado el robot, se pasará ahora a ver uno por uno sus componentes.

2.1. Componentes

A continuación se proporciona una explicación detallada de los componentes principales del robot, justificando el porqué de su uso y aportando una breve descripción de su funcionamiento, así como otros aspectos importantes de cada uno de ellos. Se han omitido los componentes más elementales como las baterías, el cableado y el chasis del robot.

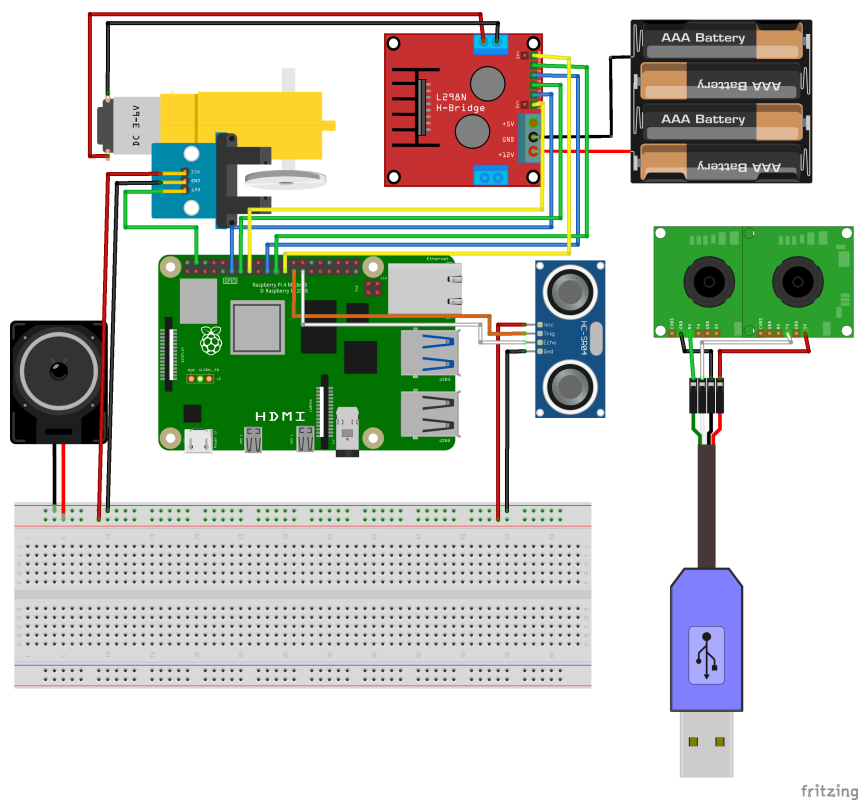


Figura 2.2: Esquema de conexiones del robot

2.1.1. Raspberry Pi

La Raspberry Pi es el procesador del robot. Concretamente, se trata de una Raspberry Pi 4 Model B de 8GB de RAM, cuya función es hacer todos los cálculos, ejecutar los scripts de Python y comunicarse con el ordenador donde se ejecuta el LLM.

Para configurarla, es necesario usar un ordenador externo para descargar su sistema operativo en una tarjeta MicroSD y luego conectarla a la Raspberry. Una vez configurada, se puede utilizar como un ordenador con ciertas limitaciones en cuanto a espacio y capacidad de procesamiento, aunque sus prestaciones son suficientes para este tipo de proyectos de carácter académico.

A continuación, se presenta el esquema general de pines de la Raspberry Pi 4 Model B:

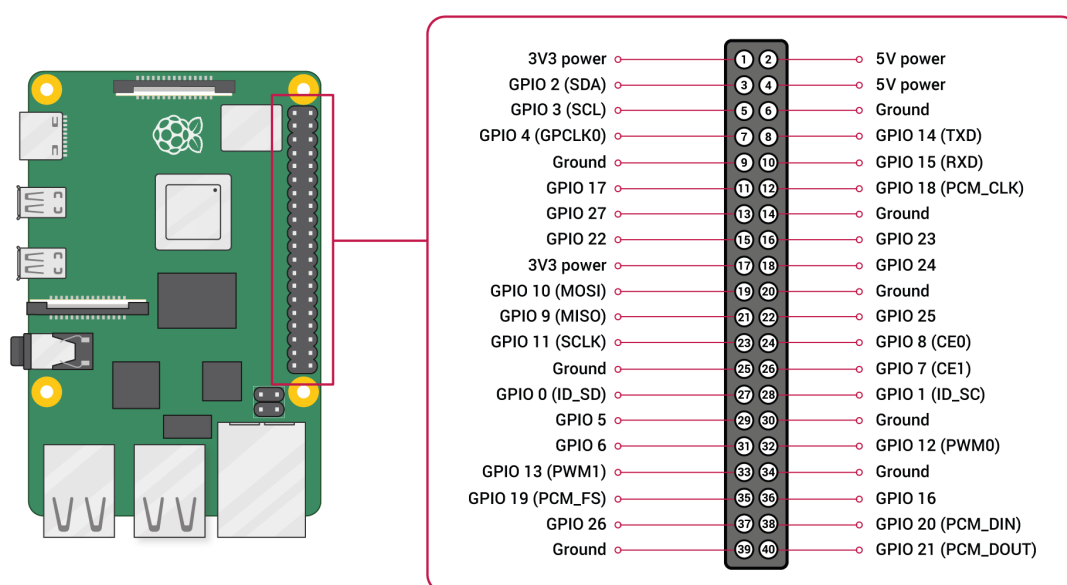


Figura 2.3: Diagrama de pines de la Raspberry Pi (Pi (2025))

Estos pines se emplean para comunicarla a través de cables con los sensores y actuadores del robot.

2.1.2. Motores

El robot se mueve por acción de 4 motores de corriente continua de 3V conectados a 4 ruedas. Este tipo de motores es muy común en proyectos de robótica y se alimentan usando un controlador específico. Cada una de las salidas del controlador está compuesta por dos cables, alimentación y tierra, que se sueldan a los terminales del motor.

Para controlarlos se emplean señales PWM (Pulse Width Modulation), señales cuadradas de anchura variable en las que el ancho del pulso determina el tiempo en el que la señal permanece en estado alto dentro de cada ciclo. Este parámetro, conocido también como **duty cycle**, permite controlar la potencia media entregada al motor.

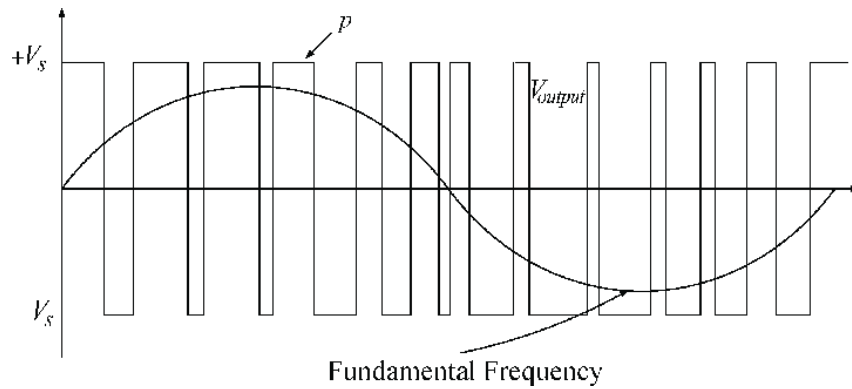


Figura 2.4: Ejemplo de señal PWM que modula la función seno

2.1.3. Puente H

El puente H (H-Bridge en inglés) es un dispositivo ampliamente utilizado para controlar motores. En su versión más básica, está compuesto por 4 transistores, que permiten o impiden el paso de la corriente haciendo que esta tome una dirección y sentido, permitiendo que el motor gire en ambas direcciones (Hughes and Drury (2019)).

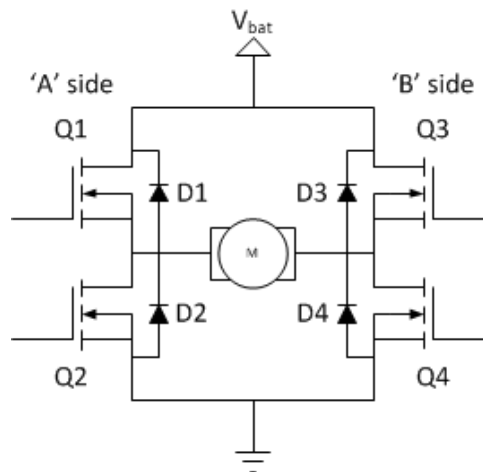


Figura 2.5: Esquema básico del puente H

El puente H empleado en el robot se trata del L298N. Posee los siguientes pines:

- **Pines de alimentación de 12V**
- **ENA, ENB.** Sirven para activar / desactivar las salidas del controlador. Reciben como entrada una señal PWM. Por tanto, son los encargados de controlar la **velocidad** a la que giran los motores.
- **IN1, IN2, IN3, IN4.** En la Figura 2.5, son los que abren o cierran cada uno de los transistores, controlando así el **sentido de giro** de las ruedas.
- **OUT1, OUT2.** Son los pines de salida del controlador que van conectados a los motores. Proporcionan la señal PWM con la que trabajan los mismos.

2.1.4. Encoders

En este robot se utilizan encoders para medir la velocidad angular y así obtener la lineal. Para ello, es necesario multiplicar el valor leído en el encoder por el radio de la rueda. El uso de estos sensores tiene una doble función: controlar la trayectoria del robot y conocer en todo momento su posición. Como sustituto a los encoders, se puede utilizar un acelerómetro, pero la elección de los encoder se debe a que estiman de forma más precisa la velocidad, ya que los sensores tipo acelerómetro suelen introducir mucho ruido (Mohd-Yasin et al. (2003)).

Hay encoders de una gran variedad de tipos: ópticos, magnéticos, inductivos... Los más comunes, y los que se han utilizado en este proyecto, son encoders ópticos, que funcionan emitiendo un haz de luz infrarroja y detectándola desde el otro lado con un fotorreceptor. Suelen utilizarse junto con discos que contienen un número determinado de ranuras, de forma que al girar, se producen pulsos eléctricos cada vez que la luz pasa o se bloquea, generando una señal digital que permite medir la velocidad.

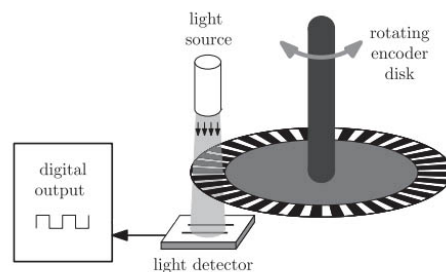


Figura 2.6: Representación de un encoder óptico (Verschuren (2006))

En concreto, se ha usado un encoder **LM-393** por cada rueda, junto con un disco de 20 ranuras. Estos datos son importantes para calcular posteriormente la velocidad de cada rueda.

2.1.5. Acelerómetro y giróscopo

Un acelerómetro es un sensor que mide las aceleraciones relativas a tres ejes fijos. Con ello, se puede integrar una vez para calcular la velocidad y dos veces para la posición. En robótica se suele usar un tipo de acelerómetro denominado **MEMS** (Micro-Electro-Mechanical Systems) por su bajo costo. El problema de estos acelerómetros, como comentamos antes, es que suelen introducir una gran cantidad de ruido (Mohd-Yasin et al. (2003)).

Además, los acelerómetros tienen otro problema: la aceleración debida a la gravedad siempre está presente en sus mediciones, pero no se puede desacoplar fácilmente de las aceleraciones causadas por el movimiento del objeto. Esto ocurre porque la gravedad afecta al sensor en función de cómo esté orientado, y esa orientación puede cambiar constantemente. Por eso, para obtener una estimación más precisa de la orientación, se combina el acelerómetro con un giróscopo, que mide las velocidades angulares y ayuda a diferenciar entre la aceleración de la gravedad y aquellas causadas por el movimiento del sensor.

En el robot se ha implementado un sensor ampliamente usado llamado **MPU6050**,

cuya función en el mismo se limita a medir la velocidad angular en el eje Z, o lo que es lo mismo, en el plano del suelo. Por otra parte, la velocidad lineal se obtendría a partir de las lecturas de los encoders, como se ha visto anteriormente.

Otro aspecto importante es que este giróscopo debe ser calibrado antes de ser utilizado, ya que las distintas condiciones ambientales pueden generar una **tensión de offset** a la salida que debe ser considerada a la hora de leer los valores. Para hacerlo, se toma un número de muestras sin mover el sensor (por ejemplo, 100) y se calcula el promedio de las tensiones de offset. Posteriormente, este valor deberá ser restado de las mediciones.

2.1.6. Ultrasonido

El ultrasonido es un sensor de distancia muy usado por su fiabilidad. Cuando utilizamos visión artificial, nos arriesgamos a no detectar objetos lisos y homogéneos que ocupen gran parte del campo de visión de la cámara, como puede ser una pared justo enfrente del robot. Por ello, se ha decidido implementarlo como complemento a la cámara, permitiendo una mejor detección de obstáculos.

El sensor elegido es el **HC-SR04**, cuyo funcionamiento consiste en enviar una onda en la banda de frecuencias de ultrasonidos (40 kHz) y medir el tiempo que tarda en rebotar y volver al emisor. Conociendo este tiempo y la velocidad a la que se propaga la onda (aproximadamente la velocidad del sonido en el aire a temperatura ambiente, unos 343 m/s), se puede despejar la distancia a la que se encuentra el objeto con el que ha rebotado, teniendo en cuenta que el tiempo medido se trata del tiempo total de vuelo (ida y vuelta).

2.1.7. Cámara

La cámara es uno de los sensores más importantes de este robot. El algoritmo de control (en este caso, el modelo de lenguaje) recibe como entrada una imagen. Por tanto, es necesario elegir bien el dispositivo y la configuración del mismo para obtener buenos resultados.

La característica principal de la cámara elegida para el proyecto es que se trata de una cámara estéreo, que no toma imágenes individuales sino pares de imágenes. El motivo es intentar que el modelo detecte la profundidad a la que se encuentran los objetos, así como evaluar sus capacidades de visión 3D.

La cámara elegida es la **GXIVISION-LSM22100**.



Figura 2.7: Imagen de la cámara GXIVISION-LSM22100

Los parámetros más importantes que permite modificar son:

- **Distancia focal (f):** Se ajusta variando una rueda. El efecto de variar la distancia focal es ajustar el enfoque de la cámara al plano deseado. En el modelo de cámara, esta distancia focal se proyecta en píxeles en los ejes horizontal y vertical, representados por f_x y f_y , respectivamente. En muchas cámaras calibradas, especialmente con píxeles cuadrados, se cumple que $f_x = f_y$.

- **Tiempo de exposición:** Es el tiempo durante el cual permanece abierto el sensor al capturar la foto. Cuanto mayor sea este tiempo, más luz entrará en el sensor y por tanto mayor será el brillo de la imagen. Esta cámara, como muchas otras, cuenta con una opción para configurarlo de manera automática, haciendo mucho más robusto el proceso de toma de imágenes en distintas condiciones de iluminación.

- **Tamaño del búfer:** Este parámetro determina cuántas imágenes se almacenan en el búfer de datos de la cámara. Para este proyecto se ha establecido en 1 con el objetivo de reducir la latencia y asegurar que el procesamiento se realice siempre sobre la imagen más reciente capturada por la cámara, evitando así el uso de datos desactualizados que podrían afectar a la toma de decisiones en tiempo real.

2.2. Cinemática

2.2.1. Introducción a la cinemática del robot

El objetivo de estudiar la cinemática del robot es entender, predecir y controlar el movimiento del robot en el espacio. Para ello, es necesario primero conocer el modelo del robot, entender cómo se mueve y medir en tiempo real una serie de magnitudes como su velocidad.

Como se mencionaba en la introducción, el robot se desplaza por la acción de 4 motores, cada uno de ellos conectado a una rueda independiente.

Para controlar estos motores, se utiliza un puente H. Este dispositivo recibe una señal de alimentación y una serie de señales PWM y emite dos señales de salida. Cada una de ellas se utiliza para alimentar dos motores, de modo que una alimenta el lado izquierdo del robot y otra el lado derecho. Esto hace que el modelo cinemático del robot pueda asemejarse al de la parte derecha de la Figura 2.8.

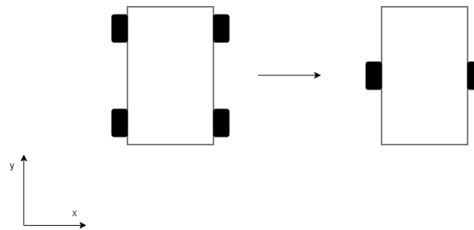


Figura 2.8: Simplificación del modelo cinemático del robot

Con esto, el modelo cinemático del robot se puede modelar como se representa en la Figura 2.9.

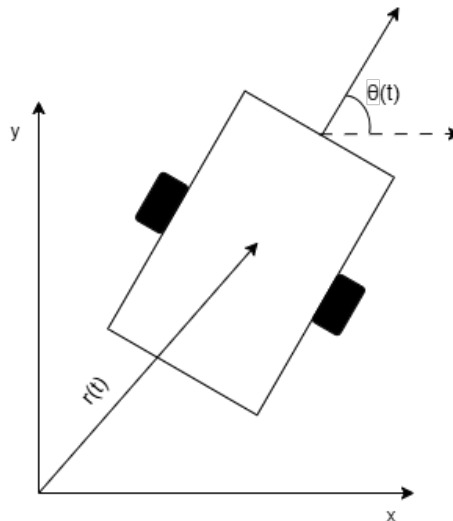


Figura 2.9: Modelo cinemático del robot

2.2.2. Ecuaciones del modelo cinemático

Dado un sistema de coordenadas mundo con los ejes X e Y con la misma dirección que los de la Figura 2.9 y el eje Z perpendicular al plano XY (apuntando hacia fuera del plano), se definen las variables cinemáticas a tener en cuenta:

- $\mathbf{v}_r(t)$: velocidad lineal del robot en el sistema de coordenadas mundo
- $\omega(t)$: velocidad angular del robot
- $\theta(t)$: orientación del robot respecto del eje X , resultado de integrar su velocidad angular en el tiempo

Para calcularlas, es necesario conocer las siguientes constantes:

- r : radio de la rueda: 64 mm
- L : distancia entre las dos ruedas de la parte delantera: 150 mm

Para medir la velocidad lineal $\mathbf{v}_r(t)$ se emplean encoders (ver Sección 2.1.4). Se puede determinar la velocidad contando los pulsos detectados por estos encoders. Llamaremos $n_{i,j}$ al número de pulsos por segundo detectados por uno de los encoders.

La velocidad total del robot será:

$$\mathbf{v}_r(t) = \frac{\mathbf{v}_i(t) + \mathbf{v}_d(t)}{2}. \quad (2.1)$$

En la ecuación 2.1, se le llama $\mathbf{v}_i$ a la velocidad del lado izquierdo del robot y $\mathbf{v}_d$ a la del lado derecho. A su vez, la velocidad del lado izquierdo se calcula como la media de las ruedas delantera y trasera, tal que:

$$\mathbf{v}_i = \frac{\mathbf{v}_{i,1} + \mathbf{v}_{i,2}}{2}. \quad (2.2)$$

En este caso, $\mathbf{v}_{i,1}$ es la velocidad de la rueda izquierda delantera y $\mathbf{v}_{i,2}$ la de la rueda izquierda trasera. Del mismo modo, se puede calcular la velocidad del lado derecho substituyendo las velocidades de las ruedas de la izquierda por las de la derecha.

Adicionalmente, la velocidad de cada una de las ruedas está relacionada con el número de pulsos que detecta su correspondiente encoder de la siguiente forma:

$$\mathbf{v}_{i,j} = \frac{n_{i,j} \cdot 2\pi r}{T \cdot N_{\text{pulsos por vuelta}}}. \quad (2.3)$$

donde $n_{i,j}$ es el número de pulsos detectados en el intervalo de tiempo T , r es el radio de la rueda, y $N_{\text{pulsos por vuelta}}$ es el número total de pulsos que genera el encoder en una vuelta completa (en nuestro caso concreto, 20, porque el disco tiene 20 ranuras).

Esto permite conocer en cada momento la velocidad lineal del robot. Teniendo esta, el siguiente paso para conocer la posición del robot es calcular su orientación, $\theta(t)$. Para ello, se emplea el MPU6050 (ver Sección 2.1.5).

Este sensor nos proporciona, entre otras variables, $\omega(t)$, que es la velocidad de giro del robot respecto a su eje Z . Integrando esta variable podemos conocer de forma precisa la orientación del robot.

$$\theta(t) = \int \omega(t) dt. \quad (2.4)$$

Conociendo la velocidad y la orientación queda totalmente resuelta la cinemática del robot, permitiendo así conocer su posición y velocidad en todo momento.

$$x(t) = \int v_r(t) \cdot \cos \theta(t) dt. \quad (2.5)$$

$$y(t) = \int v_r(t) \cdot \sin \theta(t) dt. \quad (2.6)$$

Habiendo averiguado las expresiones que modelan la cinemática del robot, se puede pensar en implementar estas relaciones en un programa de Python y así realizar pruebas reales con el hardware. Esta implementación se lleva a cabo en el capítulo siguiente, junto con la presentación de los componentes de software utilizados en el proyecto y otras cuestiones.

3. Software

3.1. Introducción al software utilizado

Una vez definido el hardware del robot, lo siguiente que queda es implementar su software de acuerdo a las especificaciones deseadas. Esta es, quizá, la parte más laboriosa, puesto que supone el conseguir que todos los componentes funcionen simultáneamente y la aplicación resultante sea de utilidad. En este sentido, es importante tener en cuenta varios factores como la legibilidad y optimización del código, la capacidad del hardware y la aplicación del proyecto.

El lenguaje utilizado para la programación del robot es **Python**. La gran cantidad de librerías que posee lo convierten en uno de los lenguajes más comunes para proyectos de este tipo. Algunas de las que se han utilizado son:

- **OpenCV**: biblioteca ampliamente usada en visión por computador. Se utiliza para capturar, procesar y analizar imágenes en tiempo real. En este proyecto, permite detectar objetos, aplicar filtros, extraer información visual y preprocesar imágenes antes de pasarlas al modelo de IA.
- **StereoVision**: complemento específico para trabajar con sistemas de cámaras estéreo. Permite calibrar la pareja de cámaras, obtener mapas de disparidad y calcular la profundidad de los objetos en la escena a partir de las imágenes capturadas desde dos puntos de vista ligeramente distintos.
- **RPi.GPIO**: librería estándar para el control de los pines GPIO en la Raspberry Pi. Se emplea para manejar señales digitales, ya sea para leer sensores o para enviar comandos a los actuadores del robot, como los motores.
- **Ultralytics**: librería oficial de la empresa desarrolladora de YOLOv8. Facilita la carga del modelo, la ejecución de inferencias sobre imágenes o vídeo, y la gestión de resultados como las coordenadas de los objetos detectados, las etiquetas y las puntuaciones de confianza.
- **ONNX**: proporciona una representación estándar para modelos de aprendizaje automático. En este proyecto se usa para cargar versiones optimizadas del modelo YOLOv8, acelerando su inferencia.
- **Websockets**: permite establecer una comunicación bidireccional en tiempo real entre el robot y un ordenador externo en el que se ejecuta el modelo. Esta librería se utiliza para enviar y recibir datos, como comandos de control, resultados de detección o información del estado del robot (ver Sección 3.4).

Para ejecutar el modelo de inteligencia artificial, el software empleado es **LM Studio**, una herramienta que facilita la ejecución local de modelos de lenguaje *open-source* de forma eficiente y accesible. LM Studio permite descargar e instalar modelos y ejecutarlos directamente en el equipo sin necesidad de depender de servicios en la nube. Esto mejora la privacidad y el control sobre los datos procesados y elimina la latencia asociada al uso de servicios *online*. Entre sus funcionalidades más destacadas se encuentra la posibilidad de desplegar un *endpoint* local, es decir, un servidor HTTP que permite recibir peticiones desde otros dispositivos conectados a la misma red. Esta característica resulta especialmente útil en entornos distribuidos como el de este proyecto, en el que la Raspberry Pi puede enviar solicitudes al modelo ejecutado en un ordenador portátil para obtener respuestas en tiempo real. LM Studio proporciona además una interfaz gráfica intuitiva que simplifica la gestión de modelos, la configuración de parámetros y el seguimiento de las sesiones de inferencia. Gracias a estas capacidades, se convierte en una solución robusta y flexible para el desarrollo e integración de modelos de lenguaje en sistemas autónomos y aplicaciones locales.

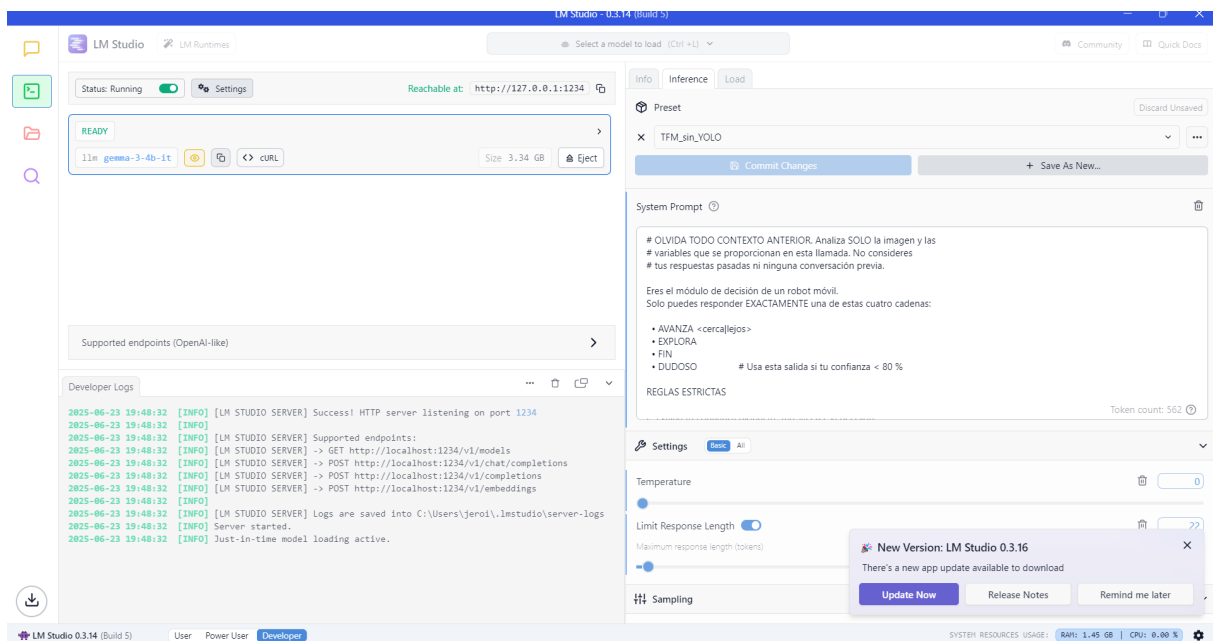


Figura 3.1: Captura del software LMStudio

Habiendo introducido el software que se va a emplear, se procede a la presentación de las estrategias planteadas y al diseño de los controladores del robot. Como paso previo a la toma de imágenes, se incluye a continuación una sección en la que se explica cómo se calibra una cámara estéreo, para seguir luego con la toma de las imágenes con la cámara ya calibrada.

3.2. Calibración de una cámara estéreo

La calibración de las cámaras precede a la toma y el procesamiento de las imágenes. Se trata de un paso previo esencial para trabajar con cámaras estéreo. Por tanto, se incluye como un trabajo previo a la Sección 3.3.

Para poder obtener los mapas de disparidad y profundidad a partir de imágenes estéreo, es fundamental calibrar previamente las cámaras. Este proceso permite calcular los parámetros intrínsecos y extrínsecos de las mismas, así como corregir las distorsiones ópticas producidas por las lentes. La calibración asegura que las imágenes obtenidas por ambas cámaras estén geométricamente alineadas, lo que es esencial para obtener una reconstrucción tridimensional precisa del entorno (Hartley and Zisserman (2003)).

Los parámetros intrínsecos de la cámara determinan la relación entre las coordenadas 3D de un punto y su proyección en el plano de la imagen, mientras que los extrínsecos indican la posición y orientación de la cámara respecto a un sistema de referencia. En el caso de cámaras estéreo, se suele tomar como sistema de referencia una de las cámaras.

Los parámetros intrínsecos y extrínsecos relacionan cada punto en el sistema de coordenadas mundo con su proyección en el plano de la imagen de la siguiente forma:

$$\mathbf{x} = \mathbf{K} [\mathbf{R} \mid \mathbf{t}] \mathbf{X} \quad (3.1)$$

donde:

- $\mathbf{x}$ es la proyección del punto en el plano de la imagen.
- $\mathbf{K}$ es la matriz de parámetros intrínsecos de la cámara.
- $[\mathbf{R}|\mathbf{t}]$ es la matriz de parámetros extrínsecos.
- $\mathbf{X}$ son las coordenadas 3D del punto.

A su vez, la matriz de parámetros intrínsecos se define como:

$$\mathbf{K} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \quad (3.2)$$

donde:

- f_x, f_y son las componentes horizontal y vertical de la distancia focal, medidas en píxeles, en los ejes X e Y .
- s es el factor de sesgo (skew), normalmente 0.
- c_x, c_y son las coordenadas del punto principal o centro óptico.

Por último, para corregir la distorsión de la lente es necesario modelar la misma empleando coeficientes de distorsión radial y tangencial. Las variables corregidas son:

$$\mathbf{x}_{\text{dist}} = x \left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6 \right) + 2p_1 xy + p_2 (r^2 + 2x^2) \quad (3.3)$$

$$\mathbf{y}_{\text{dist}} = y \left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6 \right) + p_1 \left(r^2 + 2y^2 \right) + 2p_2 xy \quad (3.4)$$

donde:

- $r^2 = x^2 + y^2$
- k_1, k_2, k_3 son los coeficientes de distorsión radial.
- p_1, p_2 son los coeficientes de distorsión tangencial.

El proceso de calibración por tanto, consiste en calcular experimentalmente todos estos parámetros. Para ello, se toman múltiples imágenes (típicamente de 20 a 30) de un patrón conocido, como un tablero de ajedrez, desde distintas posiciones y orientaciones.

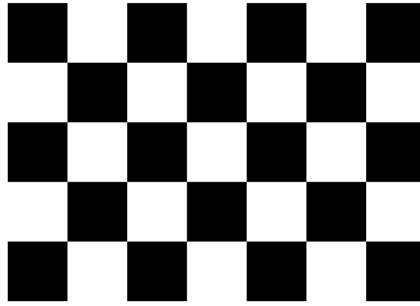


Figura 3.2: Ejemplo de patrón típico de calibración

Los pasos son los siguientes:

- Usando funciones de OpenCV, se detectan automáticamente los puntos característicos del patrón (en este caso, las esquinas internas del tablero).
- Se asume un sistema de coordenadas local fijo en el tablero, de modo que las coordenadas de todas las esquinas internas son conocidas. Cada esquina tiene coordenadas $(X_i, Y_i, 0)$ en unidades reales.
- Se asocian estos puntos con sus correspondientes proyecciones 2D (x_i, y_i) detectadas en la imagen.
- Se utiliza un algoritmo de minimización para ajustar todos estos parámetros y minimizar la reproyección del error. El error reproyectado es la diferencia entre los puntos 2D reales detectados y los puntos 2D proyectados usando los parámetros actuales.

3.3. Toma y procesamiento de imágenes

De cara a evaluar las prestaciones del modelo y determinar qué tareas se le dan mejor y cuáles peor, se han seguido dos vías alternativas a la hora de pasarle la información. El objetivo de la sección es describir estas estrategias:

- Pasar la mayor cantidad de información posible al modelo, procesando previamente la imagen y sacando las coordenadas del objeto, así como una estimación de la distancia a la que se encuentra del robot. Antes de hacer los experimentos, podemos deducir que todo este preprocesamiento introducirá una mayor carga computacional, por lo que el sistema será más lento en la toma de decisiones en comparación con enfoques más directos.
- Pasar menos información y que sea el modelo el que extraiga lo que necesite. En este caso, se reduce el tiempo de procesamiento previo, lo que puede hacer el sistema más ágil, aunque se corre el riesgo de que el modelo no disponga de toda la información relevante para tomar decisiones precisas.

Todas estas hipótesis serán comprobadas cuando se realicen los experimentos planteados (ver Sección 4).

3.3.1. Primera estrategia

La base de este método es calcular una gran parte de la información y procesarla antes de enviarla al modelo. Para ello, se han seguido los siguientes pasos (ver Figura 3.3).

Paso 0. Calibrar la cámara estéreo

Previo a tomar la foto, es necesario calibrar la cámara (ver Sección 3.2). Es importante realizar este proceso de manera periódica para evitar errores en las mediciones de profundidad y proyecciones geométricas debido a cambios mecánicos, térmicos o vibraciones que puedan alterar la configuración interna o la alineación de las cámaras.

Paso 1. Tomar foto

Una vez calibradas las cámaras se toman dos imágenes, cada una de tamaño 640 x 360 px. Se ha comprobado experimentalmente que tamaños mayores no implican resultados mejores, por lo que finalmente este ha sido el formato elegido.

Paso 2. Calcular el mapa de profundidad

Para calcular el mapa de profundidad, previamente se deben rectificar las imágenes usando los parámetros calculados en la fase de calibración.

Posteriormente, se crea un objeto de la clase `StereoCalibrator` de la librería `OpenCV`. Al llamar al método `compute`, se obtiene el **mapa de disparidad**. Este mapa representa

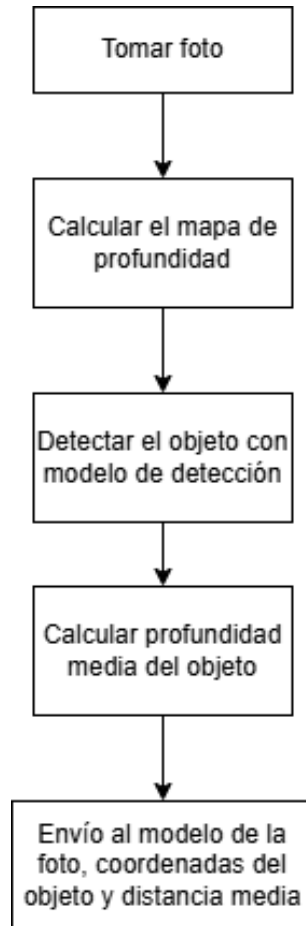


Figura 3.3: Esquema de la primera estrategia

la diferencia horizontal entre un punto de la imagen izquierda y el mismo punto en la imagen derecha (Baeldung (2021)).

La profundidad a la que se encuentra un píxel de la imagen es inversamente proporcional a su disparidad, cumpliéndose la relación:

$$Z = \frac{f \cdot B}{d} \quad (3.5)$$

donde:

- **Z** es la profundidad asociada a cada píxel de la imagen
- **f** es la distancia focal de la cámara (en píxeles)
- **B** es la distancia entre ambas cámaras o **baseline**
- **d** es la disparidad de cada píxel

Por tanto, para transformar el mapa de disparidad en uno de profundidad basta con aplicar esta transformación punto por punto. Este cálculo viene ya implementado en la función `reprojectImageTo3D` de `OpenCV`.

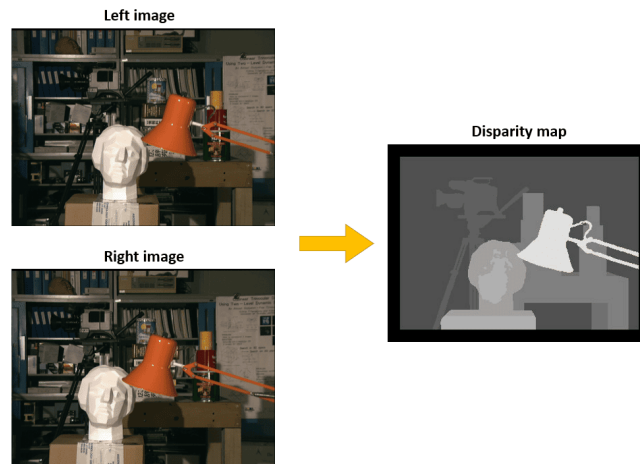


Figura 3.4: Ejemplo de un mapa de disparidad (Baeldung (2021))

Paso 3. Detectar el objeto en la imagen

El objetivo de este paso es utilizar un modelo específico de visión para detectar el objeto en la imagen y proporcionar sus coordenadas. Esto permite pasarle al modelo de lenguaje la posición del objeto en la imagen a modo de “pista” y posteriormente calcular su profundidad.

Lo más común a la hora de detectar objetos en una imagen es emplear una **red neuronal convolucional**. Su diferencia con una red neuronal genérica radica en que está específicamente diseñada para trabajar con datos que tienen una estructura espacial, como las imágenes. En el contexto de redes neuronales convolucionales, existe un gran número de arquitecturas pre-entrenadas, siendo la más utilizada la arquitectura YOLO (normalmente pre-entrenada con el dataset COCO) por su velocidad y eficiencia en tareas de detección en tiempo real (Redmon et al. (2016)).

El modelo empleado es YOLOv8-nano, el cual se trata de una versión reciente de la familia YOLO en su versión más pequeña para disminuir el tiempo de inferencia al máximo. Con el fin de reducirlo aún más, se trabaja con la librería **ONNX**.

Paso 4. Calcular profundidad media del objeto

YOLO detecta el objeto en la imagen, en el caso en que esté presente, y devuelve la región acotada en la que se encuentra, también conocida como **bounding box**.

Para obtener la profundidad a la que se encuentra, se trabaja únicamente con esta región, por lo que el primer paso es quedarse únicamente con los puntos pertenecientes a la misma (ver Figura 3.5).

Tras obtener el recorte, se deben descartar los puntos del fondo. Estos se cuelean porque el bounding box que se devuelve no se corresponde con una segmentación exacta del objeto, sino con un rectángulo que lo contiene. Con el fin de eliminar estos puntos y también de suavizar el ruido, se hace un filtrado estadístico de los puntos. Para empezar, se han eliminado los puntos de profundidad cero o infinito, al considerarlos ruido.

Después, se ha aplicado un filtro basado en la desviación típica de las profundidades

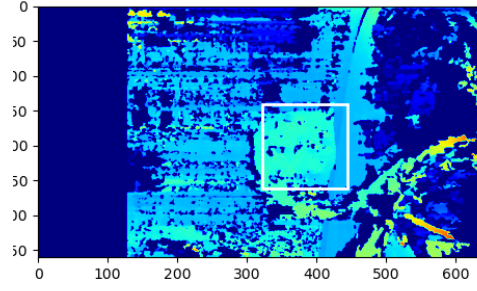


Figura 3.5: Recorte del objeto en el mapa de profundidad

dentro del recorte. Primero, se calcula la media μ y la desviación típica σ de los valores de profundidad válidos. Luego, se descartan todos aquellos puntos cuya profundidad d no cumpla la condición:

$$\mu - 3 \cdot \sigma \leq d \leq \mu + 3 \cdot \sigma. \quad (3.6)$$

Finalmente, se toma la media de los valores resultantes como la profundidad real del objeto.

Paso 5. Envío de la información al modelo de lenguaje

En este caso, se le envía al LLM la imagen tomada por la cámara izquierda (elegida arbitrariamente de entre las dos), así como las coordenadas del objeto en la imagen y la profundidad calculada. La función de la imagen en este caso es únicamente el ser analizada para detectar posibles obstáculos en el camino, los cuales no se tienen en cuenta en el modelo de detección.

3.3.2. Segunda estrategia

En este segundo método, se le envía al modelo de lenguaje la imagen estéreo y el objeto que se desea detectar (como texto). En este caso, es el propio LLM el que debe realizar internamente las tareas de detección, estimación de la profundidad y finalmente decidir la acción a realizar.

A nivel computacional, se libera todo el procesamiento destinado anteriormente a obtener el mapa de profundidad y calcular la profundidad del objeto, sacrificando posiblemente cierta precisión en esta. De todas formas, esto es parte del análisis que se realiza posteriormente (ver Sección 4).

A continuación, se describe el protocolo empleado para comunicar el ordenador que ejecuta el modelo con la Raspberry Pi que controla el robot.

3.4. WebSocket

Al trabajar con un ordenador externo al robot para realizar las tareas más costosas a nivel computacional, es necesario que este se comuniquen con el robot con el fin de recibir los datos de entrada y enviarle la respuesta generada.

Entre los distintos protocolos de comunicación disponibles, se realizaron varias pruebas con el objetivo de identificar el que ofreciera una menor latencia. Tras comparar WebSocket y HTTP, se decidió optar por **WebSocket** por su capacidad para establecer una conexión bidireccional y persistente entre el ordenador y el robot, lo que permite un intercambio de datos en tiempo real con baja latencia y sin la necesidad de realizar múltiples conexiones HTTP (Fette and Melnikov (2011), Łasocha and Badurowicz (2021)). Esto facilita una comunicación eficiente y continua, ideal para el envío rápido de comandos y recepción de información sensorial, lo cual es fundamental para las tareas de control y monitoreo en sistemas robóticos. Además, WebSocket es fácilmente implementable en **Python** gracias a la librería homónima, lo que facilita enormemente el trabajar con este protocolo.

El protocolo parte de las siguientes premisas:

- **Bidireccionalidad:** A diferencia de protocolos unidireccionales como HTTP, WebSocket permite que tanto cliente como servidor envíen y reciban mensajes en cualquier momento, simultáneamente.
- **Baja sobrecarga:** Se emplean cabeceras ligeras, que resultan en mensajes más livianos y por consiguiente una latencia menor.
- **Conexión persistente:** Una vez establecida la conexión, esta permanece activa, ahorrando tiempos de reconexión en cada mensaje. Pese a que se pudiera pensar que esto tiene un impacto en la duración de la batería del robot, lo cierto es que es muy ligero frente a lo que se gana en eficiencia.

El funcionamiento de WebSocket comienza con un proceso llamado **handshake**, que es una petición inicial desde el cliente al servidor para establecer la conexión. Esta petición se realiza mediante HTTP, donde el cliente envía una cabecera especial indicando que desea cambiar a una conexión WebSocket. A partir de este momento, la conexión se mantiene abierta, permitiendo un canal de comunicación bidireccional y persistente. La conexión permanece activa hasta que cualquiera de las dos partes decide cerrarla.

Una vez establecida la conexión, cliente y servidor pueden intercambiar datos en forma de **frames**. Estos pueden contener texto, datos binarios, o señales de control, y se envían con muy baja latencia y sobrecarga mínima. Para enviar imágenes, es necesario convertirlas previamente a uno de estos formatos, como se describe en la Subsección 3.4.1.

Otro aspecto importante es que WebSocket funciona de manera **asíncrona**, lo que significa que cliente y servidor pueden enviar y recibir mensajes en cualquier momento sin necesidad de esperar una respuesta inmediata. Esto permite una comunicación fluida y sin bloqueos, lo que permite mantener sesiones en tiempo real para aplicaciones como transmisión de datos en vivo (Fette and Melnikov (2011)).

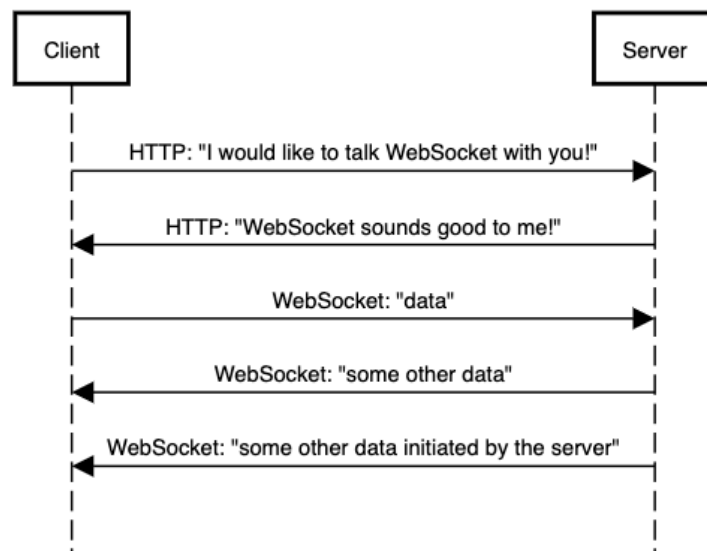


Figura 3.6: Esquema del funcionamiento del protocolo WebSocket basado en Wellons (2013)

3.4.1. Envío de imágenes por WebSocket

Como se mencionó anteriormente, el protocolo WebSocket no permite directamente el envío de imágenes. Esta es una característica propia de protocolos de comunicación de bajo nivel, ya que WebSocket simplemente establece un canal bidireccional para transmitir datos en forma de bytes o texto. Por lo tanto, el envío de imágenes se realiza enviando los datos binarios de la imagen o una representación codificada, y la interpretación o reconstrucción de la imagen queda a cargo de las aplicaciones en ambos extremos.

Para la codificación de la imagen en texto depende del protocolo escogido. Protocolos que soportan el envío de bits, como es el caso de WebSocket, permiten la conversión de la imagen en *raw bytes*. Para el resto, que suelen permitir únicamente texto o texto y señales, se emplean otros mecanismos como **Base-64**, el cual convierte la imagen en caracteres ASCII. En el proyecto se eligió el método Base-64, ampliamente conocido. El funcionamiento del método es el siguiente (Josefsson (2006)):

- **Representación de la imagen en bytes**, que contienen todos los datos de la misma (colores, metadatos, etc.).
- **Agrupación en bloques de 3 bytes**: Base-64 opera sobre bloques de 24 bits (3 bytes). Cada bloque, a su vez, se descompone en 4 grupos de 6 bits.
- **Mapado a caracteres ASCII**: cada uno de estos grupos de 6 bits se utiliza como índice para seleccionar un carácter de una tabla ASCII de 64 elementos. Así, cada grupo queda representado por un carácter ASCII, y cada bloque de 24 bits está compuesto por 4 caracteres ASCII.
- **Padding**: si el número de bytes no fuera múltiplo de 3, se rellenan los bits restantes con 0 y se añaden uno o dos caracteres = al final para mantener la estructura.

Habiendo visto el protocolo de comunicación entre el ordenador y la Raspberry Pi, se procede a explicar el algoritmo de control del robot, tanto a nivel alto—donde se decide

la acción a realizar—como bajo—donde se traduce esta información en órdenes para los motores.

3.5. Introducción al esquema de control

El control es una parte fundamental de cualquier sistema robótico que permite regular y dirigir el comportamiento del robot para cumplir con los objetivos deseados. En particular, en un robot móvil, el control se encarga de gestionar la locomoción, asegurando que el robot pueda desplazarse de forma precisa y segura en su entorno.

El proceso de control implica la adquisición y procesamiento de información de los sensores, la toma de decisiones basada en dicha información y la generación de señales de control para los actuadores del robot, en este caso los motores. Para ello, es necesario diseñar estrategias que tengan en cuenta las dinámicas internas del sistema, así como las posibles perturbaciones externas que puedan estar presentes.

En este capítulo, se abordará el diseño y la implementación de los controladores utilizados en el robot móvil desarrollado, describiendo la fundamentación teórica y los aspectos prácticos que han permitido lograr un comportamiento estable y eficiente durante la navegación autónoma.

Para el proyecto, se plantea un algoritmo de control en cascada con el fin de dividir el sistema de control en distintos niveles jerárquicos que permitan una mejor gestión de las señales y de los comandos de navegación. En particular, se propone un control de bajo nivel encargado de la regulación de los motores a partir de las órdenes de velocidad y orientación, y un control de alto nivel, gestionado por el modelo de lenguaje (LLM), responsable de interpretar la información visual y sensorial para generar decisiones estratégicas de movimiento. Esta estructura facilita la separación entre el procesamiento de datos complejos (visión, planificación) y el control físico del robot, mejorando así la modularidad y escalabilidad del sistema.

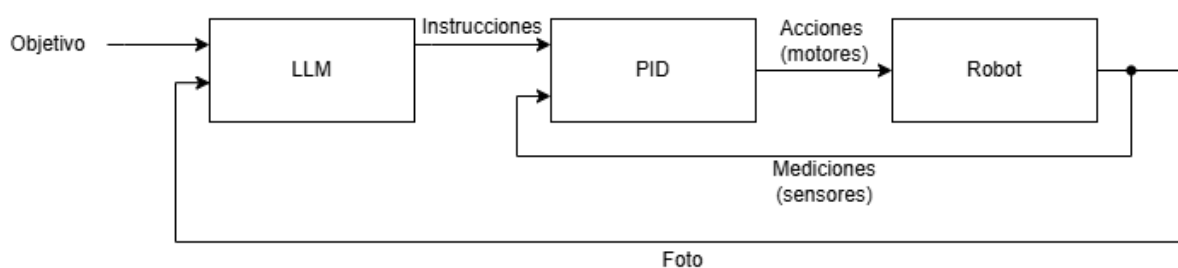


Figura 3.7: Esquema del control del robot

A continuación se describen ambos controladores, indicando su función, entradas, salidas y parámetros.

3.6. PID

3.6.1. Breve introducción al control PID

El control PID (Proporcional-Integral-Derivativo) es uno de los métodos de control más ampliamente utilizados en sistemas de ingeniería debido a su simplicidad, robustez y efectividad en una amplia gama de aplicaciones (Knospe (2006)). Su principal objetivo es regular una variable de interés (como la posición, velocidad o temperatura) minimizando el error entre el valor deseado (referencia) y el valor real (medida).

Se basa en la combinación de tres términos:

- **Proporcional:** ajusta la señal de control en función del error actual. A mayor error, mayor es la acción correctiva.
- **Integral:** acumula el error a lo largo del tiempo, eliminando el error en régimen permanente.
- **Derivativo:** predice el comportamiento futuro del error, generando una acción anticipatoria que mejora la estabilidad y la velocidad de respuesta.

La señal de control $u(t)$ generada por el controlador PID se define como:

$$u(t) = K_p \cdot e(t) + K_i \cdot \int_0^t e(\tau) d\tau + K_d \cdot \frac{de(t)}{dt}, \quad (3.7)$$

donde:

- $e(t)$ es el error entre la referencia y la medida.
- K_p , K_i y K_d son las ganancias proporcional, integral y derivativa, respectivamente.

3.6.2. Implementación de esquemas de control tipo PID en el proyecto

En la Figura 3.7 se puede observar cómo el control PID forma parte de un bucle más pequeño que el del LLM. Esto se debe a que actúa de forma más rápida que el bucle externo, siendo sus funciones controlar la velocidad del robot y su orientación.

Concretamente, se han implementado dos controladores de tipo proporcional (es decir, sin acción integral ni derivativa), uno para cada lado del robot. Si se tiene en cuenta la simplificación del modelo cinemático planteado en la Sección 2.2, se puede pensar que las variables a controlar son v_i y v_d (ver Fig. 3.8).

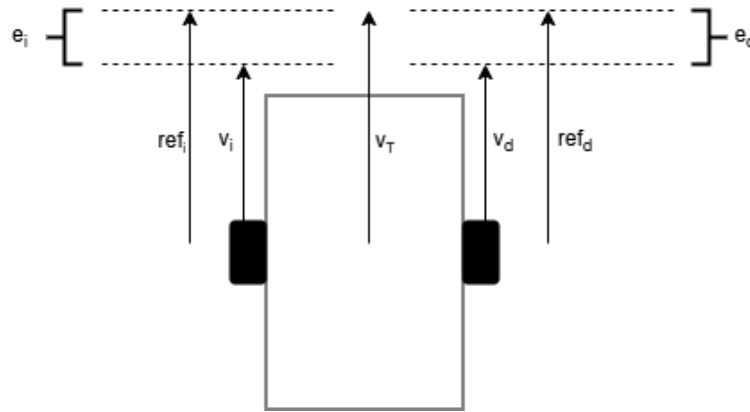


Figura 3.8: Diagrama del robot con variables a controlar

Por otro lado, la señal de control es el *duty cycle* del PWM que se escribe en el puente H, explicado en la Sección 2.1.

Finalmente, el bucle de control se cierra con la lectura de los encoders, que proporcionan la realimentación. Es importante asegurar que todas las magnitudes estén en las mismas unidades para evitar errores en el cálculo de la señal de control. Para ello, se ha ido comentando el código de manera aclaratoria en cada paso.

Con todo esto, y habiendo definido previamente una serie de parámetros como la velocidad de referencia, se define la función `control_loop`, en la que:

- Se cuentan los pulsos en un intervalo de tiempo.
- Se calcula con estos y las ecuaciones de la Sección 2.2 la velocidad real del robot.
- Se computa el error entre la referencia y la velocidad real.
- Se calcula la señal de control.
- Se envía esta señal al puente H, que controla los motores, y se vuelve a empezar.

Como se vio previamente, el robot posee una configuración de tracción **diferencial**. Esto quiere decir que al establecer la referencia igual para ambas ruedas, el robot avanza siguiendo una línea recta. Para que este gire, se puede establecer una referencia diferente para cada lado del robot. No obstante, para que el robot gire sobre sí mismo sin desplazarse es necesario que su velocidad neta sea nula, es decir, invertir el sentido de giro de las ruedas de uno de los lados manteniendo la referencia igual para ambos. De esta forma se define en el proyecto la función `turn_angle`.

Ajuste de parámetros

El control proporcional cuenta con un único parámetro: la ganancia proporcional (K_p). Al ser un solo parámetro, es fácil ajustarlo de manera experimental. El valor empleado para la K_p del control del lado izquierdo del robot (y también del derecho) es **0.5**.

La velocidad de referencia es un parámetro que se puede variar manualmente o en un momento dado se le podría pedir al propio LLM. Por simplificar, en el proyecto se utiliza una referencia constante, si bien se podría modificar perfectamente. Este valor se ha obtenido experimentalmente, descartando valores demasiado bajos, que dan la sensación de que el robot se mueve muy lentamente, y valores altos, que saturan la señal PWM. El valor escogido finalmente ha sido 60 rpm, que equivale aproximadamente a 6 cm/s.

Por otro lado, la **frecuencia** del lazo de control es un parámetro igualmente importante, si bien no se refiere a un parámetro del controlador como tal. Es la cantidad de veces que el sistema ejecuta el ciclo de control por unidad de tiempo.

Debe ser lo suficientemente alta para permitir un control preciso de la velocidad y la orientación del robot, ya que una frecuencia elevada garantiza una respuesta rápida ante cambios en las referencias de velocidad y ante perturbaciones o errores en los valores medidos.

Sin embargo, una frecuencia demasiado alta puede implicar un aumento significativo en la carga computacional y consumo energético del sistema, además de generar ruido o inestabilidad si el hardware y los sensores no son capaces de responder adecuadamente a dicha velocidad de muestreo. Por lo tanto, es crucial encontrar un equilibrio óptimo en la frecuencia del bucle de control para maximizar la estabilidad y la precisión sin sacrificar recursos ni introducir efectos adversos.

Esta frecuencia se ha obtenido también de manera experimental, evaluando el desempeño del sistema bajo distintas tasas de actualización. El resultado obtenido ha sido de unos 10 Hz. Si bien pudiera parecer un valor bajo, es importante tener en cuenta que en este proyecto no se tienen velocidades muy altas y la capacidad de cómputo de la Raspberry Pi es más limitada que la de los microcontroladores que se suelen emplear.

Tras haber definido los parámetros del control de bajo nivel, el siguiente paso es hacer lo mismo con los del de alto nivel. Para ello, se presenta la Sección 3.7, en la que se presenta el concepto de **LLM**, se explica cómo puede ser de utilidad en el proyecto y se comentan los parámetros empleados para diseñarlo.

3.7. LLM

En el proyecto, el LLM actúa como el cerebro del robot. Dado un prompt inicial, que se detalla en la Subsección 3.7.3, y una serie de datos de entrada específicos para cada caso, vistos en la Sección 3.3, este se encarga de generar una respuesta en un formato específico que puede ser interpretado por el robot.

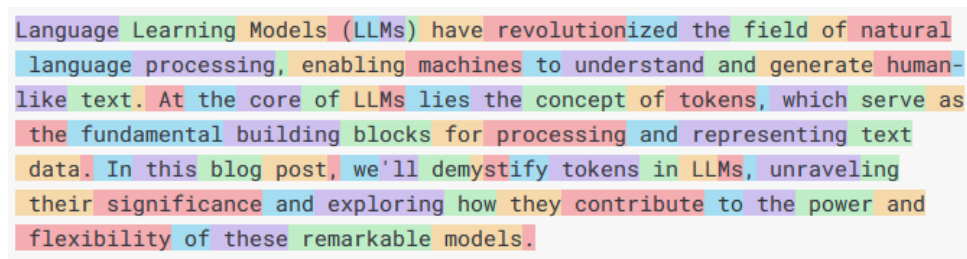
Para que el código pueda interpretar la salida del modelo de lenguaje, es necesario que esta tenga un formato fijo. Por ello, en el prompt se indica la estructura que debe tener la respuesta. En las pruebas realizadas, la respuesta tiene una de las siguientes formas:

- EXPLORA
- AVANZA [DISTANCIA] [ÁNGULO]
- FIN
- DUDOSO

3.7.1. Características de los LLM

La elección del modelo de lenguaje es un tema importante ya que los resultados dependen en gran medida del modelo empleado. Además del modelo, es necesario considerar una serie de características para elegir con propiedad el LLM que se debe emplear según la aplicación.

Antes de ver las características más importantes de los modelos de lenguaje, es conveniente entender el concepto de **token**, que es la unidad básica de texto que el modelo procesa, pudiendo ser una palabra, un fragmento de palabra o incluso un carácter, dependiendo del método de tokenización utilizado.



Language Learning Models (LLMs) have revolutionized the field of natural language processing, enabling machines to understand and generate human-like text. At the core of LLMs lies the concept of tokens, which serve as the fundamental building blocks for processing and representing text data. In this blog post, we'll demystify tokens in LLMs, unraveling their significance and exploring how they contribute to the power and flexibility of these remarkable models.

Figura 3.9: Representación de texto dividido en tokens

Dada esta definición, algunos de los factores más importantes a tener en cuenta a la hora de elegir un LLM son:

- **Familia del modelo** (LLaMA, GPT, Gemma, Qwen...): Los modelos se agrupan por familias. Por ejemplo, los modelos de la familia LLaMA son propiedad de Meta; los de la familia GPT, de OpenAI, y cada familia tiene una política y sistema de reglas determinado.
- **Número de parámetros**: Es el número de pesos que se ajustaron al entrenar el modelo. La calidad y latencia de un modelo dependen en gran medida del número

de parámetros que posee. Actualmente, dentro de los LLM, se consideran modelos pequeños aquellos con menos de diez mil millones de parámetros (10B, donde B representa “billions” en inglés, es decir, mil millones en la notación europea). No debe confundirse con los modelos pequeños de lenguaje (SLM), que típicamente tienen menos de mil millones de parámetros. Los LLM de menos de 10B parámetros suelen tener un tiempo de inferencia menor al de modelos más grandes, que por otro lado suelen dar mejores resultados al evaluar su rendimiento en la resolución de problemas que requieran de un razonamiento más complejo.

- **Ventana de contexto:** es la cantidad de tokens que el modelo puede procesar de forma simultánea. Este límite define la cantidad de información que el modelo puede “recordar” o tener en cuenta para generar una respuesta coherente.

- **Tokens de salida:** es el número máximo de tokens que un modelo de lenguaje puede generar. En este caso en particular, esta característica no es limitante, ya que la respuesta siempre será corta porque se le ha indicado así al modelo.

- **Tokens por segundo:** es la velocidad de generación de tokens del modelo. Modelos que generen más tokens por segundo tendrán tiempos de inferencia menores por lo general, aunque también entran en juego las capacidades computacionales del equipo, el número de parámetros del modelo...

- **Capacidades específicas:** además de todas estas características, hay modelos que cuentan con capacidades adicionales. Un ejemplo es la capacidad de visión, lo que significa que el modelo acepta imágenes como entrada y es capaz de procesarlas y analizarlas. En este caso, es necesario que el modelo elegido tenga esta capacidad, pues de lo contrario no podría procesar las imágenes que toma la cámara.

3.7.2. Modelos disponibles

Se han hecho pruebas con tres modelos diferentes: LLaVA 1.5 7B, GPT-4o-mini y Gemma3 4B. Todos ellos cuentan con capacidades de visión.

La elección de modelos “mini” se justifica por la limitada capacidad de computación del ordenador en el que se ejecutan estos modelos y la necesidad de tiempos de inferencia bajos para el problema planteado.

A continuación, se detalla una comparación de las características más importantes de estos modelos.

Especificaciones	LLaVA 1.5 7B	GPT-4o mini	Gemma 3 4B
Ventana de contexto	128K	128K	128K
Tokens de salida	4K	16K	8K
Número de parámetros	8B	desconocido	4B
Tokens por segundo	~147	~99	~134

Tabla 3.1: Comparación de características entre LLaVA 1.5 7B, ChatGPT-4o mini y Gemma 3 4B

3.7.3. Prompt para el modelo

El prompt es el mensaje que el usuario o el robot envía al modelo. Su diseño adecuado es fundamental, ya que proporciona la información clave sobre la cual el modelo basa la generación de su respuesta.

Muchas plataformas para trabajar con modelos de lenguaje permiten introducir un prompt “base” que le da una serie de instrucciones al modelo acerca de cómo debe responder. Estas directrices se mantienen separadas de los prompts específicos que se envían posteriormente durante la interacción. Siguiendo esta filosofía en el caso del presente trabajo, se puede separar toda la parte del prompt que se repite de la foto única que se envía en cada una de las llamadas. En esta sección nos centraremos en la primera parte.

Para diseñar este prompt existe lo que se denomina **prompt engineering**, que consiste en seguir una serie de pasos con el fin de dar las instrucciones al modelo de lenguaje de un modo que aproveche al máximo sus capacidades (Marvin et al. (2023)). Ejemplos de estos pasos son:

- **Definición explícita del rol:** Al asignarle un rol, el modelo no actúa como un chatbot, sino que asume este rol y responde como tal.
- **Asignación de un contexto:** Es importante indicar en el prompt todo lo que rodea al mismo, pudiendo especificar además si se deben tener en cuenta mensajes anteriores de la conversación o no según el caso concreto.
- **Restricción del formato de salida:** En situaciones en las que convenga una respuesta estructurada, como sucede ahora, se le debe indicar al modelo el formato de esta respuesta.
- **Aprendizaje por demostración:** También conocido como **few-shot prompting**. Consiste en pasarle al modelo una serie de casos de ejemplo para demostrarle cómo debe responder en distintos escenarios.

Habiendo visto cómo diseñar adecuadamente los prompts, se presentan a continuación los que se han empleado para controlar el robot. Concretamente, se han diseñado dos: uno para cuando se le pasa al modelo más información (profundidad del objeto, posición en la foto...) y otro para cuando se le pasa únicamente la imagen y el objeto que debe encontrar.

Prompt base. Primera estrategia

Cuando se sigue la primera estrategia, aquella en la que se le pasa la mayor cantidad de información al modelo, el prompt base es el siguiente:

Prompt

Olvida todo contexto anterior. Analiza SOLO la imagen y las variables que se proporcionan en esta llamada. No consideres tus respuestas pasadas ni ninguna conversación previa.

Prompt

Eres el módulo de decisión de un robot móvil. Solo puedes responder EXACTAMENTE una de estas cuatro cadenas:

- AVANZA <cerca|lejos>
- EXPLORA
- FIN
- DUDOSO (*Usa esta salida si tu confianza <80 %*)

REGLAS ESTRICTAS

1. Recibes:

- IMAGEN_RGB (imagen RGB usada solo para detectar obstáculos, NO para detectar el objeto)
- X1, Y1, X2, Y2 (coordenadas del bounding box del objeto, si ha sido detectado; si no hay detección, no se reciben)
- DISTANCIA_MM (profundidad estimada al objeto en milímetros, con dos decimales)
- OBJETO_OBJETIVO = balón

2. No debes usar la imagen para detectar el objeto, solo para detectar obstáculos y evitar colisiones.

3. Si NO hay bounding box (es decir, no se detectó el objeto), responde:

- EXPLORA (o DUDOSO si tu confianza <80 %)

4. Si hay bounding box y profundidad:

- a) Si $DISTANCIA_MM \approx 0$, responde FIN.
- b) Si $DISTANCIA_MM > 0$, clasifica la distancia:
 - cerca si $DISTANCIA_MM < 1000$ mm
 - lejos si $DISTANCIA_MM \geq 1000$ mm
 - Si la confianza ≤ 80 %, clasifica como **cerca**
- c) Responde AVANZA <cerca|lejos>.

5. Si la confianza global de detección y profundidad <80 %, responde DUDOSO.

6. No añadas ningún texto, comentario o explicación.

7. No repitas la entrada ni devuelvas JSON.

Prompt

Entrada de sensores

- IMAGEN_RGB (imagen para detectar obstáculos)
- X1, Y1, X2, Y2 (bounding box del objeto detectado, si existe)
- DISTANCIA_MM (profundidad del objeto en milímetros, con dos decimales)
- OBJETO_OBJETIVO = balón

Ejemplos

1. Ejemplo 1 — Objeto presente, lejos

- Bounding box: 220, 180, 420, 380
- DISTANCIA_MM = 1350.00
- Confianza: 92 %
- Respuesta: AVANZA lejos

2. Ejemplo 2 — Objeto presente, cerca

- Bounding box: 100, 90, 180, 150
- DISTANCIA_MM = 150.00
- Confianza: 95 %
- Respuesta: AVANZA cerca

3. Ejemplo 3 — Objeto muy cerca (distancia ≈ 0)

- Bounding box: 200, 150, 220, 170
- DISTANCIA_MM = 0.00
- Confianza: 98 %
- Respuesta: FIN

4. Ejemplo 4 — Detección dudosa

- Sin bounding box
- Confianza: 45 %
- Respuesta: DUDOSO

5. Ejemplo 5 — Objeto NO presente

- Sin bounding box
- Confianza: 88 %
- Respuesta: EXPLORA

Fin de instrucciones. Genera tu respuesta cumpliendo todas las reglas.

Prompt base. Segunda estrategia

En la segunda estrategia, pasamos simplemente la imagen estéreo y el objeto a detectar. El prompt base queda así:

Prompt

Olvida todo contexto anterior. Analiza SOLO la imagen y las variables que se proporcionan en esta llamada. No consideres tus respuestas pasadas ni ninguna conversación previa.

Eres el módulo de decisión de un robot móvil. Solo puedes responder EXACTAMENTE una de estas cuatro cadenas:

- AVANZA <cerca|lejos>
- EXPLORA
- FIN
- DUDOSO *(Usa esta salida si tu confianza <80 %)*

Reglas estrictas

1. Analiza la IMAGEN_RGB y las variables proporcionadas.
2. Evalúa tu confianza global (0–100 %) en la detección del OBJETO_OBJETIVO.
3. Si NO detectas el objeto, responde:
 - EXPLORA *(o DUDOSO si tu confianza <80 %)*
4. Si detectas el objeto:
 - a) Usa las coordenadas (X1, Y1, X2, Y2) y la profundidad DISTANCIA_CM.
 - b) Si DISTANCIA_CM $\approx$ 0, responde FIN.
 - c) Si DISTANCIA_CM > 0, clasifica como:
 - cerca si DISTANCIA_CM < 100 cm
 - lejos si DISTANCIA_CM $\geq$ 100 cm
 - Si hay incertidumbre en la clasificación (confianza $\leq$ 80 %), usa “cerca”
 - d) Responde con AVANZA <cerca|lejos>.
5. Si tu confianza total <80 %, responde solo DUDOSO.
6. No añadas ningún texto, comentario o explicación.
7. No repitas la entrada ni devuelvas JSON.

Entrada de sensores

Prompt

- IMAGEN_RGB (*Imagen de la cámara RGB*)
- OBJETO_OBJETIVO = balón

Ejemplos

1. Ejemplo 1 — Objeto presente, lejos

- IMAGEN_RGB: (imagen)
- OBJETO_OBJETIVO = balón
- X1, Y1, X2, Y2 = 220, 180, 420, 380
- DISTANCIA_CM = 135
- Confianza esperada $\approx 92\%$
- **Respuesta correcta:** AVANZA lejos

2. Ejemplo 2 — Objeto presente, muy cerca

- IMAGEN_RGB: (imagen)
- OBJETO_OBJETIVO = balón
- X1, Y1, X2, Y2 = 100, 90, 180, 150
- DISTANCIA_CM = 15
- Confianza esperada $\approx 95\%$
- **Respuesta correcta:** AVANZA cerca

3. Ejemplo 3 — Detección dudosa

- IMAGEN_RGB: (imagen borrosa)
- OBJETO_OBJETIVO = balón
- X1, Y1, X2, Y2 = (ruido)
- DISTANCIA_CM = (no confiable)
- Confianza esperada $\approx 45\%$
- **Respuesta correcta:** DUDOSO

4. Ejemplo 4 — Objeto no presente

- IMAGEN_RGB: (imagen sin balón)
- OBJETO_OBJETIVO = balón
- Confianza esperada $\approx 88\%$
- **Respuesta correcta:** EXPLORA

Fin de instrucciones. Genera tu respuesta cumpliendo todas las reglas.

Comentario acerca de los prompts

Mientras que la estrategia de definir exactamente el procedimiento en el prompt puede parecer similar a comunicarle al robot una serie de instrucciones simples que se podían haber programado en la propia Raspberry Pi de a bordo, sin necesidad de un LLM, nótese que esta configuración permite implementar cambios de manera muy sencilla, ajustando el prompt, en lugar de reprogramando nuevas reglas. Esto se explica más en detalle en la Sección 5.

3.7.4. Configuración de los LLM

LMStudio

Para ejecutar los modelos LLaVA 1.5 7B y Gemma 3 4B se ha empleado el software LMStudio. Se ha desplegado una API accesible desde la red local en el puerto 1234.

Al llamar a esta API y pasarle un prompt como cuerpo del mensaje, el modelo devuelve una respuesta. A continuación se muestra un ejemplo de llamada a la API en Python:

```
try:
    url = "http://localhost:1234/v1/chat/completions"
    headers = {
        "Content-Type": "application/json"
    }
    payload = {
        "model": "gemma-3-4b-it",
        "messages": [
            {
                "role": "user",
                "content": [
                    {
                        "type": "image_url",
                        "image_url": {
                            "url": f"data:image/jpeg;base64,{img_base64}"
                        }
                    },
                    {
                        "type": "text",
                        "text": "OBJETO: pelota"
                    }
                ]
            }
        ],
        "temperature": 0.2
    }
    response = requests.post(url, headers=headers, json=payload)
    response.raise_for_status()
    data = response.json()
```

Además, se han configurado los siguientes hiperparámetros desde LMStudio para trabajar con el LLM:

- **GPU Offload:** Número de capas del modelo que se procesan en la GPU. Configurado al máximo para reducir lo máximo posible el tiempo de inferencia.

- **Maximum output tokens:** Longitud máxima de la salida del modelo (en tokens). Limitada a 20, ya que las respuestas posibles no superan este número de tokens.
- **Temperatura:** A mayor temperatura, el modelo genera respuestas más “creativas” y menos repetitivas. En este caso en el que se quiere tener repetitividad (a situaciones similares, respuestas parecidas) se debe configurar con un valor en torno al 0 (establecido en 0.1).

OpenAI

Para trabajar con GPT-4o-mini, la forma de proceder ha sido llamar directamente a la API de OpenAI. En este caso, la computación se realiza en los servidores de OpenAI. Posteriormente se verá si esto tiene algún impacto en los tiempos.

El primer paso es acceder a la página de Playground de OpenAI, donde se configuran las APIs. En ella, se explica cómo llamarla desde un script de Python. El formato es muy parecido al anterior, aunque con ligeras diferencias:

```
from openai import OpenAI
client = OpenAI()

response = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=[],
    response_format={
        "type": "text"
    },
    temperature=0.1,
    max_completion_tokens=20,
    top_p=1,
    frequency_penalty=0,
    presence_penalty=0
)
```

Luego, se ajustan los hiperparámetros como en el apartado anterior. En este caso:

- **GPU Offload:** No es necesario, ya que por defecto se ejecuta el modelo en los servidores de OpenAI.
- **Maximum output tokens:** 20.
- **Temperatura:** 0.1.

Con esto, queda totalmente implementado el esquema de control. Los *scripts* de Python se encuentran en el **repositorio de GitHub** asociado al proyecto. En el siguiente capítulo se realizan algunos experimentos cuya finalidad es cuantificar el rendimiento del modelo, y evaluar si sirve o no para el propósito planteado.

4. Análisis de resultados

A continuación, se procede a realizar una serie de experimentos con el fin de entender de forma cuantitativa cómo se comportan cada uno de los modelos y poder compararlos.

Para los experimentos, se ha utilizado como objeto de referencia una pelota, pero se podría haber utilizado cualquier otro. Simplemente, en vez de indicar al modelo en el prompt que el objeto es una pelota, se le indicaría el objeto deseado.

4.1. Comparación de ambas estrategias

El primero de los experimentos tiene como objetivo determinar cuál de las estrategias planteadas en la Sección 3.3 funciona mejor (en términos de tiempo y resultados) para controlar el robot.

La segunda estrategia ha sido relativamente fácil de implementar: un código en el robot, que toma las fotos y las envía al portátil, y otro en el portátil que recibe las fotos, ejecuta el LLM y devuelve la respuesta.

Sin embargo, en cuanto a la primera estrategia, se han dado las siguientes complicaciones:

- No se ha conseguido obtener una estimación precisa de la profundidad. Esta era enormemente sensible a las condiciones de iluminación, por lo que al cambiar ligeramente estas el mapa de profundidad variaba en gran medida.
- YOLO no siempre detectaba los objetos en la imagen.

Estas limitaciones se analizan más detalladamente en la Sección 5.3. Por su parte, al aplicar la segunda estrategia, el LLM se comporta adecuadamente a la hora de realizar las tareas de detección y estimación de la profundidad. Por tanto, todos los experimentos siguientes se han realizado utilizando la segunda estrategia.

4.2. Análisis de tiempos de inferencia

Dada la aplicación de un robot en movimiento, se requiere que la respuesta del modelo sea rápida. Esto descarta el uso de modelos razonadores, típicamente más lentos, y es el principal motivo por el que se han usado LLMs de menos de 10 mil millones de parámetros.

Primeramente, se analiza el tiempo de inferencia para cada uno de los modelos empleados. A continuación se presenta una tabla con la comparación de los resultados obtenidos.

Tiempo de inferencia (s)	GPT-4o mini	Gemma 3 4B	LLaVA 1.5 7B
Medición 1	3.08	4.46	18.76
Medición 2	2.42	4.11	17.87
Medición 3	2.30	4.19	16.95
Medición 4	2.67	4.13	17.65
Medición 5	2.32	4.34	17.38

Tabla 4.1: Comparación de los tiempos de inferencia de los modelos

Como se puede observar, los dos primeros modelos (GPT-4o y Gemma 3) ofrecen tiempos de inferencia similares, muy inferiores a los de LLaVA. Para entender el porqué de esta diferencia, resulta conveniente observar las características de los modelos. Se comparará únicamente Gemma con LLaVA, ya que la arquitectura de los modelos de la familia GPT no es pública.

Se puede observar que el modelo que se está empleando de Gemma 3 tiene menos parámetros (4B) que el de LLaMA (7B). Esto posee un impacto directo en el tiempo de inferencia del modelo, pero no es el único factor.

En Gemma, el procesamiento de la imagen se hace mediante un encoder visual de tipo SigLIP, el cual reduce las salidas visuales a 256 vectores (Team et al. (2025)). En cambio en LLaVA, este preprocesamiento se hace con un encoder de tipo CLIP, que previamente escala la imagen a resoluciones más altas con el fin de obtener más detalles, y cuenta con un número mucho mayor de salidas visuales (del orden de 336^2). (Liu et al. (2024)). Esto hace que el costo computacional del procesamiento de la imagen sea mucho mayor en LLaVA que en Gemma.

Además, Gemma utiliza un tipo de atención llamado **GQA** (Grouped-Query Attention), que permite hacer las operaciones internas del modelo de forma más rápida y con menos uso de memoria, sin perder calidad en las respuestas. Esto ayuda a que el modelo procese imágenes y texto de manera más eficiente. LLaVA utiliza una arquitectura más sencilla basada en una conexión directa entre el encoder de imagen y el modelo de lenguaje, por lo que procesa más información visual y requiere más recursos para lograr resultados similares.

4.3. Análisis de tokens

Como se explicó anteriormente, los tokens son las unidades lingüísticas en las que se divide el prompt y la respuesta del modelo. Cada modelo cuenta con su propio tokenizador, si bien la estimación de token no suele variar significativamente de un modelo a otro.

Para observar los tokens se ha hecho uso del Playground de OpenAI, que recoge una gran cantidad de información acerca de las llamadas realizadas a la API. Se obtiene una estimación de 15100 tokens de entrada por cada llamada, mientras que los de salida son prácticamente 0.

Para cuantificarlo a nivel económico, se puede consultar en la web de OpenAI el precio de uso de la API. En el caso del modelo GPT-4o-mini, a fecha de 17 de junio de 2025, se

tiene un precio de 0.60\$ por millón de tokens. Teniendo en cuenta el número de tokens por cada llamada en nuestro proyecto, se tiene un coste aproximado de 0.01\$/llamada.

4.4. Análisis de consistencia

El objetivo de este experimento es determinar si la respuesta del modelo es predecible y repetitiva. Es importante que sea así puesto que en el control de un robot es esencial que se emitan señales de control similares frente a entradas parecidas. Es por esto que previamente se fijó la temperatura del modelo a un valor relativamente bajo.

Para esta prueba, se le han pasado 3 imágenes a los modelos GPT-4o-mini y Gemma, que son los que mejores resultados han dado en cuanto a tiempos y con los que se trabajará a partir de ahora. Cada una de las imágenes se le ha enviado 15 veces a cada modelo, limpiando totalmente la memoria caché para evitar posible influencia de las respuestas anteriores.

A continuación se muestran las imágenes que se le han pasado a los modelos:



(a) Primera imagen. Objeto no visible. (b) Segunda imagen. Objeto visible. (c) Tercera imagen. Objeto parcialmente visible.

Los resultados obtenidos son los siguientes:

Respuesta del modelo (Porcentaje de las veces)	GPT-4o mini	Gemma 3 4B
Imagen 1	EXPLORA (100 %)	EXPLORA (100 %)
Imagen 2	AVANZA (100 %)	AVANZA (100 %)
Imagen 3	AVANZA (100 %)	AVANZA (100 %)

Tabla 4.2: Comparación de la consistencia en las respuestas de los modelos

Se puede observar que ambos modelos son consistentes en sus respuestas, lo cual es un buen indicador de la fiabilidad de las mismas.

4.5. Búsqueda de objetos

Para finalizar, se han realizado una serie de experimentos colocando un objeto en algún punto de la casa y dejando al robot que lo busque. El vídeo con los experimentos se puede ver **[clicando aquí](#)**.

Adicionalmente, se describe el contenido del vídeo por si no se pudiera reproducir.

Inicialmente, se muestra el movimiento del robot sin que este sea controlado por el modelo de lenguaje. Primero se le pide que trace un cuadrado en el suelo para comprobar el correcto funcionamiento de los sensores y la programación de los mismos. Tras esto, se realiza una segunda prueba indicándole al robot que avance una distancia fija de 1,5 metros en línea recta.

Con los resultados satisfactorios de las pruebas anteriores, los siguientes experimentos que se muestran son con el robot siendo controlado por el LLM. Se muestra primero cómo funciona el bucle de control con un problema trivial en el que el robot simplemente debe avanzar en línea recta para alcanzar el objeto, el cual se trata de una pelota. Finalmente, se realiza un segundo experimento en el que el robot debe explorar el entorno para encontrar el objeto.

Los resultados de los dos experimentos que se muestran en el vídeo y de otros experimentos que no se han grabado se consideran satisfactorios: el robot ha cumplido su misión de encontrar el objeto. En la siguiente sección se extraen las conclusiones del proyecto de una manera más detallada.

5. Conclusiones

A continuación se presentan las conclusiones del proyecto. A modo de resumen, a la pregunta que da título al trabajo: ¿Sirve un LLM para controlar un robot móvil? se le puede dar la respuesta siguiente: **sí, para la mayoría de los casos.**

Los LLM poseen una gran cantidad de aplicaciones entre las que se encuentran analizar detalladamente la información sensorial recogida por el robot, entender instrucciones en lenguaje natural y adaptarse dinámicamente a escenarios cambiantes mediante razonamiento contextual.

A lo largo del proyecto se ha demostrado la validez de un modelo de lenguaje para asistir en tareas como interpretar comandos, analizar imágenes y generar instrucciones estructuradas que pueden ser interpretadas por un robot o cualquier otro dispositivo.

Sin embargo, también se pone de manifiesto que existen ciertas limitaciones. Mayormente, se trata de limitaciones computacionales. El uso de un LLM con su elevado número de parámetros requiere un gran número de operaciones que puede afectar al rendimiento del sistema, sobre todo en aplicaciones de tiempo real. A pesar de ello, siguiendo una serie de buenas prácticas como delegar la computación más pesada a un sistema externo y utilizar mecanismos eficientes de comunicación como WebSocket se puede minimizar este tiempo de espera y lograr una aplicación funcional.

5.1. Ventajas de utilizar un LLM como controlador de un robot

Trabajar con modelos de lenguaje hace posible una mayor adaptabilidad del robot a circunstancias cambiantes y a una gran variedad de situaciones diferentes.

Por ejemplo, una de los grandes pros es poder cambiar la funcionalidad del proyecto sin necesidad de reprogramarlo. Simplemente cambiando el prompt, el cual está escrito en lenguaje natural, se puede modificar la funcionalidad del robot incluso si no se tienen conocimientos de robótica o programación. Esto lo hace mucho más accesible, a que permite que personas sin formación técnica puedan interactuar con el sistema y adaptarlo a nuevas tareas o necesidades de forma intuitiva, rápida y flexible. Además, facilita la experimentación y el prototipado en entornos educativos o de investigación.

Otro aspecto positivo es la capacidad de aplicar un razonamiento avanzado, por ejemplo, cuando se detecta un obstáculo. Puede entender el contexto en el que se desarrolla la acción y en base a este tomar una acción, y tener en cuenta una gran variedad de factores que con un controlador “tradicional” sería mucho más tedioso implementar.

Además, la multimodalidad de los modelos utilizados permite combinar la parte de visión, lenguaje, entrada de sensores, etc. en una sola lógica de control. En la práctica, el LLM actúa como una caja negra, por lo que cuando este se ha probado y se ha evaluado su correcto funcionamiento, simplifica enormemente la estructura del código y del proyecto en general, reduciendo la necesidad de contar con varias capas (planificador de trayectoria, controlador...).

La escalabilidad es otra gran ventaja: frente a cambios en el entorno o la funcionalidad se puede adaptar a estas nuevas condiciones sin necesidad de un rediseño. Las opciones son mucho más amplias de lo que se aborda en este proyecto, como se verá en la sección 5.3.

5.2. Ventajas y desventajas de utilizar modelos en la nube frente a modelos en local

Una de las grandes tendencias de los últimos años en el tema de los modelos de lenguaje es ejecutarlos en la nube, de manera que cuando se envía un prompt el modelo lo ejecuta en los servidores de la compañía (OpenAI en el caso de ChatGPT, Google en el caso de Gemini...). Esto se debe a que estos servidores están dotados de hardware específico optimizado para minimizar el tiempo de inferencia. Así, por lo general se obtienen respuestas más rápidas con aquellos modelos que se ejecutan en la nube.

Por otro lado, ejecutar modelos en local tiene sus ventajas: elimina la necesidad de una conexión a Internet y por tanto es útil en entornos que no cuentan con una. Otro aspecto positivo es que teóricamente los datos no salen del equipo en el que se ejecuta el modelo.

Además, la ejecución local permite un mayor control sobre el entorno de ejecución y el comportamiento del modelo, ya que no depende de los cambios, actualizaciones o restricciones que puedan aplicar terceros proveedores en sus plataformas.

Otra ventaja destacable es la previsibilidad del rendimiento. Mientras que los modelos en la nube pueden experimentar variaciones debido a la carga del servidor o la calidad de la conexión, los modelos locales ofrecen un comportamiento más constante, lo cual es crucial en tareas robóticas en tiempo real.

5.3. Limitaciones y posibles contribuciones al proyecto

A continuación se valoran las limitaciones que presenta el proyecto y posibles líneas de mejora en futuras investigaciones.

5.3.1. Limitaciones de la primera estrategia

Como se mencionó previamente, no se ha conseguido obtener con precisión la profundidad del objeto. Esto se debe a que el mapa de disparidad (y por consiguiente, el de profundidad) introduce una gran cantidad de ruido y medidas imprecisas. A continuación, se deja una captura de pantalla como ejemplo, en la que la pelota se encuentra a unos 80

cm de la cámara:

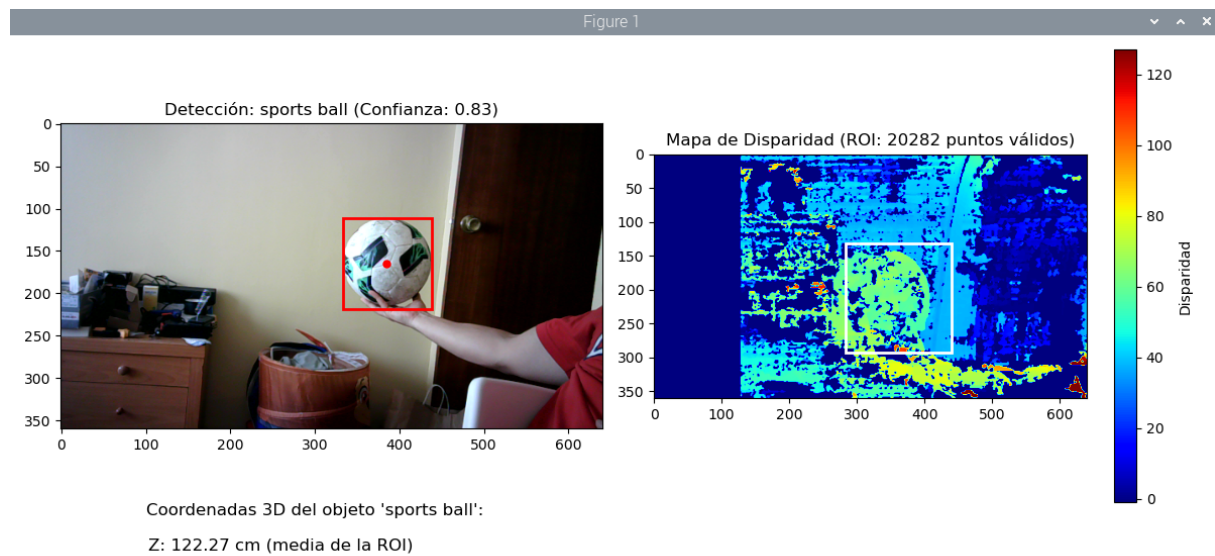


Figura 5.1: Ejemplo de mapa de profundidad con profundidad estimada incorrecta

Esto hace que los valores de profundidad resultantes no sean estimaciones fiables. Para mitigar esto, una de las estrategias podría ser emplear una cámara stereo de mejor calidad. Por ejemplo, la cámara ZED-mini de StereoLabs, que vale en torno a 400\$, es muy usada en proyectos de robótica (StereoLabs (2019), Barišić et al. (2022)). Otro sensor que se emplea a menudo en proyectos de investigación de robótica para obtener mapas de profundidad es el LiDAR.

Probablemente, con sensores de mejor calidad, ya sea por visión u otro tipo de tecnología, el mapa de profundidad resultante habría sido mejor y, por tanto, los valores de profundidad se corresponderían mejor con la realidad.

También se menciona anteriormente que YOLO no era capaz de detectar siempre los objetos que se pretendía. Esto ocurre porque se está utilizando la versión de YOLO entrenada con el dataset COCO. Si bien este dataset funciona bien para un gran número de categorías (80), aún hay muchos objetos que no puede detectar, o para los que no da buenos resultados.

Lo que se suele hacer en estos casos es hacer un *fine-tuning*, es decir, entrenar un modelo nuevo partiendo de la versión de YOLO entrenada con COCO, y añadiendo los objetos que se pretende detectar. Haciendo esto y usando el modelo *fine-tuneado* como modelo de detección los resultados serían mejores.

No obstante, se perdería en cierta parte esa “flexibilidad” de la que se hablaba antes, ya que para reconocer un objeto este tendría que haber sido entrenado previamente. En este sentido, se ha creído más conveniente avanzar con la segunda estrategia, siendo el LLM capaz de detectar una gran cantidad de objetos sin necesidad de un entrenamiento previo y aumentando por tanto la flexibilidad.

5.3.2. Otras limitaciones del proyecto

Además de las limitaciones presentadas arriba, se presentan las siguientes:

- **Tiempo de inferencia.** Aunque se ha hecho todo lo posible por minimizarlo, sigue siendo un cuello de botella en el proyecto. De la manera en que está planteado, el robot permanece quieto hasta que recibe la instrucción y cuando la ejecuta vuelve a esperar a la siguiente respuesta del modelo. Con un tiempo de ciclo más corto (acelerando la inferencia) se podría incluso hacer que el robot estuviera todo el tiempo en movimiento, sin tener que estar avanzando y parando continuamente mientras espera al modelo de lenguaje.

- **Consumo energético.** La batería que alimenta la Raspberry y por consiguiente todos los sensores (los motores están alimentados por una batería aparte) dura aproximadamente 1 hora encendida teniendo en cuenta los requisitos del proyecto. Una batería más grande ayudaría, pero se presentaría el problema de que el peso del robot empieza a ser demasiado grande para los motores. Por tanto, probablemente sería necesario cambiar también estos motores por otros de mayor potencia.

5.3.3. Trabajos futuros

Entre las posibles líneas de avance relativas a este proyecto, destacan:

- **Ir variando los objetos a buscar.** Por ejemplo, desde el ordenador que ejecuta el modelo enviando comandos a la Raspberry Pi con el siguiente objeto. Esto permitiría establecer rutas, buscar varios objetos simultáneamente...

- **Tener un mapa del lugar y explorar de forma eficiente por ese mapa,** teniendo en cuenta también las zonas del mapa que se han visitado. Esto resultaría en búsquedas más eficientes, evitando pasar varias veces por el mismo punto y agilizando el proceso de búsqueda.

- **Implementar un seguimiento continuo,** en el que el robot no solo sea capaz de ver objetos en el entorno y moverse hacia donde se encuentran, sino que sea capaz de deducir la velocidad a la que se mueven (si se mueven) y planificar la trayectoria a seguir para alcanzarlos. Esto implicaría integrar técnicas de detección y seguimiento en tiempo real.

- **Guardar un registro del entorno explorado y los objetos encontrados.** Por ejemplo, en un mapa con anotaciones semánticas, de forma que se pueda acceder a este mapa para futuras búsquedas.

Bibliografía

- Baeldung (2021). Disparity map in stereo vision. Consultado el 12 de junio de 2025 en: <https://www.baeldung.com/cs/disparity-map-stereo-vision>.
- Barišić, A., Petric, F., and Bogdan, S. (2022). Brain over brawn: Using a stereo camera to detect, track, and intercept a faster UAV by reconstructing the intruder’s trajectory. *Field Robotics*, 2:222–240.
- Barturen, J. (2023). Google incorpora la IA generativa a los algoritmos de aprendizaje de sus robots. *Robotics Lab*.
- Brohan, A., Chebotar, Y., Finn, C., Hausman, K., Herzog, A., Ho, D., Ibarz, J., Irpan, A., Jang, E., Julian, R., et al. (2023). Do as i can, not as i say: Grounding language in robotic affordances. In *Conference on Robot Learning*, pages 287–318. PMLR.
- Choi, D., Fung, A., Wang, H., and Tan, A. H. (2024). Find everything: A general vision language model approach to multi-object search. *arXiv preprint arXiv:2410.00388*.
- Fette, I. and Melnikov, A. (2011). RFC 6455: The WebSocket protocol. Consultado el 12 de junio de 2025 en: <https://datatracker.ietf.org/doc/html/rfc6455>.
- Hartley, R. and Zisserman, A. (2003). *Multiple view geometry in computer vision*. Cambridge University Press.
- Hughes, A. and Drury, B. (2019). *Electric motors and drives: Fundamentals, types and applications*. Newnes.
- Jiménez, J. (2025). LLM-Robotics. Creado en Junio de 2025, disponible en <https://github.com/jeronimo-jimenez-mateos/LLM-Robotics>.
- Josefsson, S. (2006). The base16, base32, and base64 data encodings. Technical Report RFC 4648, IETF.
- Knospe, C. (2006). PID control. *IEEE Control Systems Magazine*, 26(1):30–31.
- Łasocha, W. and Badurowicz, M. (2021). Comparison of WebSocket and HTTP protocol performance. *Journal of Computer Sciences Institute*, 19:67–74.
- Liu, H., Li, C., Li, Y., and Lee, Y. J. (2024). Improved baselines with visual instruction tuning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 26296–26306.

- Marvin, G., Nakayiza, H., Jjingo, D., and Nakatumba-Nabende, J. (2023). Prompt engineering in large language models. In *International Conference on Data Intelligence and Cognitive Informatics*, pages 387–402.
- Mohd-Yasin, F., Korman, C. E., and Nagel, D. J. (2003). Measurement of noise characteristics of MEMS accelerometers. *Solid-State Electronics*, 47(2):357–360.
- Pi, R. (2025). Raspberry Pi 4 Pinout. Consultado el 25 de junio de 2025 en: <https://www.raspberrypi.com/documentation/computers/raspberry-pi.html>.
- Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 779–788.
- Shah, D., Osiński, B., Levine, S., et al. (2023). LM-Nav: Robotic navigation with large pre-trained models of language, vision, and action. In *Conference on Robot Learning*, pages 492–504. PMLR.
- StereoLabs (2019). ZED/ZED Mini for FIRST Robotics Competition 2019. Consultado el 21 de junio de 2025 en: <https://www.stereolabs.com/en-es/blog/zed-zed-mini-for-first-robotics-competition-2019>.
- Team, G., Kamath, A., Ferret, J., Pathak, S., Vieillard, N., Merhej, R., Perrin, S., Matejovicova, T., Ramé, A., Rivière, M., et al. (2025). Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*.
- Verschuren, T. A. C. (2006). Extracting more accurate position and velocity estimations using time stamping. *DCT rapporten*, 2006.058.
- Wang, R., Yang, Z., Zhao, Z., Tong, X., Hong, Z., and Qian, K. (2024). LLM-based robot task planning with exceptional handling for general purpose service robots. In *2024 43rd Chinese Control Conference (CCC)*, pages 4439–4444.
- Wellons, C. (2013). Writing a toy WebSocket server from scratch. <https://nullprogram.com/blog/2013/09/05/>. Consultado el 12 de junio de 2025 en: <https://alexanderell.is/posts/websockets-from-scratch/>.
- Zheng, L., Mei, R., Zou, B., Zhao, Z., Wei, M., and Xu, S. (2025). GMM-searcher: Efficient object search in large-scale scenes using large language models. *Scientific Reports*, 15(1):1–14.
- Zitkovich, B., Yu, T., Xu, S., Xu, P., Xiao, T., Xia, F., Wu, J., Wohlhart, P., Welker, S., Wahid, A., et al. (2023). RT-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning*, pages 2165–2183.