TERRY:
Coche con control remoto y modo autónomo de seguimiento de líneas

Proyecto final de Sistemas Electrónicos

Alejandro Alcázar Mendoza
(Grado en Ingeniería Matemática e Inteligencia Artificial)

Jorge González Pérez
(Grado en Ingeniería Matemática e Inteligencia Artificial)

Curso 2024–2025

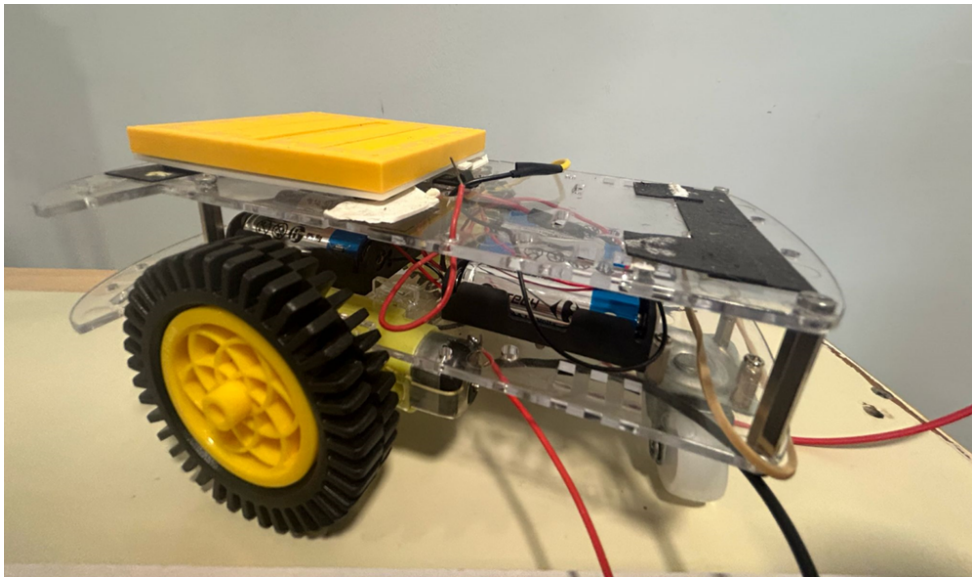

Figura 1: Prototipo de TERRY sobre el chasis proporcionado por el laboratorio.

Resumen

En este trabajo se presenta el diseño y desarrollo de **TERRY**, un pequeño vehículo basado en una Raspberry Pi que dispone de dos modalidades principales de funcionamiento: un modo de *control remoto* mediante un mando inalámbrico y un modo *autónomo* capaz de seguir una línea negra impresa en el suelo. El sistema integra sensores de infrarrojos para la detección de línea, sensores de ultrasonidos para la detección de obstáculos, motores de corriente continua controlados mediante PWM y una interfaz de comunicación inalámbrica.

La Raspberry Pi actúa como núcleo de procesamiento, ejecutando una lógica basada en módulos de software que se encargan de la lectura de sensores, la interpretación de las entradas del mando y el control de los motores. Además, se implementa un panel de control web para monitorizar en tiempo real el estado del coche. Se describen la arquitectura hardware y software, los distintos modos de operación, las especificaciones iniciales del proyecto, así como los principales problemas encontrados durante la implementación y las soluciones adoptadas.

Índice

1. Introducción y planteamiento del problema **3**

2. Objetivos **3**

3. Especificaciones iniciales del proyecto **4**
- 3.1. Requisitos técnicos . . . 4
- 3.2. Limitaciones y simplificaciones . . . 5
- 3.3. Software y configuración . . . 5

4. Descripción general del sistema **6**
- 4.1. Resumen funcional . . . 6
- 4.2. Arquitectura general . . . 6

5. Diseño hardware **7**
- 5.1. Listado de componentes . . . 7
- 5.2. Sensores de ultrasonidos . . . 7
- 5.3. Sensores de infrarrojos . . . 8
- 5.4. Motores y tracción . . . 10

6. Diseño software **11**
- 6.1. Organización general del código . . . 11
- 6.2. Módulo de control inalámbrico . . . 11
- 6.3. Módulo de detección de obstáculos . . . 12
- 6.4. Módulo de control de dirección . . . 12
- 6.5. Módulo de control de motores . . . 13
- 6.6. Modo exhibición y panel web . . . 13
- 6.7. Scripts de automatización . . . 14
- 6.8. Diseño completo . . . 14
- 6.9. Implementación mediante máquina de estados . . . 15

7. Problemas detectados durante el desarrollo **15**

8. Soluciones adoptadas y mejoras propuestas **15**

9. Conclusiones **16**

1. Introducción y planteamiento del problema

Dentro de la asignatura de *Sistemas Electrónicos* se plantea el desarrollo de un proyecto práctico que combine el uso de sensores, actuadores y procesamiento embebido sobre una placa de cómputo como la Raspberry Pi.

El proyecto **TERRY** tiene como objetivo construir un coche autónomo con dos modos de operación bien diferenciados:

- **Modo sigue líneas:** el coche sigue una trayectoria marcada en el suelo mediante una cinta negra sobre fondo blanco.
- **Modo control remoto:** el usuario maneja el coche manualmente mediante un mando inalámbrico conectado a la Raspberry Pi vía Bluetooth.

En ambos modos, el sistema cuenta con:

- **Sensores IR** (infrarrojos) para detección de la línea en el suelo.
- **Sensores ultrasónicos** para medir la distancia a posibles obstáculos frontales.
- **Motores DC** con control PWM para el movimiento y la dirección.
- Un **buzzer** (no finalmente integrado físicamente) para emitir señales acústicas.
- Una **interfaz de comunicación inalámbrica** para el mando y, potencialmente, para monitorización por Wi-Fi.

La Raspberry Pi actúa como *unidad central de control*, gestionando los datos de sensores y actuadores mediante una arquitectura modular programada en Python. Sobre esta base se han construido tanto el comportamiento autónomo (sigue líneas) como el modo de control remoto (mando + detección de obstáculos).

Adicionalmente, se ha desarrollado un *panel de control web* que permite monitorizar el estado del vehículo y sirve como interfaz de exhibición del sistema.

2. Objetivos

Los objetivos principales del trabajo son los siguientes:

- Diseñar e implementar la electrónica necesaria para dotar a un coche de sensores de **ultrasonidos** e **infrarrojos**, así como el control de dos motores de tracción independientes.
- Desarrollar en Python los **drivers de bajo nivel** para la lectura de sensores y el control de los motores mediante PWM.
- Implementar una **lógica de alto nivel** que permita:
  a) el control manual mediante un mando inalámbrico,
  b) el seguimiento de líneas negro/blanco,
  c) y la detección básica de obstáculos tanto en modo manual como en modo automático.
- Crear una **interfaz web**, basada en Flask, para visualizar en tiempo real el estado de TERRY (sensores, motores, batería, etc.).
- Automatizar tareas de arranque y configuración mediante **scripts de Bash** para cada modo de funcionamiento.
- Identificar los **problemas prácticos** más relevantes (tolerancias mecánicas, corrección de trayectorias, conflicto entre sensores, etc.) y proponer soluciones.

3. Especificaciones iniciales del proyecto

En esta sección se resumen las especificaciones establecidas en la propuesta inicial del proyecto, que han guiado el diseño posterior de TERRY.

3.1. Requisitos técnicos

Sensores y detección

- **Sensores infrarrojos:** al menos 2 sensores (tipo TCRT5000 o similares) para detectar la línea en el suelo. La lectura debe ser lo suficientemente rápida como para permitir correcciones frecuentes de la trayectoria.
- **Sensores ultrasónicos (HC-SR04):** 2 sensores frontales para la detección de obstáculos y medida de distancia. Su información se utiliza tanto en modo manual como automático para frenar o detener el coche si hay un objeto cercano.
- **Frecuencia de muestreo:** se plantea una frecuencia de muestreo de los sensores de al menos 50 Hz para lograr una respuesta fluida.
- **Buzzer:** dispositivo acústico previsto para señales de aviso (por ejemplo, ante la presencia de un obstáculo o como claxon). Por limitaciones de corriente no se integró finalmente en el montaje, pero se incluyó en el diseño inicial.
- **Modo manual:** los sensores ultrasónicos se emplean para frenar el coche de manera automática si hay un obstáculo dentro de un umbral de distancia.
- **Modo sigue líneas:** los sensores IR se utilizan para seguir la línea negra. Los ultrasónicos actúan de forma similar al modo manual, deteniendo o frenando el coche si se detecta un obstáculo en el recorrido.

Control de motores

- **Motores DC:** dos motores para la tracción del coche, uno por cada rueda motriz.
- **Control PWM:** la velocidad se controla mediante PWM con una frecuencia mínima de 1 kHz, para conseguir una aceleración suave y evitar vibraciones perceptibles.
- **Control de dirección y velocidad:** tanto en modo automático como manual, el control debe ser suficiente para evitar desviaciones excesivas y colisiones, ajustando la velocidad y el giro de cada motor de manera independiente.
- **Mando inalámbrico:** se utiliza un mando (*Pro Controller*) para controlar la dirección y la velocidad en modo manual.

Modos de operación

- **Modo sigue líneas:** el coche ajusta su dirección en función de los sensores IR, siguiendo una línea negra sobre superficie clara.
- **Modo control remoto:** se reciben las señales del mando vía Bluetooth para controlar dirección y velocidad.
- **Cambio de modo:** inicialmente se plantea realizarlo mediante un botón en el mando o físico en el coche. En la implementación final se opta por scripts de arranque diferenciados para cada modo.

Interfaz de comunicación inalámbrica

- **Bluetooth:** conexión con el mando mediante la Raspberry Pi.
- **Posible extensión Wi-Fi:** prevista para monitorización remota del estado del coche (en parte materializada mediante el panel web).

Otras funcionalidades opcionales

- **LED RGB:** para indicar visualmente el estado del coche (modo actual, presencia de obstáculos, etc.).
- **Acelerómetro/giroscopio (MPU6050 o similar):** para medir inclinación o detectar colisiones.
- **Sensor de luz (LDR):** para adaptar el comportamiento del coche a la iluminación ambiental.
- **Pantalla OLED o LCD:** para mostrar información como velocidad o modo actual.
- **Pulsadores físicos:** para cambiar de modo manualmente sin depender del mando.

En la implementación actual de TERRY, estas extensiones se han dejado como *mejoras futuras*.

3.2. Limitaciones y simplificaciones

Para acotar el alcance del proyecto se adoptan varias simplificaciones:

- Se utiliza una **línea de color negro** sobre fondo blanco, en lugar de emplear líneas de distintos colores con significados diferentes (por ejemplo, verde para salida, rojo para parada, etc.). La parada se establece mediante una **línea negra horizontal** donde ambos sensores IR detectan simultáneamente negro.
- El sistema se prueba sobre **superficies planas**, sin considerar terrenos irregulares o fuertes pendientes.
- No se considera la detección avanzada de cruces o bifurcaciones complejas: la lógica se centra en seguir una pista relativamente simple y claramente contrastada.

3.3. Software y configuración

Desde la propuesta inicial se prevé el uso de las siguientes librerías de Python en la Raspberry Pi:

- **RPi.GPIO**: para el control directo de pines GPIO, especialmente en el control de motores.
- **gpiozero**: para facilitar el acceso a sensores (ultrasonidos, entradas digitales, etc.).
- **Librerías para el mando**: en la propuesta se contempla el uso de `pyPS4Controller`, `pygame` o `evdev`. En la implementación final se utiliza `evdev` para leer eventos del *Pro Controller*.

4. Descripción general del sistema

4.1. Resumen funcional

El sistema final se puede describir como un **coche teledirigido inteligente**:

- En modo mando, el usuario gobierna la dirección y la velocidad mediante un *Pro Controller*, mientras los sensores de ultrasonidos (*"eyes"*) detectan obstáculos cercanos y pueden provocar un frenado automático.
- En modo sigue líneas, el coche utiliza sensores de infrarrojos apuntando al suelo para distinguir entre superficie blanca y línea negra, y corrige la dirección de los motores para mantener la trayectoria. En paralelo, los sensores de ultrasonidos siguen activos para detener el vehículo ante objetos inesperados.

La Raspberry Pi ejecuta el código Python que integra las lecturas de sensores, el estado del mando y la generación de las señales PWM que controlan los motores. En paralelo, un servidor web Flask proporciona una interfaz visual para observar el estado del sistema.

4.2. Arquitectura general

La arquitectura global se compone de los siguientes bloques:

- **Unidad de control:** Raspberry Pi, responsable de ejecutar el código de alto nivel (Python) y de gestionar los periféricos.
- **Sensores:**
  - Dos **sensores de ultrasonidos**, integrados en la clase `Ultrasonidos`, para la detección de obstáculos frontales.
  - Dos **sensores de infrarrojos** conectados a un conversor analógico-digital `MCP3008`, encapsulados en la clase `Infrarrojos`, para distinguir entre superficie clara y línea oscura.
- **Actuadores:**
  - Dos motores de corriente continua, controlados mediante **PWM por hardware** para ajustar su velocidad y sentido de giro.
  - Un **zumbador** (`Buzzer`) previsto en el diseño para señales acústicas, que finalmente no se montó por limitaciones de corriente disponibles.
- **Interfaz de usuario:**
  - Un mando **Pro Controller**, gestionado mediante la clase `ProControllerReader` usando la librería `evdev`.
  - Un **panel web**, servido por Flask, que muestra en tiempo real el estado de sensores y motores.

5. Diseño hardware

5.1. Listado de componentes

De acuerdo con la propuesta inicial y el montaje final, los componentes principales son:

Componentes imprescindibles

- Raspberry Pi (control central del sistema).
- Sensores IR (2 unidades, tipo TCRT5000 o similar) para detección de la línea.
- Sensores ultrasónicos HC-SR04 (2 unidades) para detección de obstáculos.
- Motores DC (2 unidades) para la tracción.
- Tres bornes enchufables para alimentar motores y Raspberry.
- Buzzer (activo o pasivo) para señales acústicas (no montado finalmente).
- Mando inalámbrico (*Pro Controller*) para control remoto.
- Baterías para la alimentación del coche.
- Chasis, soporte y ruedas, proporcionados por el laboratorio.
- Cableado y protoboard para el montaje de sensores y actuadores.

En el momento de la propuesta se disponía ya de: sensores ultrasónicos, chasis, baterías, ruedas, protoboard, motores, zumbador, dos bornes enchufables y cableado extra, así como de un sensor IR adicional aportado por uno de los integrantes. Faltaba por adquirir un borne enchufable y otro sensor infrarrojo.

Componentes adicionales potenciales

- Acelerómetro/giroscopio (MPU6050 o similar) para medir inclinación o detectar colisiones.
- Sensor de luz (LDR) para adaptar el comportamiento a la iluminación ambiental.
- LEDs RGB para indicar distintos estados del coche.
- Pantalla OLED o LCD para mostrar información relevante.
- Pulsadores físicos para cambiar de modo manualmente.

5.2. Sensores de ultrasonidos

Los sensores de ultrasonidos HC-SR04 se han empleado para detectar la presencia de obstáculos dentro de un cierto rango de distancias. Cada sensor se alimenta a 5 V y proporciona una señal de salida también de 5 V, lo que obliga a adaptar el nivel lógico para su lectura segura desde la Raspberry Pi (3,3 V).

Divisor de tensión para la señal de eco

Para adaptar la salida de 5 V a 3,3 V se diseña un divisor de tensión formado por dos resistencias R_1 y R_2. Imponiendo una corriente del orden de 1 mA y una tensión de salida de aproximadamente 3,3 V, se obtiene la condición:

$$R_1 + R_2 \approx 5\,\text{k}\Omega, \qquad V_o = V_{cc}\frac{R_2}{R_1 + R_2} \approx 3{,}3\,\text{V}.$$

De esta relación se llega a valores ideales $R_2 \approx 3{,}3\,\text{k}\Omega$ y $R_1 \approx 1{,}7\,\text{k}\Omega$. Para ajustarse a valores comerciales disponibles en el laboratorio, se eligen:

$$R_1 = 2{,}2\,\text{k}\Omega, \qquad R_2 = 3{,}3\,\text{k}\Omega,$$

lo que reduce la salida a unos 3 V, dentro del margen seguro para la Raspberry Pi.

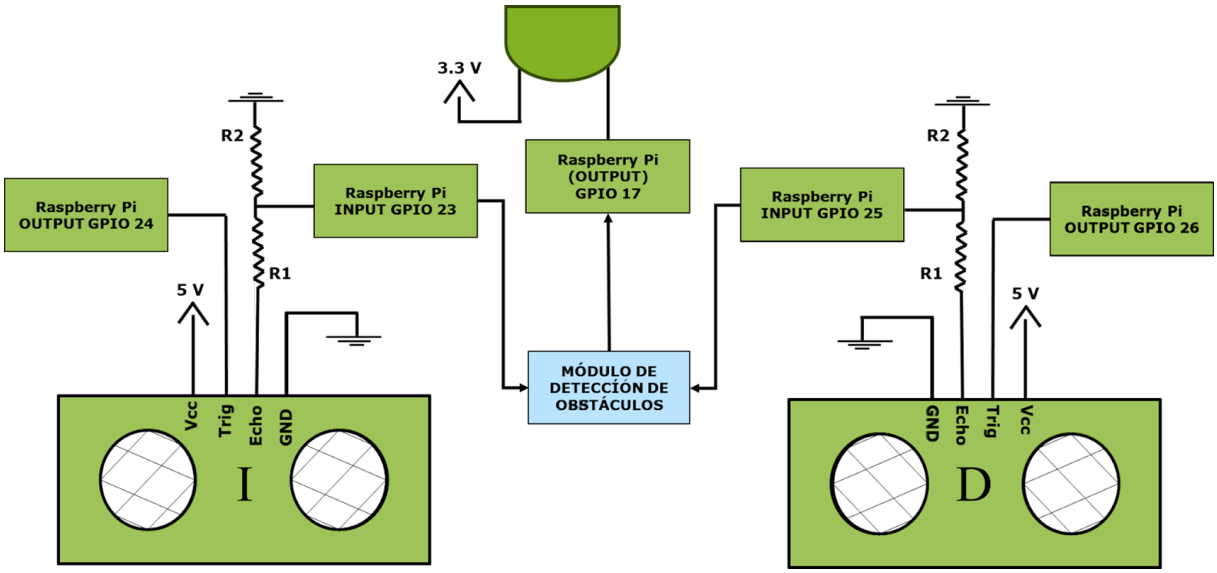

Figura 2: Esquema conceptual del divisor de tensión para la señal de eco del HC-SR04.

La clase `Ultrasonidos` se construye sobre la clase `DistanceSensor` de `gpiozero` e incorpora un método `check_distance` que compara la distancia medida con un umbral `threshold_distance`. El estado se codifica como:

- `0`: no hay objeto cerca,
- `1`: hay un objeto dentro del umbral de distancia.

Estos sensores se combinan en la clase `Eyes`, que fusiona la información de los dos dispositivos y ofrece un estado global del frente del coche para la lógica de control.

5.3. Sensores de infrarrojos

Para el modo de seguimiento de líneas se han utilizado sensores de infrarrojos apuntando hacia el suelo. El sensor IR del que se dispone se alimenta a 3,3 V, y su salida varía entre:

$$0\,\text{V} \quad \text{(superficie clara)} \qquad \text{y} \qquad 3{,}3\,\text{V} \quad \text{(superficie negra)}.$$

Detección por umbral (entrada analógica)

Una primera opción consiste en conectar los sensores IR a un *canal analógico* a través de un conversor ADC (en este caso, `MCP3008`) y determinar experimentalmente un umbral a partir del cual se considera que se ha detectado el color negro.

En la implementación final:

- La clase `Infrarrojos` encapsula la lectura analógica y compara con un umbral configurado.
- El método `check_black` actualiza el estado del sensor:
  - `0`: el suelo no es negro,
  - `1`: el suelo es negro.

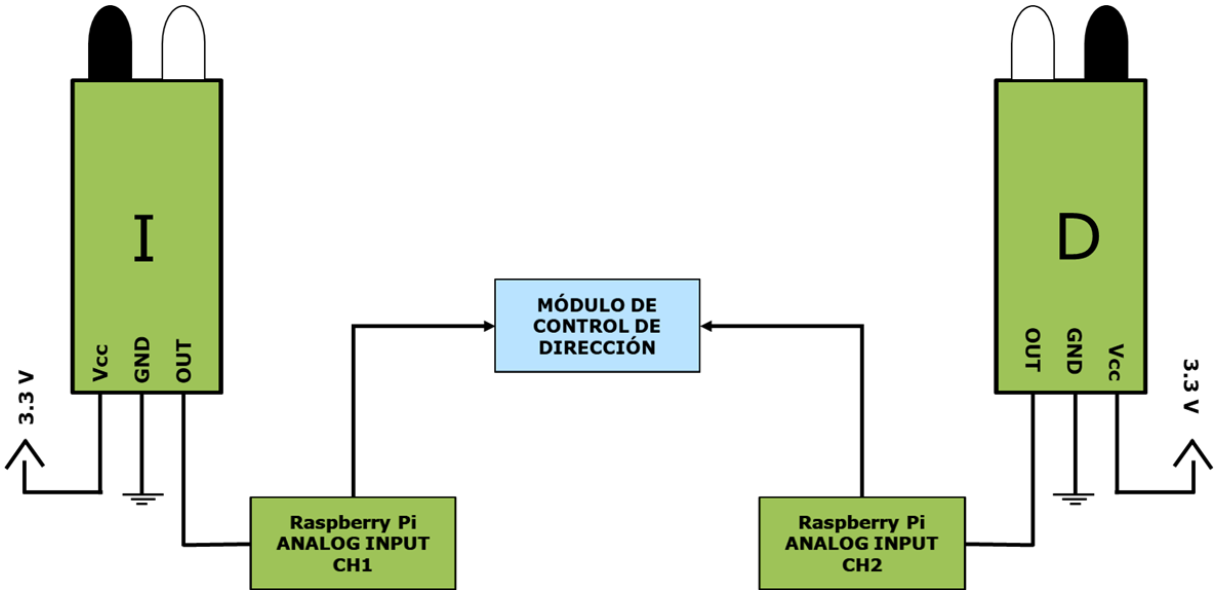

Figura 3: Esquema de implementación final.

Comparador hardware (opción alternativa)

La propuesta incluía también una alternativa puramente hardware: usar un amplificador operacional en configuración de *comparador*, de modo que la salida sea directamente digital (0 V o 3,3 V) cuando se supera un valor de referencia fijado mediante un divisor de tensión. Esta solución simplificaría el software, pero requiere componentes adicionales y no se llegó a implementar en el prototipo final.

Combinación de sensores IR: clase `Lines`

La clase `Lines` combina la información de los dos sensores IR (izquierdo y derecho) y determina un estado simbólico simplificado:

- `STRAIGHT`: ninguno de los sensores ve negro.
- `LEFT`: el sensor izquierdo ve línea y el derecho no.
- `RIGHT`: el sensor derecho ve línea y el izquierdo no.
- `BOTH`: ambos detectan línea (casos de parada o bifurcación).

Este estado se utiliza directamente en el módulo de *control de dirección* del software.

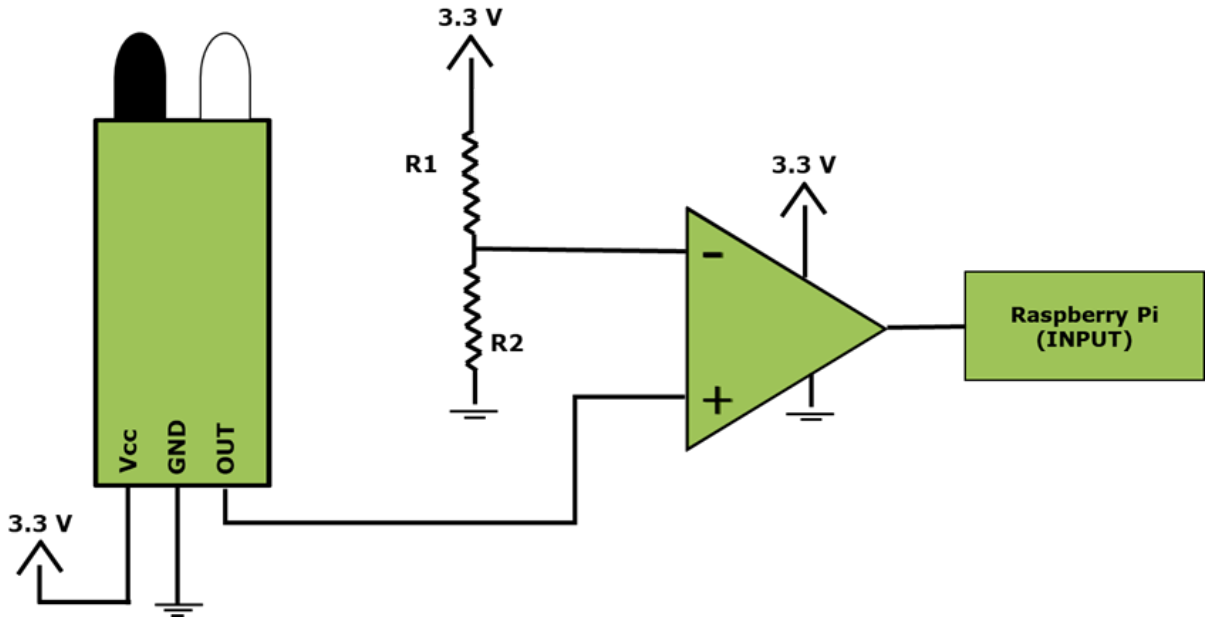

Figura 4: Esquema conceptual de detección de línea por comparador con referencia fija.

5.4. Motores y tracción

El vehículo dispone de dos motores DC independientes integrados en el chasis. La conexión eléctrica se realiza mediante la placa ya incorporada en el coche:

- Motor derecho conectado al borne etiquetado como `motor 1`.
- Motor izquierdo conectado al borne etiquetado como `motor 2`.
- Alimentación de los motores a través de las baterías del chasis.

La Raspberry Pi controla los motores mediante señales PWM de alta frecuencia (al menos 1 kHz) generadas sobre los pines GPIO y aplicadas a las entradas de la placa de potencia. Las clases `Motors`, `Motors_eyes` y `Motors_lines` contienen la lógica de bajo nivel para:

- arrancar y detener los motores,
- ajustar la velocidad de cada rueda de forma independiente,
- invertir el sentido de giro (marcha adelante/atrás),
- introducir un **factor de corrección** para compensar desequilibrios mecánicos entre motores.

La necesidad de este factor de corrección se debe a diferencias de fabricación y montaje entre las ruedas y los motores, que provocan desviaciones en la trayectoria si ambos reciben exactamente la misma señal PWM.

6. Diseño software

El diseño del software se inspira directamente en la estructura de módulos descrita en la propuesta original (módulo de control inalámbrico, detección de obstáculos, control de dirección y control de motores) y se materializa en una organización en paquetes Python.

6.1. Organización general del código

El código se organiza en varios directorios:

- `drivers/`: clases que representan los componentes de bajo nivel:
  - `clase_infrarrojo.py`, `clase_ultrasonidos.py`: adaptadores de sensores IR y ultrasónicos.
  - `eyes.py`, `lines.py`: abstracciones de alto nivel para combinar sensores.
  - `motors.py`, `motors_eyes.py`, `motors_lines.py`: control de motores en distintos modos.
  - `controller.py`: lectura de eventos del *Pro Controller*.
  - `car.py`, `car_eyes.py`, `car_lines.py`: lógica específica de cada modo de funcionamiento.
- `web/`: servidor Flask y plantilla HTML para el panel de control.
- `main.py`, `main_eyes.py`, `main_lines.py`: puntos de entrada a los distintos modos.
- `scripts de Bash`: automatizan el arranque y la selección de modo.

A continuación se describen los módulos funcionales principales, siguiendo la estructura de la propuesta (punto 4).

6.2. Módulo de control inalámbrico

Este módulo se encarga de la **conexión con el mando por Bluetooth** para controlar el coche en modo manual y, eventualmente, para cambiar de modo. La idea planteada en la propuesta es que el coche pueda arrancar en modo automático por defecto, acercarse al inicio de la línea y, mediante una pulsación en el mando, cambiar a modo sigue líneas.

En la implementación actual: La clase `ProControllerReader` utiliza la librería `evdev` para leer eventos del dispositivo de entrada del mando. Se mapean botones y direcciones a acciones abstractas: avanzar, frenar, girar a izquierda/derecha, cambiar velocidad, etc. Los archivos `main_eyes.py` y `car_eyes.py` usan esta información para actualizar continuamente el estado de los motores.

Figura 5: Módulo de control inalámbrico

6.3. Módulo de detección de obstáculos

Este módulo se encarga de **vigilar la presencia de obstáculos** mediante los sensores ultrasónicos y de comunicar al módulo de control de motores que debe frenar o detener el coche para evitar colisiones.

Características principales:

- Se ejecuta tanto en modo manual como automático.
- La clase `Eyes` fusiona la información de los dos sensores.
- Cuando la distancia medida cae por debajo de un umbral, se genera una orden de `STOP` o se reduce la velocidad.

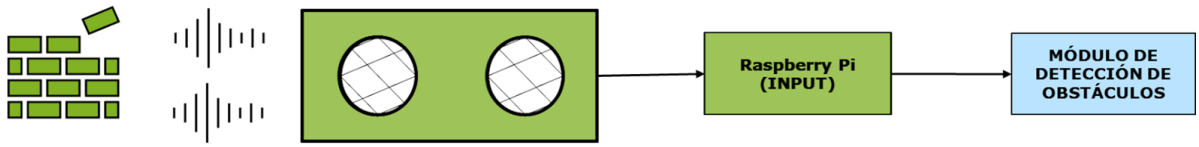

Figura 6: Módulo de detección de obstáculos

6.4. Módulo de control de dirección

El módulo de control de dirección es clave en el **modo sigue líneas**. Su objetivo es mantener la línea negra centrada bajo el coche a lo largo de todo el recorrido. Para ello se utilizan dos sensores IR, uno a cada lado de la parte frontal.

Tal y como se recoge en la propuesta, se consideran cuatro casos básicos:

A.) **Ningún sensor detecta negro.**
Ambos sensores ven suelo blanco. La instrucción es seguir hacia delante sin desviarse.

B.) **El sensor derecho detecta negro.**
Solo el sensor derecho ve negro, lo que indica que la línea se encuentra a la derecha del coche. La instrucción es girar hacia la derecha (aumentando velocidad del motor izquierdo y/o reduciendo la del derecho).

C.) **El sensor izquierdo detecta negro.**
Solo el sensor izquierdo ve negro, lo que indica que la línea está a la izquierda. La instrucción es girar hacia la izquierda.

D.) **Ambos sensores detectan negro.**
Ambos sensores ven línea negra simultáneamente. Esta condición se interpreta como **parada** (por ejemplo, final de trayecto) y se detiene el coche.

En el código, estos cuatro casos se representan mediante los estados `STRAIGHT`, `RIGHT`, `LEFT` y `BOTH` de la clase `Lines`, que son consumidos por la lógica de control de motores en `car_lines.py`.

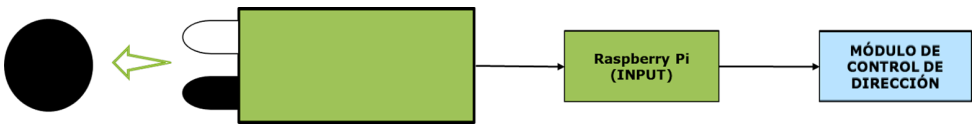

Figura 7: Módulo de control de dirección

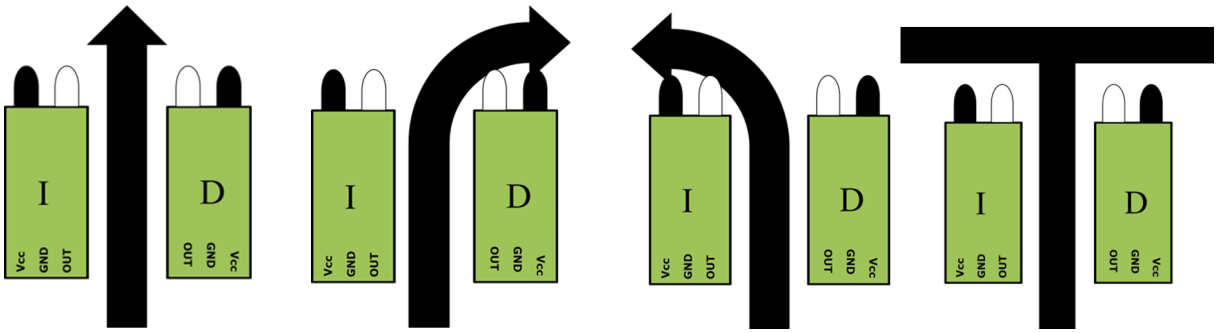

Figura 8: Control de infrarrojos

6.5. Módulo de control de motores

El módulo de control de motores recibe **instrucciones abstractas** de los módulos anteriores (control inalámbrico, obstáculos, dirección) y las traduce en señales PWM concretas para cada motor.

En la propuesta se menciona el uso de `RPi.GPIO`; en la práctica se emplea una combinación de `RPi.GPIO` y `gpiozero`, según la conveniencia, para:

- Configurar los pines GPIO como salidas PWM.
- Ajustar el ciclo de trabajo de cada motor según la velocidad deseada.
- Implementar funciones de `forward`, `backward`, `turn_left`, `turn_right` y `stop`.
- Aplicar factores de corrección individuales para cada motor.

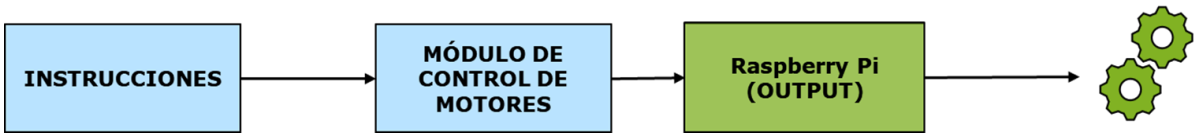

Figura 9: Módulo de control de motores

6.6. Modo exhibición y panel web

El archivo `main.py` lanza un objeto `Car` (definido en `drivers/car.py`) y arranca el servidor web Flask definido en `web/app.py`. El servidor expone:

- la página principal `/`, que sirve `index.html`,
- varios endpoints JSON (por ejemplo, `/get_state`, `/get_battery`) que permiten consultar el estado de sensores, motores y batería.

El panel web muestra, entre otros:

- la velocidad de cada motor,
- el estado de los sensores de ultrasonidos e infrarrojos,
- información sobre la conexión del mando y la batería.

Este modo se ha utilizado principalmente como **modo de exhibición**, en el que el vehículo se mantiene estacionario o se realizan movimientos controlados para ilustrar el funcionamiento de los sensores y la arquitectura del sistema.

6.7. Scripts de automatización

Para facilitar el uso del sistema, se han desarrollado varios scripts de Bash:

- `terry.sh`: lanza el modo exhibición (panel web).
- `terry_eyes.sh`: ejecuta el modo mando + obstáculos.
- `terry_lines.sh`: inicia el modo sigue líneas.

Estas utilidades se encargan de:

- activar el entorno virtual de Python,
- lanzar el programa correspondiente,
- configurar la conexión con el mando (cuando sea necesario).

6.8. Diseño completo

El esquema completo del diseño es el siguiente:

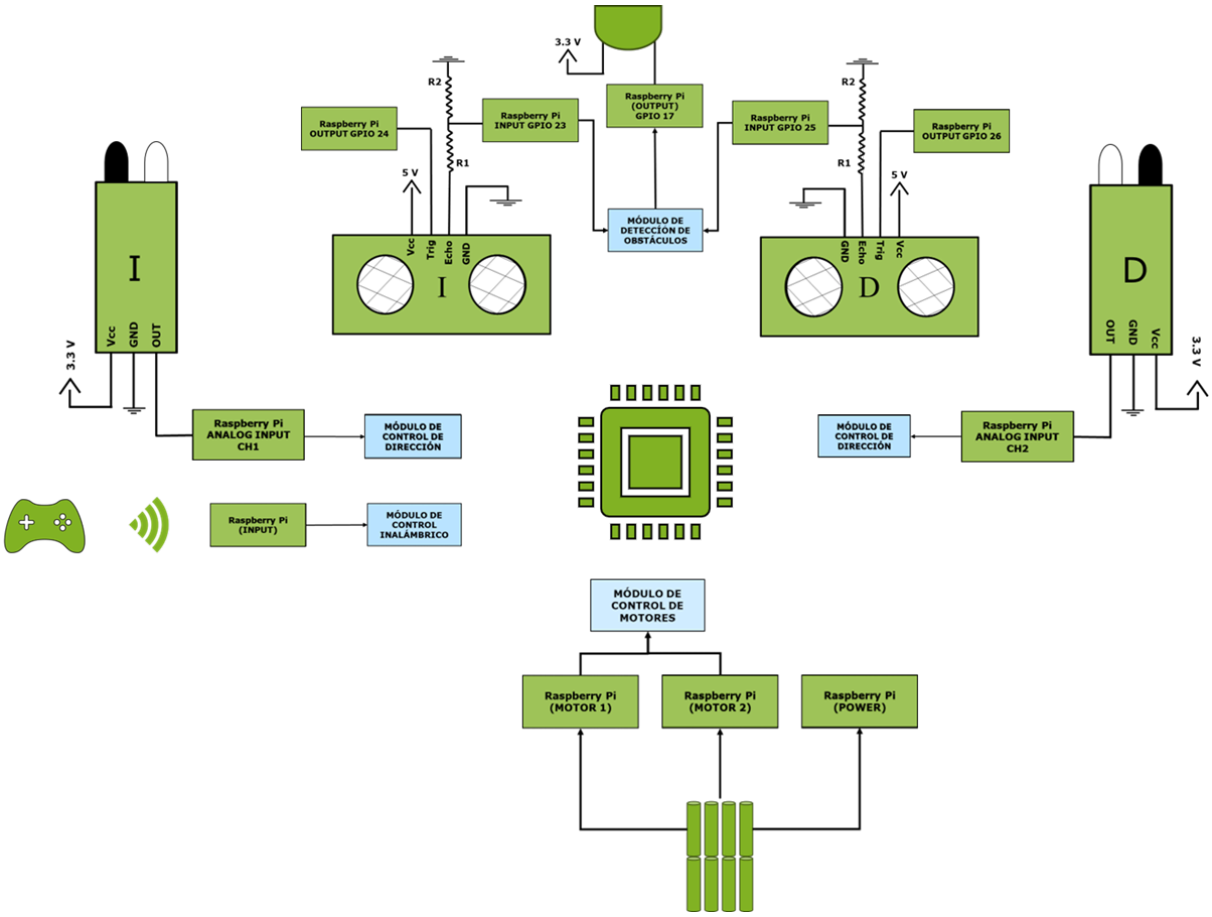

Figura 10: Arquitectura completa

6.9. Implementación mediante máquina de estados

Finalemente la implementación mediante una máquina de estados queda representada en el siguiente esquema:

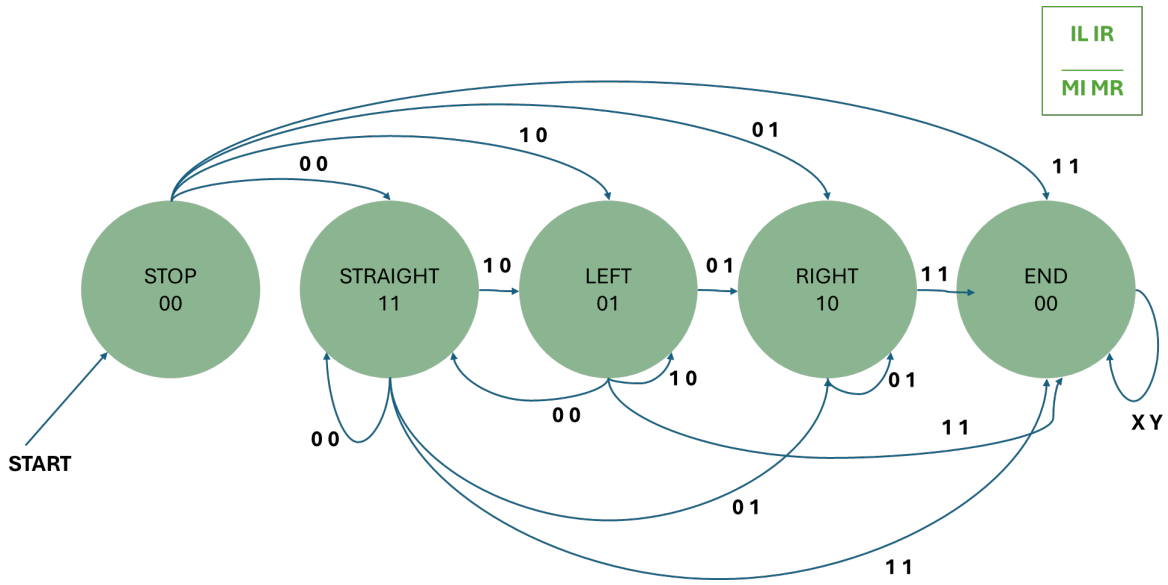

Figura 11: Diseño de máquina de estados

7. Problemas detectados durante el desarrollo

A lo largo del proyecto se han identificado varias dificultades prácticas:

1. **Conflicto entre modos y sensores.**
   Aunque el hardware permite disponer simultáneamente de sensores de ultrasonidos e infrarrojos, en la práctica ha sido necesario *conectar y desconectar* ciertos sensores en función del modo en ejecución para evitar interferencias o lecturas incorrectas. Esto complica el cambio rápido entre modos.

2. **Diferencias entre motores y desalineación mecánica.**
   Los dos motores no responden de forma idéntica a las mismas señales PWM, y pequeñas desalineaciones en la colocación de las ruedas hacen que el vehículo tienda a desviarse. Esto obliga a introducir factores de corrección específicos para cada motor.

3. **Sensibilidad a las condiciones del entorno.**
   El modo sigue líneas depende fuertemente del contraste entre la línea y el fondo, así como de la iluminación ambiental. Cambios en las condiciones de luz pueden exigir reajustar los umbrales de los sensores de infrarrojos.

4. **Gestión del mando y la conexión.**
   La detección y configuración del *Pro Controller* no siempre es inmediata, y en ocasiones es necesario repetir el proceso de emparejamiento o reiniciar ciertos servicios para conseguir una conexión estable.

8. Soluciones adoptadas y mejoras propuestas

Para mitigar los problemas anteriores se han aplicado las siguientes soluciones y se han identificado posibles mejoras:

- **Automatización del entorno y de la conexión del mando.**
  Se han creado **scripts de Bash** que automatizan la activación del entorno de Python y la conexión del mando. Esto reduce el tiempo necesario para poner en marcha cada modo y disminuye la probabilidad de errores de configuración.

- **Corrección periódica de la trayectoria.**
  En el modo sigue líneas se ha introducido una estrategia de *corrección periódica*: apagar y encender los motores con cierta frecuencia para corregir desviaciones acumuladas. Aunque esto reduce ligeramente la velocidad del vehículo, aumenta la precisión del recorrido y ayuda a compensar las diferencias entre motores y la desalineación de las ruedas.

- **Uso de factores de corrección en los motores.**
  Las clases de control de motores incorporan parámetros que permiten ajustar de forma independiente la velocidad efectiva de cada motor, de modo que se pueda compensar que uno de ellos sea más "rápido" que el otro.

- **Separación lógica de modos.**
  La existencia de ficheros `main.py`, `main_eyes.py`, `main_lines.py` y clases `Car` específicas para cada caso (`car.py`, `car_eyes.py`, `car_lines.py`) permite mantener la lógica de cada modo aislada, lo que simplifica el razonamiento y facilita la depuración.

Como **trabajo futuro** se plantea:

- Diseñar un sistema de **cambio de modo dinámico** sin necesidad de desconectar sensores físicamente.

- Integrar técnicas de **control más avanzadas** (por ejemplo, control PID sobre la trayectoria de la línea).

- Añadir **telemetría y registro de datos** para analizar el comportamiento del vehículo a lo largo del tiempo.

- Implementar algunas de las extensiones de la propuesta: LED RGB, acelerómetro, sensor de luz, pantalla de estado, etc.

9. Conclusiones

El proyecto **TERRY** ha permitido integrar de forma práctica conceptos de electrónica, programación embebida y desarrollo de aplicaciones web. Sobre una Raspberry Pi se ha desplegado un sistema que combina:

- lectura de sensores de ultrasonidos e infrarrojos,

- control de motores mediante PWM,

- recepción de eventos de un mando inalámbrico,

- y publicación de un panel web para monitorización en tiempo real.

Se han conseguido los dos objetivos funcionales principales:

1. control manual del coche mediante el *Pro Controller* con detección de obstáculos,

2. seguimiento autónomo de una línea negra en el suelo.

Aunque se han encontrado diversas limitaciones de tipo mecánico y de integración entre modos, las soluciones adoptadas (scripts de automatización, corrección periódica de trayectoria, factores de corrección en los motores) han permitido disponer de un prototipo funcional que cumple los requisitos planteados en la propuesta inicial y sirve como base para futuras ampliaciones.

En conjunto, el proyecto ha sido una experiencia completa de diseño, implementación y prueba de un sistema electrónico real, muy próximo a aplicaciones de robótica móvil y vehículos autónomos a pequeña escala.