Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de la Telecomunicación

Proyecto de Coche Autónomo con Arduino

Autor: Ignacio Sandoval Laborde
Tutor: Juan Antonio Sánchez Segura

Dpto. Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2024

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de la Telecomunicación

Proyecto de Coche Autónomo con Arduino

Autor:
Ignacio Sandoval Laborde

Tutor:
Juan Antonio Sánchez Segura
Profesor colaborador

Dpto. Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2024

Trabajo Fin de Grado: Proyecto de Coche Autónomo con Arduino

Autor: Ignacio Sandoval Laborde

Tutor: Juan Antonio Sánchez Segura

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2024

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

Quiero agradecer primeramente a mi familia y amigos que han estado conmigo día a día, apoyándome y animándome a seguir adelante, especialmente en los momentos más difíciles de mi carrera.

En segundo lugar, expreso mi agradecimiento a todos los profesores que me han enseñado lo que significa ser un ingeniero de telecomunicación, compartiendo generosamente sus conocimientos.

A mi pareja Elena, por su apoyo incondicional y constante motivación, que ha sido fundamental para alcanzar este logro.

A mi compañero Fran, por las innumerables jornadas y noches de estudio compartidas, ayudándonos mutuamente a superar los desafíos de la carrera.

A mis padres y a mi hermana, por su amor y sacrificio, que me han permitido llegar hasta aquí.

A mi amigo Lozano, por su pesadez con que acabará de una vez la carrera ya que solo me faltaba el TFG y por ofrecerme su casa y su amplio salón para realizar las pruebas experimentales del proyecto.

Finalmente, un agradecimiento especial a Juan Antonio Sánchez por su apoyo, orientación a lo largo del desarrollo de este proyecto y por los momentos de risas en clase.

Ignacio Sandoval Laborde

Sevilla, 2024

Resumen

En el Trabajo Fin de Grado (TFG), se desarrolló un coche autónomo utilizando la plataforma Arduino. Este vehículo, equipado con dos habilidades clave –seguir líneas y detectar obstáculos, se convirtió en una versión innovadora de un robot de apoyo de almacenamiento.

Como si fuera una brújula digital, el coche utiliza un algoritmo inteligente para seguir una línea negra sobre un fondo blanco. Este algoritmo analiza la información de tres sensores de línea y ajusta la dirección del coche, asegurándose de que se mantenga firme en su camino predefinido.

Un sensor de ultrasonidos, como un vigilante incansable, protege al coche de choques inesperados. Al detectar obstáculos cercanos, el sistema activa alertas sonoras que se intensifican en frecuencia a medida que el objeto se acerca, proporcionando una señal de advertencia intuitiva que garantiza una respuesta rápida y segura.

El coche autónomo no se limita a las tareas básicas de navegación. Se le dotó de funcionalidades avanzadas, como la lectura de temperatura a través del sensor MPU6050. Este añadido permite al sistema monitorear las condiciones ambientales y tomar decisiones informadas en tiempo real.

Las características del coche autónomo lo convierten en una herramienta ideal para entornos industriales que exigen un transporte seguro y eficiente. En almacenes y entornos de producción automatizados, el coche mejora la eficiencia operativa al reducir el riesgo de colisiones y optimiza la gestión logística al proporcionar un transporte confiable y automatizado de mercancías.

Este proyecto no solo representa un avance tecnológico en la automatización industrial, sino que también demuestra la viabilidad de soluciones autónomas para mejorar la seguridad y eficiencia en entornos industriales modernos. El coche autónomo con Arduino es un ejemplo tangible del potencial que tiene la tecnología para transformar la forma en que se trabaja e interactúa con el mundo que nos rodea.

Abstract

In the Final Degree Project (TFG), an autonomous car was developed using the Arduino platform. This vehicle, equipped with two key abilities - line following and obstacle detection, became an innovative version of a storage support robot.

Like a digital compass, the car uses an intelligent algorithm to follow a black line on a white background. This algorithm analyzes information from three line sensors and adjusts the direction of the car, ensuring that it stays firmly on its predefined path.

An ultrasonic sensor, like a tireless watchman, protects the car from unexpected collisions. Upon detecting nearby obstacles, the system triggers audible alerts that intensify in frequency as the object approaches, providing an intuitive warning signal that ensures a quick and safe response.

The autonomous car is not limited to basic navigation tasks. It was equipped with advanced functionalities, such as temperature reading through the MPU6050 sensor. This addition allows the system to monitor environmental conditions and make informed decisions in real time.

The autonomous car's features make it an ideal tool for industrial environments that demand safe and efficient transportation. In warehouses and automated production environments, the car improves operational efficiency by reducing the risk of collisions and optimizes logistics management by providing reliable and automated transportation of goods.

This project not only represents a technological breakthrough in industrial automation, but also demonstrates the feasibility of autonomous solutions to improve safety and efficiency in modern industrial environments. The Arduino-powered autonomous car is a tangible example of the potential of technology to transform the way we work and interact with the world around us.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

dmax	Es la distancia máxima de detección.
Pe	Es la potencia de emisión del LED infrarrojo.
R	Es la reflectividad del objeto.
k	Es una constante que incluye factores como la eficiencia del receptor y la geometría del sistema.
Ithre	Es la corriente mínima detectada por el receptor para considerar una señal válida.
Tc	Es la temperatura en grados Celsius.
TADC	Es el valor de salida del ADC.
Offset	Es el valor de compensación.
sensi	Es la sensibilidad del sensor de temperatura
Tbase	Es la temperatura base de referencia
Vcc	Voltaje de suministro positivo en un circuito electrónico
GND	Conexión de tierra o referencia de voltaje cero en un circuito.
SDA	Línea de datos serial en el protocolo I2C.
SCL	Línea de reloj serial en el protocolo I2C.
PWM	Técnica para controlar la potencia mediante la variación del ancho de los pulsos de una señal.
DC	Corriente continua
:	Tal que
<	Menor o igual
>	Mayor o igual.

1 INTRODUCCIÓN

1.1 Contexto del proyecto

En un mundo que se vuelve loco por la automatización, los robots están cada vez más presentes, ¡y no solo en las películas! Imagínate un cochecito que puede moverse solo por los almacenes, siguiendo un camino predefinido como si fuera un GPS con ruedas. En este proyecto se presenta un coche que cumple con lo anteriormente mencionado, gracias a unos sensores infrarrojos que detectan líneas negras sobre un fondo en blanco, el coche se mantiene en el camino correcto, transportando mercancías de un lado a otro sin salirse de su camino.

Además de lo anteriormente mencionado también se le ha incorporado un sensor de ultrasonido que funcionan como ojos, que evita que el coche se chocase contra un obstáculo, una persona u otro robot en el caso de unos grandes almacenes, el coche va frenando hasta pararse a cierta distancia de seguridad del obstáculo detectado, además emite una alerta sonora con una cierta frecuencia desde que detecta el objeto y empieza a reducir su velocidad, y cuanto más cerca se encuentra del objeto más rápida es esta alerta sonora para preavisar de que el coche se acerca al objeto y de tiempo de quitar el objeto. Una vez que detecta que está demasiado cerca, el coche para por completo hasta que detecta que el objeto se ha quitado que vuelve a su ruta predefinida con normalidad.

Para añadir un plus a este proyecto, este coche no solo transporta y se protege de posibles colisiones, también lleva incorporado un sensor que mide la temperatura en diferentes puntos del almacén (que en el caso práctico del proyecto realiza mediciones en las zonas definidas en el escenario por zonas donde los 3 sensores de infrarrojos detectan a la vez una zona negra). ¿Para qué? Pues para asegurarse de que las condiciones sean perfectas para los productos que se almacenan allí, también podría servir para realizar mapas de calor dentro de un almacén, por ejemplo.

Este proyecto de coche autónomo es solo un ejemplo de lo que los robots pueden hacer en la industria. La automatización nos permite trabajar de forma más eficiente, segura y productiva, liberando a los humanos de tareas repetitivas y peligrosas para que puedan enfocarse en cosas más creativas e interesantes.

1.2 Situación actual y comparación

La implementación de vehículos autónomos en la logística y el manejo de materiales es una tendencia en crecimiento en la industria moderna. Empresas de renombre como Amazon han desarrollado y desplegado robots autónomos en sus centros de distribución para mejorar la eficiencia y reducir costos operativos. Por ejemplo, los robots Kiva de Amazon se mueven de manera autónoma en los almacenes, recogiendo y entregando estantes completos de productos a los empleados, quienes entonces pueden realizar el empaquetado y envío de manera más eficiente [1].

Los robots Kiva, ahora conocidos como Amazon Robotics, utilizan una combinación de sensores y algoritmos avanzados para navegar y realizar tareas dentro de los almacenes. Equipados con cámaras, sensores de proximidad y tecnología LIDAR, estos robots pueden mapear su entorno, identificar la ubicación de los estantes y seguir rutas óptimas para recoger y entregar productos. Además, los robots se comunican entre sí y con un sistema centralizado para coordinar sus movimientos, evitar colisiones y optimizar el flujo de trabajo.

(Figura 1-1. Imagen de Kiva robot de Amazon)

En comparación, el proyecto aquí presentado utiliza sensores infrarrojos para la detección de líneas y un sensor de ultrasonidos para la evitación de obstáculos, lo cual es un enfoque más accesible y educativo. Este tipo de kit puede ser adquirido y utilizado por cualquier persona interesada en la robótica y la automatización, ofreciendo ejemplos prácticos para emprender proyectos similares.

Nuestro proyecto ha dado un paso más con la detección de objetos y la implementación de una línea negra transversal para puntos de medición de temperatura. Estas características adicionales muestran cómo se pueden expandir las funcionalidades básicas de un kit de robot para abordar necesidades específicas, como la seguridad en la navegación y el monitoreo ambiental dentro de un almacén.

Al presentar estos avances en un contexto de aplicaciones actuales, demostramos cómo incluso proyectos más pequeños pueden contribuir al panorama de la automatización industrial, destacando la accesibilidad y el potencial de aprendizaje que ofrecen estos kits en comparación con los sistemas más avanzados y costosos utilizados por grandes empresas. Aunque no alcanzamos la complejidad y capacidad de los robots Kiva, nuestro proyecto ofrece una valiosa experiencia práctica y puede ser un punto de partida para desarrollos más avanzados en el futuro.

1.3 Objetivos del trabajo

El objetivo principal de este trabajo es diseñar, implementar y evaluar un coche autónomo que utilice un conjunto de sensores para navegar de manera autónoma y segura en un entorno controlado. Los objetivos específicos incluyen:

- **Desarrollo del algoritmo de seguimiento de línea:** Implementar un sistema que permita al coche seguir una línea negra pintada en el suelo mediante sensores infrarrojos. Esto incluye el diseño del algoritmo de control que ajuste la dirección del coche basándose en las lecturas de los sensores de seguimiento.
- **Detección y gestión de obstáculos:** Integrar un sensor de ultrasonido para detectar obstáculos en el camino del coche y desarrollar estrategias de detención y alerta mediante el uso de un buzzer. El sistema debe ser capaz de ajustar su velocidad y detenerse por completo cuando se detectan obstáculos a diferentes distancias.

- **Medición de Temperatura Ambiental:** Incorporar un sensor MPU6050 para medir la temperatura en puntos específicos del recorrido. Esto incluye la lectura de la temperatura, su validación y la implementación de medidas de seguridad en caso de detección de lecturas anómalas.
- **Validación del Sistema:** Realizar pruebas experimentales para validar el desempeño del coche autónomo en diferentes escenarios. Estas pruebas deben evaluar la precisión del seguimiento de línea, la eficacia en la detección de obstáculos y la fiabilidad de las mediciones de temperatura, asegurando que el sistema cumple con los requisitos establecidos.

1.4 Estructura del documento

Este documento está organizado en varias secciones, cada una abordando diferentes aspectos del desarrollo y funcionamiento del coche autónomo:

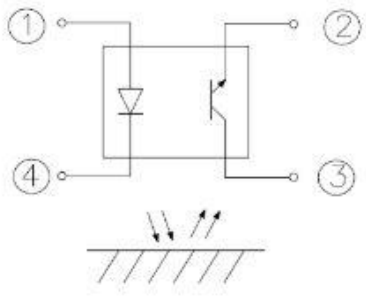
- **Fundamentos teóricos:** Se presentan los principios de funcionamiento de los sensores utilizados, como los sensores de seguimiento de línea, el sensor ultrasonido y el sensor de temperatura. También se explicará el control de motores y la integración de estos componentes en el sistema de Arduino.
- **Diseño del sistema:** Se describe la arquitectura general del coche autónomo, la selección de componentes, los diagramas de bloques y las conexiones eléctricas.
- **Implementación práctica:** Se detallan los pasos seguidos para el desarrollo del algoritmo de seguimiento de línea, la integración de la detección de obstáculos y la incorporación de la medición de temperatura.
- **Resultados experimentales:** Se presentan los resultados de las pruebas realizadas, evaluando la precisión y fiabilidad del sistema en diferentes escenarios.
- **Discusión y conclusiones:** Se analizan los resultados obtenidos, identificando las fortalezas y debilidades del sistema. Se discuten las limitaciones del proyecto, como la posible interferencia de señales o limitaciones de los sensores utilizados.
- **Anexos:** Se incluye el código fuente completo del proyecto, así como esquemas eléctricos y diagramas de conexión.

2 FUNDAMENTOS TEÓRICOS

2.1. Seguimiento de línea con sensores infrarrojos

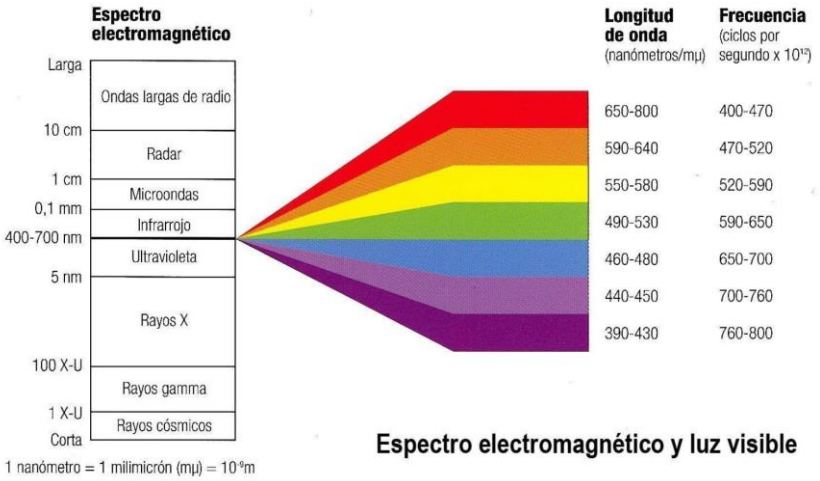
2.1.1. Principios de funcionamiento

Este tipo de robot automático utiliza sensores infrarrojos en el mío en específicamente he usado los que venían en un kit de Arduino de fabricante chino que usan 3 sensores "KTIR0711S" conectados en paralelo, los cuales se encargan de detectar la cantidad de luz reflejada por una superficie.



(Figura 2-1. Sensor infrarrojo KTIR0711S)

El sensor está conformado por un LED emisor infrarrojo, el cual genera luz en el espectro infrarrojo, este emisor hace rebotar la señal sobre la superficie que se encuentra enviando la señal mientras que un optotransistor es el encargado de recibir la luz reflejada, aquí haremos dos diferencias en las que me he basado para hacer el algoritmo de seguimiento de línea y es que si la luz choca contra una superficie blanca se reflejará y llegará al fototransistor. Si por el contrario golpea en una superficie negra, el material absorberá la mayoría de la luz y no llegará al fotorreceptor. Pero el concepto de la emisión y recepción de señales infrarrojas no es, ni mucho menos trivial, y exige tener nociones de ondas electromagnéticas (energía, frecuencia, longitud, etc.…) que sobrepasan el nivel de mi proyecto. No obstante, para aquellos que quieran profundizar más en el tema os recomiendo la excelente descripción del [2].



(Figura 2-2. Longitud de onda infrarroja)

Uno de los problemas que puede causar este tipo de sensor es conocer cuál es la distancia máxima donde este sensor puede funcionar, para ello de su datasheet he sacado la siguiente formula:

$$dmax = \sqrt{\frac{p_e \cdot R}{k \cdot I_{THre}}}$$

Donde:

- **dmax** es la distancia máxima de detección.
- **Pe** es la potencia de emisión del LED infrarrojo.
- **R** es la reflectividad del objeto.
- **k** es una constante que incluye factores como la eficiencia del receptor y la geometría del sistema.
- **Ithre** es la corriente mínima detectada por el receptor para considerar una señal válida.
- **V** es el voltaje a través del LED.
- **I** es la corriente a través del LED.

Para el cálculo de la distancia máxima, usamos los datos del datasheet (https://pdf1.alldatasheet.com/datasheet-pdf/download/196508/KINGBRIGHT/KTIR0711S.html) y teniendo en cuenta que el cálculo es diferente para detectar un color negro (R ≅0.1) o blanco (R ≅0.8):

Detección de color negro:

$$Pe=V\cdot I=1.5V\cdot 20mA=1.5V\cdot 0.02A=0.03W=30mW$$

$$dmax = \sqrt{\frac{30mW \cdot 0.1}{1 \cdot 10uA}}$$

dmax ≅ 17mm

Detección de color blanco:

$$Pe=V\cdot I=1.5V\cdot 20mA=1.5V\cdot 0.02A=0.03W=30mW$$

$$dmax = \sqrt{\frac{30mW \cdot 0.8}{1 \cdot 10uA}}$$

dmax ≅ 49mm

Gracias a estos cálculos, sabemos que los sensores de infrarrojos para que funcionen correctamente debemos tenerlos a una distancia inferior a 17mm respecto a la superficie donde realicemos las pruebas.

En este proyecto, el coche autónomo está equipado con tres sensores infrarrojos dispuestos en la parte frontal. Esta configuración permite una detección precisa de la posición de la línea mediante la combinación de las lecturas de los tres sensores, que pueden generar una variedad de patrones binarios, como 000, 001, 010, 111, etc. Cada combinación representa una posición específica del

coche respecto a la línea, proporcionando información crítica para el control de la dirección y velocidad.

2.1.2. Implementación en Arduino

La implementación en Arduino del seguimiento de línea con sensores infrarrojos se basa en la integración de los sensores y la programación del microcontrolador para interpretar las lecturas y ajustar el movimiento del coche. Cada sensor infrarrojo se conecta a un pin analógico del Arduino, y las lecturas se combinan para formar un valor binario '0' o '1' que representa el estado de los tres sensores.

Dentro del código, los 3 sensores se conectan a los pines analógicos de Arduino y se inicializan de esta forma:

```
#define PIN_TRACKING_LEFT A1
#define PIN_TRACKING_CENTER A2
#define PIN_TRACKING_RIGHT  A3

void setup() {
      pinMode(PIN_TRACKING_LEFT, INPUT);
      pinMode(PIN_TRACKING_RIGHT, INPUT);
      pinMode(PIN_TRACKING_CENTER, INPUT);
}
```

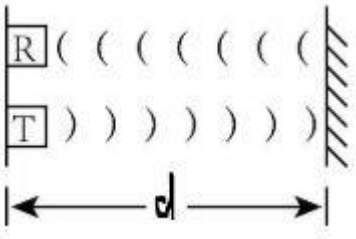
A continuación, mostraré cuál es el bucle principal que lee continuamente los valores de los sensores y ajusta la dirección del coche en consecuencia. Aquí se muestra un fragmento del código utilizado:

```
u8 getTrackingSensorVal() {
  u8 trackingSensorVal = 0;
  trackingSensorVal = (digitalRead(PIN_TRACKING_LEFT) == 1 ? 1 : 0) << 2 |
(digitalRead(PIN_TRACKING_CENTER) == 1 ? 1 : 0) << 1 |
(digitalRead(PIN_TRACKING_RIGHT) == 1 ? 1 : 0) << 0;
  return trackingSensorVal;
}
```

2.2. Detección de obstáculos con sensor ultrasonido

2.2.1. Principios de funcionamiento

El siguiente sensor primordial en mi proyecto ha sido el sensor ultrasónico HC-SR04 que es un dispositivo ampliamente utilizado en proyectos electrónicos, basados en Arduino ya que sirve para medir distancias de manera precisa y no intrusiva. El sensor HC-SR04 es un sensor de distancia de bajo costo, su uso es muy frecuente en la robótica, utiliza transductores de ultrasonido para detectar objetos.



(Figura 2-3. Funcionamiento de onda ultrasonido)

Este sensor tiene 4 patillas: una para la alimentación (VCC), otra para la tierra (GND), y otras dos que son Trig y Echo, que sirven para enviar la señal que activará el pulso y para recibir la señal convertida en impulso eléctrico respectivamente, lo cual nos ayudará a calcular la distancia a la que está el obstáculo.

(Figura 2-4. Sensor ultrasonido HC-SR04)

Su funcionamiento consiste en emitir un sonido ultrasónico por uno de sus transductores, y esperar que el sonido rebote de algún objeto presente, el eco es captador por el segundo transductor. comentado Cuando el módulo recibe las ondas ultrasónicas devueltas al encontrar un obstáculo, el pin Echo se tira hacia abajo (es decir un 0 lógico).

La duración del nivel alto en el pin Eco es el tiempo total de la onda ultrasónica desde la transmisión hasta la recepción La distancia es proporcional al tiempo que demora en llegar el eco. Esto se realiza constantemente.

Partimos de la siguiente formula:

$$v = \frac{d}{t}$$

Donde:

- **d** es dos veces la distancia hacia el objeto.
- **v** es la velocidad del sonido.
- **t** es el tiempo que demora en llegar el ultrasonido al objeto y regresar al sensor.

Donde la velocidad del sonido es 340metros/segundo, pero usaremos las unidades en cm/us pues trabajaremos en centímetros y microsegundos reemplazando en la formula tenemos:

$$\frac{340m}{s} * \frac{1s}{1000000us} x \frac{100cm}{1m} = \frac{2d}{t}$$

$$d(cm) = \frac{t(uS)}{59}$$

Según el datasheet [3] del sensor, la distancia mínima y máxima va desde 2-400 cm.

2.2.2. Implementación en Arduino

La implementación en Arduino de la detección de obstáculos con sensor de ultrasonido se basa en la integración del sensor y la programación del microcontrolador para interpretar las lecturas y ajustar el movimiento del coche. El sensor de ultrasonido se conecta a dos pines digitales del Arduino, se conecta el trigger del sensor al pin 7 y el echo del sensor al pin 8 como muestro a continuación, así quedaría la implementación en código Arduino:

```
#define PIN_SONIC_TRIG    7
#define PIN_SONIC_ECHO    8

void setup() {
      pinMode(PIN_SONIC_TRIG, OUTPUT);
      pinMode(PIN_SONIC_ECHO, INPUT);
}
```

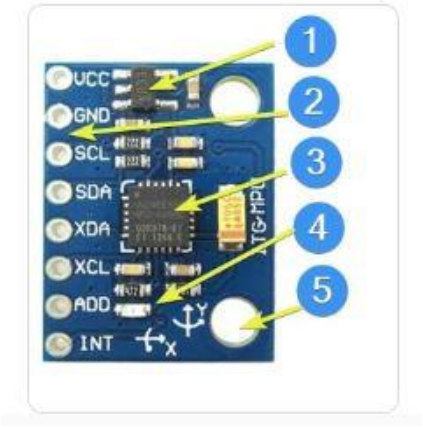
A continuación, mostraré cuál es el bucle principal que lee continuamente los valores del sensor y ajusta el movimiento del coche en consecuencia. Aquí se muestra un fragmento del código utilizado:

```
float getSonar() {
    unsigned long pingTime;
    float distance;
    digitalWrite(PIN_SONIC_TRIG, HIGH);
    delayMicroseconds(10);
    digitalWrite(PIN_SONIC_TRIG, LOW);
    pingTime = pulseIn(PIN_SONIC_ECHO, HIGH, SONIC_TIMEOUT);
    if (pingTime != 0)
        distance = (float)pingTime * SOUND_VELOCITY / 2 / 10000;
    else
        distance = MAX_DISTANCE;
    return distance;
}
```

2.3 Medición de temperatura

2.3.1. Principios de funcionamiento

El MPU6050 es un sensor de seis ejes que integra un acelerómetro y un giroscopio de tres ejes cada uno, además de un sensor de temperatura interno. Aunque es más conocido por sus capacidades de medición de movimiento, también incluye un sensor de temperatura que puede ser utilizado para medir la temperatura ambiente [4].



(Figura 2-5. Imagen de MPU6050)

1. Regulador de voltaje - Convierte la entrada de 5 V del Arduino UNO en 3,3 V. Los 3,3 V son la tensión de alimentación del circuito integrado MPU6050.
2. Opción de conector Bergstick - Puede acceder a las líneas I2C, Interrupción, configuración ADDress y alimentación en estas líneas.
3. MPU6050 IC - MPU6050 es un IC de 24 pines. Puede localizar la patilla 1 del circuito integrado con la ayuda de un pequeño punto. La imagen de arriba indica el punto en la parte superior izquierda del IC.
4. LED de indicación de alimentación
5. Orificios de montaje

El módulo MPU6050 tiene ocho pines. La siguiente tabla describe los ocho pines presentes en el módulo MPU.

	Pin Label	Pin Description
1	VCC	Entrada de 5 V. Puede conectar 5 V en el Arduino UNO a la VCC
2	GND	Tierra
3	SCL	I2C Línea de reloj serie
4	SDA	I2C Línea de datos serie
5	XDA	Línea de datos I2C Master (para conectar sensores externos)
6	XCL	Línea I2C Master Clock (para conectar sensores externos)
7	AD0	Dirección I2C Esclavo LSB. Puedes conectarlo a Low o High. Por lo tanto, puede tener dos direcciones I2C para MPU6050 ICs en su placa
8	INT	Interrumpir entrada digital

(Tabla 2–1. Pines del MPU6050)

Se puede acceder a los registros y a la memoria del MPU6050 mediante I2C. La velocidad máxima de bus I2C soportada es de 400 kHz.

La dirección esclava del MPU6050 es 0b110100x, donde x puede ser cero o uno. Por lo tanto, las dos direcciones posibles son 0b1101000 y 0b1101001.El valor de x es el nivel lógico presente en el pin de entrada AD0, como se muestra en la siguiente tabla.

AD0 Pin Logic Level	7 bit I2C address of the MPU6050
Low	1101000
High	1101001

(Tabla 2–2. Tabla de selección de dirección I2C para el MPU6050 mediante el pin AD0)

La medición de la temperatura se realiza a través de un ADC (Convertidor Analógico a Digital) interno que proporciona una salida digital correspondiente a la temperatura del chip.

La salida del ADC interno para la temperatura se puede convertir a grados Celsius utilizando la siguiente ecuación:

$$Tc = \left(\frac{T_{ADC} - Offset}{sensi}\right) + Tbase$$

donde:

- **Tc** es la temperatura en grados Celsius.
- **TADC** es el valor de salida del ADC.
- **Offset** es el valor de compensación.
- **Sensi** es la sensibilidad del sensor de temperatura (340 LSB/°C para el MPU6050).
- **Tbase** es la temperatura base de referencia (35°C para el MPU6050).

Para el MPU6050, esta ecuación se especifica como:

$$Tc = \left(\frac{T_{ADC} - 521}{340}\right) + 35$$

En esta ecuación, 521 es el valor de Offset en LSB y 340 LSB/°C es la sensibilidad del sensor.

Dado que estamos utilizando el MPU6050 únicamente para medir la temperatura, no se espera un sobrecalentamiento significativo del chip. Por lo tanto, la temperatura ambiente y la temperatura del chip serán prácticamente iguales. Esto significa que las lecturas del sensor de temperatura del MPU6050 proporcionarán una buena estimación de la temperatura ambiental en situaciones normales de operación.

Por ejemplo, si el valor de salida del TADC es 512 (un valor correspondiente al uso que se le da al MPU), la temperatura medida sería:

$$T_c = \left(\frac{512 - 521}{340}\right) + 35 \approx 34.97$$

La temperatura se puede leer a través del registro interno del sensor utilizando el protocolo I2C, que es el método de comunicación principal del MPU6050 con los microcontroladores, como el Arduino.

Características del sensor de temperatura:

- **Rango de Medición:** La salida de temperatura está calibrada en el rango de -40°C a +85°C.

- **Resolución:** La resolución de la medición de temperatura es de 16 bits.
- **Precisión:** La precisión típica es de ±1°C.

El registro de temperatura proporciona una lectura en formato bruto que debe ser convertida a grados Celsius.

2.3.2. Implementación en Arduino

La implementación en Arduino de la medición de temperatura con el MPU6050 se basa en la integración del sensor y la programación del microcontrolador para interpretar las lecturas, y comprobar que estas son correctas. Para conectar el MPU6050 a mi placa Arduino he utilizado la interfaz I2C con cables hembra a hembra, siguiendo el siguiente esquema:

Pin MPU6050	Pin Arduino
VCC	5V
GND	GND
SDA	A4
SCL	A5

(Tabla 2–3. Tabla de conexiones de pines MPU6050-Arduino)

A continuación, muestro como quedaría la implementación en código Arduino:

```
#include <Wire.h>
#include <MPU6050.h>

MPU6050 mpu;

void setup() {
      Wire.begin();
      mpu.initialize();
      if (!mpu.testConnection()) {
            Serial.println("MPU6050 no conectado correctamente");
            while (1);
      }
}
```

• **#include <Wire.h>**: Incluye la biblioteca estándar de Arduino para la comunicación I2C.

• **#include <MPU6050.h>**: Incluye la biblioteca para el MPU6050, que facilita la comunicación y la configuración del sensor. Esta biblioteca debe estar instalada previamente en tu entorno de desarrollo Arduino.

• **MPU6050 mpu;**: Declara un objeto mpu de tipo MPU6050. Esto es esencial para poder llamar a funciones específicas de la biblioteca MPU6050.h para configurar y comunicarse con el sensor.

• **Wire.begin(); :** Inicia la comunicación I2C. Es esencial llamar a Wire.begin() en setup() para inicializar la interfaz I2C y establecer la comunicación con el MPU6050.

• **mpu.initialize();**: Inicializa el MPU6050. Esta función prepara el sensor para su uso configurando parámetros internos y realizando cualquier inicialización necesaria.

• **if (!mpu.testConnection()) { ... }**: Verifica si se estableció correctamente la comunicación con el MPU6050. La función testConnection() devuelve true si se detecta correctamente el MPU6050 a través del bus I2C. Si no se detecta correctamente, se imprime un mensaje de error en el monitor serial y el Arduino entra en un bucle infinito (while (1) ;) para detener cualquier otra acción.

2.4 Alimentación del sistema

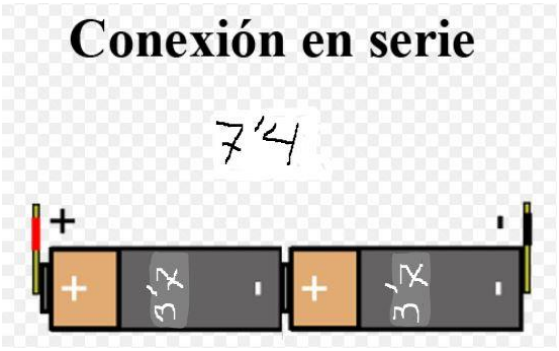
2.4.1 Principios de funcionamiento

En este proyecto un módulo de batería portátil es un elemento esencial, ya que sirve para proporcionar energía de manera autónoma a un coche que se encuentra en continuo movimiento, evitando así el uso de cables a una corriente estática. Este módulo se alimenta mediante dos pilas 18650 en serie para obtener el voltaje necesario para hacer funcionar a todos los elementos del coche autónomo.

Las pilas 18650 son células de ion litio recargables ampliamente utilizadas debido a su alta capacidad de energía y voltaje nominal de 3.7V por celda.

(Figura 2-6. Imagen de pilas 18650)

Si conectamos dos pilas 18650 en serie obtenemos un voltaje total de aproximadamente 7.4V (considerando 3.7V por celda), que es adecuado para muchos dispositivos electrónicos y microcontroladores como Arduino.



(Figura 2-7. Dibujo circuito pilas en serie)

El módulo de batería portátil incluye:

- **Configuración en Serie**: Las dos pilas 18650 se conectan en serie para suministrar un voltaje combinado mayor. Esto proporciona una mayor capacidad de almacenamiento de energía y una tensión suficiente para alimentar los componentes del sistema.
- **Protección de Batería**: Es crucial incorporar circuitos de protección para las pilas 18650, que supervisan el voltaje y la corriente para prevenir sobrecargas, sobredescargas y cortocircuitos. Esto asegura la seguridad y prolonga la vida útil de las baterías.
- **Regulación de Voltaje**: Dependiendo de los requisitos del sistema, es posible que se necesite un regulador de voltaje para estabilizar la salida de las pilas a un voltaje constante, especialmente si se requiere alimentar componentes sensibles que operan a un voltaje específico.

2.4.2. Implementación en Arduino

La implementación en Arduino del módulo de batería portátil se basa en la integración del módulo de batería y la programación del microcontrolador para interpretar las lecturas de voltaje y asegurar que el sistema se alimenta correctamente. A continuación, muestro como quedaría la implementación en código Arduino:

```
#define PIN_BATTERY     A0
float batteryVoltage = 0;
void setup() {
     pinMode(PIN_BATTERY, INPUT);
}
A continuación, enseño una función para asegurar que el sistema se alimenta
correctamente:

bool getBatteryVoltage() {
  if (!isBuzzered) {
    pinMode(PIN_BATTERY, INPUT);
    int batteryADC = analogRead(PIN_BATTERY);
    if (batteryADC < 614)
    {
      batteryVoltage = batteryADC / 1023.0 * 5.0 * 4;
      return true;
    }
  }
  return false; }
```

Esta función tiene como objetivo leer el voltaje de una batería conectada a un pin analógico de un Arduino (A0 en nuestro caso) y devolver un valor booleano (true o false) basado en si el voltaje leído está dentro de un rango específico.

1. **Condición Inicial**: La función primero verifica si la variable global ***isBuzzered*** es false. Si ***isBuzzered*** es true, la función termina inmediatamente y devuelve false. Esto implica que hay una condición externa que controla si el voltaje de la batería debe ser leído o no.
2. **Configuración del Pin de Batería**: Si la condición inicial se cumple, el pin designado para leer el voltaje de la batería (***PIN_BATTERY***) se configura como una entrada. Esto permite al Arduino leer el valor analógico presente en ese pin.
3. **Lectura del Valor Analógico**: El Arduino lee el valor analógico del pin de la batería y lo almacena en una variable llamada ***batteryADC***. Este valor será un número entre 0 y 1023, que representa la tensión de la batería.

4. **Verificación del Valor Leído**: La función comprueba si el valor leído (***batteryADC***) es menor que 614. Este valor umbral está relacionado con una proporción del voltaje de referencia del Arduino (5V). En este contexto, un valor de 614 corresponde aproximadamente a 3V cuando se considera un divisor de voltaje que reduce la tensión de la batería.
5. **Cálculo del Voltaje de la Batería**: Si el valor leído es menor que 614, la función calcula el voltaje real de la batería usando la fórmula: ***batteryVoltage = batteryADC / 1023.0 * 5.0 * 4***. Este cálculo convierte el valor analógico leído en un voltaje real teniendo en cuenta que el valor analógico es una fracción del voltaje de referencia (5V) y ajustando por un factor de escala (* 4) correspondiente al divisor de voltaje utilizado.
6. **Resultado de la Función**:
    - Si el valor leído es menor que 614 y el cálculo del voltaje se realiza, la función devuelve true, indicando que el voltaje de la batería está dentro de un rango aceptable.
    - Si **isBuzzered** es true o el valor leído no cumple con la condición (***batteryADC*** no es menor que 614), la función devuelve false.

2.5. Sistema de alertas

2.5.1. Principios de funcionamiento

Un buzzer es un componente de audio. Existen buzzer activos y pasivos. Los buzzer activos tienen oscilador en su interior, estos sonarán siempre que reciban alimentación. Los buzzer pasivos requieren un oscilador externo (generalmente PWM con diferentes frecuencias) para emitir un sonido [5].

(Figura 2-8. Dibujo buzzer activo y pasivo)

Los *buzzer* activos son más fáciles de usar. Generalmente, sólo pueden emitir una frecuencia de sonido específica. En cambio, los buzzer pasivos requieren un circuito externo para emitir sonidos, pero pueden controlarse para que emitan sonidos de varias frecuencias. La frecuencia de resonancia del buzzer pasivo es de 2 kHz, lo que significa que el buzzer pasivo es más fuerte cuando la frecuencia de resonancia es de 2kHz.

En este coche, el buzzer es un buzzer activo. Así que sólo puede emitir una frecuencia específica de sonido y genera sonidos cuando se aplica un voltaje adecuado a sus terminales. La función tone() de Arduino no se puede utilizar para este tipo de buzzer.

2.5.2. Implementación en Arduino

La implementación en Arduino del buzzer activo se basa en la integración del componente de buzzer y la programación del microcontrolador para controlar su activación. A continuación, se muestra cómo se puede realizar la implementación en código Arduino:

```
#define PIN_BUZZER      A0

bool isBuzzered = false;
void setup() {
     setBuzzer(false);
}
```

```
void setBuzzer(bool flag) {
  isBuzzered = flag;
  pinMode(PIN_BUZZER, flag);
  digitalWrite(PIN_BUZZER, flag);
}
```

A continuación, enseño una función para realizar que el sistema de alertas:

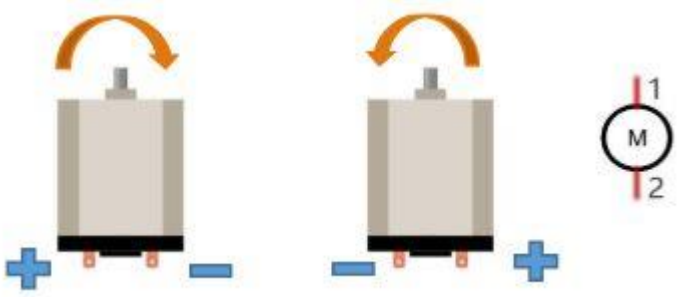
```
void alarm(u8 beat, u8 repeat) {
  beat = constrain(beat, 1, 9);
  repeat = constrain(repeat, 1, 255);
  for (int j = 0; j < repeat; j++) {
    for (int i = 0; i < beat; i++) {
      setBuzzer(true);
      delay(100);
      setBuzzer(false);
      delay(100);
    }
    delay(500);
  }
}
```

2.6. Sistema de movimiento

2.6.1. Principios de funcionamiento

El sistema de movimiento implementado en este proyecto se basa en el control de cuatro motores utilizando modulación por ancho de pulso (PWM) mediante un microcontrolador Arduino. Cuando el motor está conectado a la fuente de alimentación, girará en una dirección. Si se invierte la polaridad

el motor girará en sentido contrario. La velocidad del motor depende de la tensión entre los dos extremos. Cuanto mayor sea la tensión mayor será la velocidad.

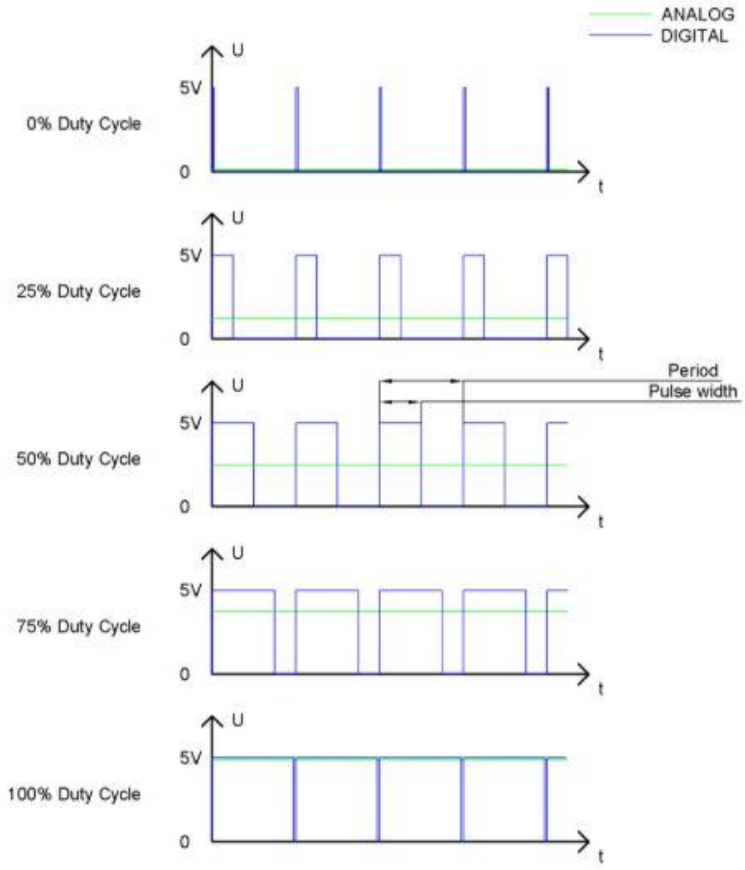


(Figura 2-9. Dibujo de motores)

Además, los motores de este proyecto están configurados para recibir señales PWM desde Arduino, la modulación por ancho de pulso (PWM) es una técnica eficaz para controlar motores eléctricos, ya que proporciona un método para variar la potencia suministrada a los motores ajustando el ciclo de trabajo de la señal [6].

Las señales enviadas a los motores a través de los pines digitales son ondas cuadradas con frecuencias específicas. Estas ondas consisten en ciclos repetitivos de niveles altos y bajos, alternando en intervalos regulares. El tiempo total de cada ciclo se conoce como periodo, siendo la inversa del periodo la frecuencia de la señal. La duración del nivel alto de la señal se denomina anchura de pulso, y el ciclo de trabajo representa el porcentaje de tiempo que el nivel alto está activo en relación con el periodo total de la forma de onda (T).

Cuanto más tiempo dure la salida de niveles altos, mayor será el ciclo de trabajo y mayor será la tensión correspondiente en la señal analógica. Las siguientes figuras muestran cómo varía la tensión de la señal analógica entre 0V-5V (el nivel alto es 5V) correspondiente al ancho de pulso 0%-100%:



(Figura 2-10. Variación de señal analógica entre 0V-5V)

Cuanto mayor sea el ciclo de trabajo PWM, mayor será la potencia de salida. Ahora que entendemos esta relación, podemos utilizar PWM para controlar el brillo de un LED o la velocidad de un motor de DC, etc.

2.6.2. Implementación en Arduino

La implementación en Arduino de movimiento del coche se basa en la integración de los 4 motores y la programación del microcontrolador para controlar su movimiento. A continuación, se muestra cómo se puede realizar la implementación en código Arduino:

```
#define MOTOR_DIRECTION     1
#define PIN_DIRECTION_LEFT  4
#define PIN_DIRECTION_RIGHT 3
#define PIN_MOTOR_PWM_LEFT  6
#define PIN_MOTOR_PWM_RIGHT 5

void setup() {
      pinMode(PIN_DIRECTION_LEFT, OUTPUT);
      pinMode(PIN_MOTOR_PWM_LEFT, OUTPUT);
      pinMode(PIN_DIRECTION_RIGHT, OUTPUT);
      pinMode(PIN_MOTOR_PWM_RIGHT, OUTPUT);
```

```
}
```

A continuación, enseño una función para realizar que el sistema de movimiento del coche:

```
void motorRun(int speedl, int speedr) {
  int dirL = 0, dirR = 0;
  if (speedl > 0) {
    dirL = 0 ^ MOTOR_DIRECTION;
  } else {
    dirL = 1 ^ MOTOR_DIRECTION;
    speedl = -speedl;
  }

  if (speedr > 0) {
    dirR = 1 ^ MOTOR_DIRECTION;
  } else {
    dirR = 0 ^ MOTOR_DIRECTION;
    speedr = -speedr;
  }
  speedl = constrain(speedl, 0, 255);
  speedr = constrain(speedr, 0, 255);
  digitalWrite(PIN_DIRECTION_LEFT, dirL);
  digitalWrite(PIN_DIRECTION_RIGHT, dirR);
  analogWrite(PIN_MOTOR_PWM_LEFT, speedl);
  analogWrite(PIN_MOTOR_PWM_RIGHT, speedr);
}
```

2.7. Módulo Bluetooth

2.7.1. Principios de funcionamiento

El módulo Bluetooth se integra al Arduino utilizando el puerto serie para la comunicación. Es importante destacar que el mismo puerto serie se emplea tanto para cargar programas en Arduino como para la comunicación con el módulo Bluetooth. Por lo tanto, durante el proceso de carga del programa en Arduino, el módulo Bluetooth debe estar desconectado para evitar posibles fallos en la carga del programa.

Cuando el módulo Bluetooth no está conectado a otro dispositivo, se puede configurar mediante comandos AT. Una vez establecida la conexión, el módulo Bluetooth opera como un canal de datos y ya no puede reconfigurarse. Por defecto, el módulo Bluetooth utiliza una velocidad de transmisión de 9600 baudios [7], sin paridad, 8 bits de datos y 1 bit de parada. El nombre predeterminado del Bluetooth es "BT05" y su modo de rol es esclavo.

Para evitar confusiones al conectar múltiples módulos Bluetooth con el mismo nombre alrededor, es recomendable personalizar el nombre del módulo mediante el comando AT durante la configuración inicial.

2.7.2. Implementación en Arduino

La implementación en Arduino de la comunicación bluetooth se basa en la integración de los 4 motores y la

programación del microcontrolador para controlar su movimiento. A continuación, se muestra cómo se puede realizar la implementación en código Arduino:

```
#include <SoftwareSerial.h>
SoftwareSerial BTSerial(0, 1);

void setup() {
      Serial.begin(9600);
      BTSerial.begin(9600);
}
```

Donde:

- Se utiliza la biblioteca **SoftwareSerial** para establecer una comunicación serial con el módulo Bluetooth a través de pines digitales 0 y 1 de Arduino.
- **Serial.begin(9600)** se utiliza para iniciar la comunicación serial con la computadora o terminal serial, lo que permite la depuración y el monitoreo del programa.
- **BTSerial.begin(9600)** se utiliza para iniciar la comunicación serial con el módulo Bluetooth, configurando la velocidad de transmisión (baud rate) en 9600 baudios, que debe coincidir con la configuración del módulo Bluetooth.

Esta configuración básica permite al Arduino establecer comunicación bidireccional con otros dispositivos Bluetooth compatibles, facilitando la transmisión de datos y comandos para controlar los motores u otras funciones del proyecto.

3 DISEÑO DEL SISTEMA

El éxito es la suma de pequeños esfuerzos, repetidos día tras día.

- Robert Collie -

En esta sección se describe el diseño general del sistema del coche autónomo, detallando la arquitectura general, la selección y justificación de los componentes, y los diagramas de bloques y conexiones. El coche autónomo está basado en el kit "4WD Car Kit FNK0041" de Amazon [8], que incluye un Arduino Uno, una placa auxiliar, motores, sensores de ultrasonido e infrarrojos, y el chasis del coche.

3.1 Arquitectura general de coche autónomo

La arquitectura del coche autónomo se basa en una estructura modular que facilita la integración y comunicación entre los diferentes componentes del sistema. El núcleo del sistema es el microcontrolador Arduino Uno, que gestiona la entrada de datos de los sensores y controla los actuadores.

- **Microcontrolador**: Arduino Uno
- **Placa auxiliar**: Placa adicional con elementos pre-soldados para facilitar las conexiones.
- **Sensores de entrada**:
    - Sensores infrarrojos para el seguimiento de línea.
    - Sensor de ultrasonido para la detección de obstáculos.
    - MPU650 para medir la temperatura
    - Módulo bluetooth para enviar datos.
- **Actuadores**:
    - Cuatro motores DC controlados mediante PWM para el movimiento del coche.
    - Buzzer activo para el sistema de alertas por proximidad y por medición de temperatura.
- **Sistema de alimentación**:
    - Módulo de batería portátil utilizando dos pilas 18650 en serie para alimentar el sistema.
- **Chasis del coche**:
    - Proporciona la estructura física para montar todos los componentes.

El flujo de datos comienza con los sensores de entrada que recogen información del entorno. Estos datos se envían al Arduino Uno, que los procesa y toma decisiones basadas en la lógica programada. Las señales de control se envían a los motores a través de la placa auxiliar para lograr el movimiento deseado. El módulo de batería portátil asegura que el sistema esté alimentado adecuadamente.

3.1.1 Montaje del coche autónomo

3.1.1.1 Instalación de los soportes del motor y ruedas

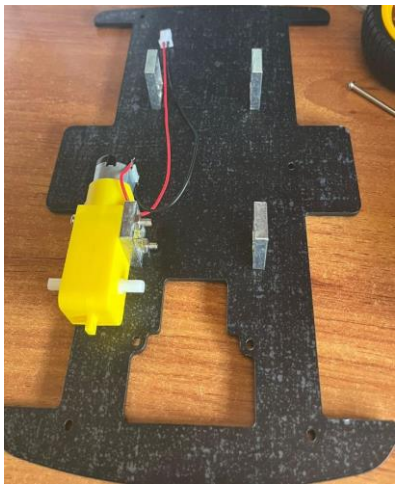
1. **Preparación del chasis**:
    - Primero, identifiqué la placa acrílica inferior de mi kit, la cual servirá como base para todos los componentes.

(Figura 3-1. Imagen del chasis)

2. **Fijación del soporte del motor**:

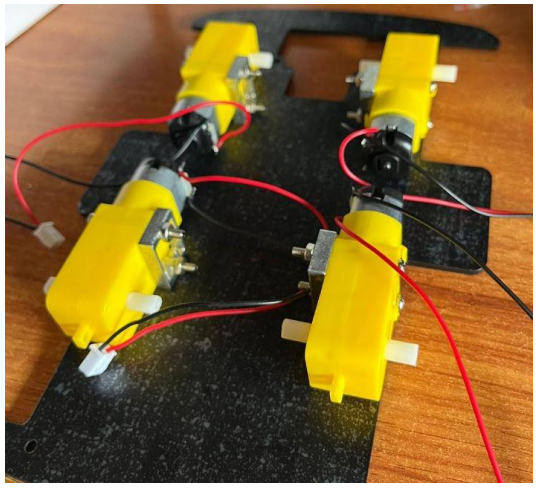
- Coloqué el soporte del motor sobre la placa acrílica inferior, alineando los agujeros.
- Utilicé un tornillo M3*8 y una tuerca M3 de la bolsa correspondiente para fijar el soporte del motor a la placa. Atornillé hasta que el soporte estuvo firmemente asegurado.
- Para una instalación adicional, utilicé un tornillo M3*30 y otra tuerca M3 en los agujeros restantes para asegurar completamente el soporte del motor.

3. **Montaje del motor**:
   - Coloque el motor en el soporte previamente instalado.
   - Me aseguré de que el cable del motor estuviera orientado hacia el interior del chasis para facilitar las conexiones posteriores.
   - Fijé el motor al soporte utilizando los tornillos y tuercas proporcionados. Atornillé hasta que el motor quedó bien sujeto.



(Figura 3-2. Imagen del chasis con un solo motor)

4. **Repetición del proceso**:
   - Repetí los pasos anteriores para instalar los otros tres motores en las posiciones correspondientes en la placa acrílica inferior.
   - Me aseguré de que todos los cables de los motores estuvieran orientados hacia el interior del chasis para mantener un cableado ordenado y fácil de manejar.

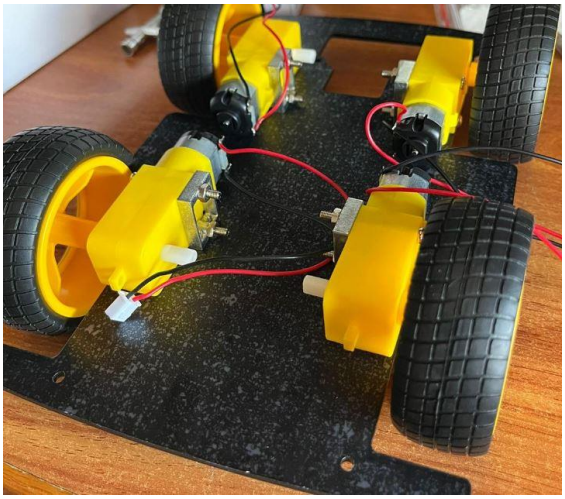


(Figura 3-3. Imagen del chasis con 4 motores)

5. **Montaje de las ruedas**:
   - Tomé una de las ruedas y la alineé con el eje del motor.

- Nota: El agujero de la rueda no es redondo; me aseguré de alinearlo correctamente con el eje del motor para una instalación adecuada.
- Empujé la rueda firmemente hasta que quedó bien ajustada al eje del motor.

6. **Repetición del proceso**:
   - Repetí el procedimiento para instalar las tres ruedas restantes en los ejes de los otros motores.

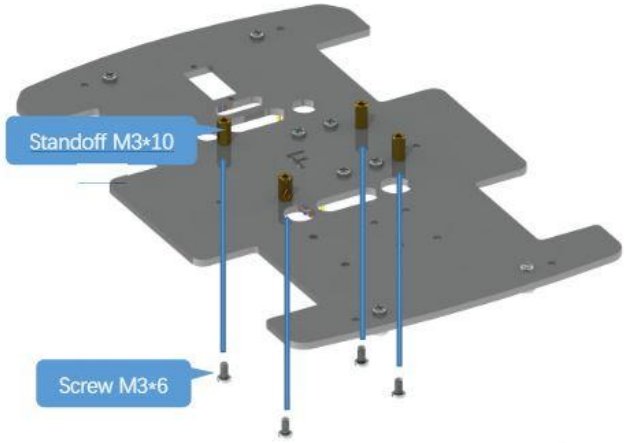


(Figura 3-4. Imagen del chasis con 4 motores y 4 ruedas)

3.1.1.2 Instalación de Arduino

Para instalar el Arduino y la placa auxiliar en el chasis del coche autónomo, seguí los pasos descritos a continuación:

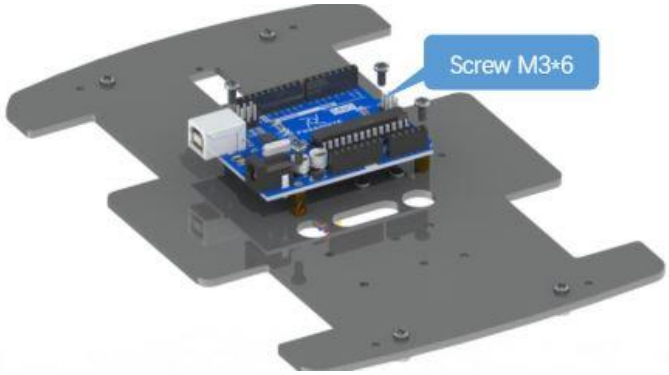
1. **Instalación de los Separadores en la Placa Acrílica Superior**: Primero, instalé los separadores (standoffs) en la placa acrílica superior del chasis. Utilicé separadores M3*10, los cuales fijé con tornillos M3*8 para asegurar que la placa acrílica esté bien sujeta y estable. Estos separadores proporcionan el espacio necesario para que la placa acrílica pueda albergar tanto el Arduino como la placa auxiliar sin que se toquen directamente con otros componentes del chasis.



(Figura 3-5. Imagen de los Separadores en la Placa Acrílica Superior)

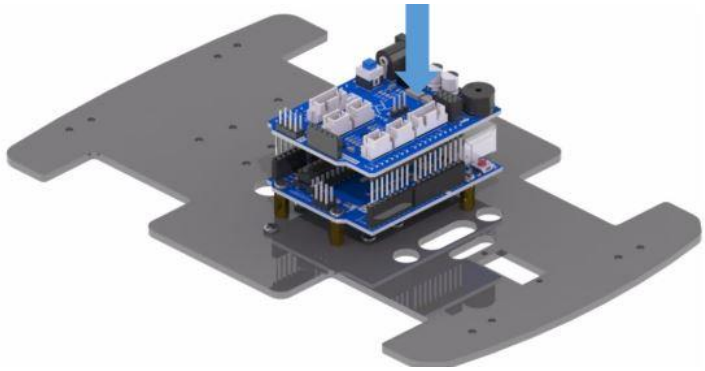
2. **Instalación de la Placa de Control Freenove**: Una vez que los separadores estuvieron en su lugar, procedí a instalar la placa de control Freenove (que es una versión de Arduino Uno) sobre los

separadores en la placa acrílica superior. Alineé los agujeros de la placa de control con los separadores y la fijé usando los tornillos M3*8 restantes. Esta placa de control será el núcleo de todas las operaciones electrónicas del coche autónomo.



(Figura 3-6. Imagen de la Placa de Control Freenove)

3. **Conexión de la Placa Auxiliar**: Después de asegurar la placa de control, conecté la placa auxiliar en la parte superior de la placa de control Freenove. Esta placa auxiliar viene con elementos pre-soldados(entre ellos el actuador de **buzzer activo**) que facilitan la conexión de los sensores y actuadores del coche. La placa auxiliar se conecta directamente a los pines de la placa de control, extendiendo las capacidades de conexión y control del sistema.



(Figura 3-7. Imagen de la Placa auxiliar)

Estos pasos aseguran que tanto el Arduino como la placa auxiliar estén firmemente instalados y listos para conectar los diversos sensores y actuadores necesarios para el funcionamiento del coche autónomo. La instalación cuidadosa y la correcta fijación de estas placas son cruciales para garantizar la estabilidad y el rendimiento del sistema.

3.1.1.3 Instalación de la batería, los sensores y modulo bluetooth

Para la alimentación del sistema, instalé un portapilas que aloja dos pilas 18650 en serie. Seguí los siguientes pasos:

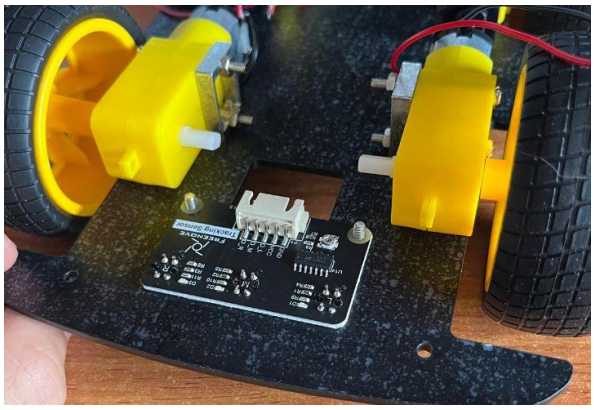
1. **Instalación del portapilas**: Fijé el portapilas en una ubicación adecuada dentro del chasis del coche autónomo utilizando un tornillo M2*10 y una tuerca M2. Aseguré que el portapilas esté firmemente sujeto para evitar movimientos que puedan desconectar las pilas durante el funcionamiento del vehículo.
2. **Conexión de la fuente de alimentación**: Conecté la fuente de alimentación del portapilas a la placa de extensión (placa superior) del kit. Esta conexión es esencial para proporcionar energía a

todos los componentes del sistema una vez que se enciende el interruptor de la placa de extensión.

(Figura 3-8. Imagen de portapilas)

A continuación, explicaré la instalación de los sensores:

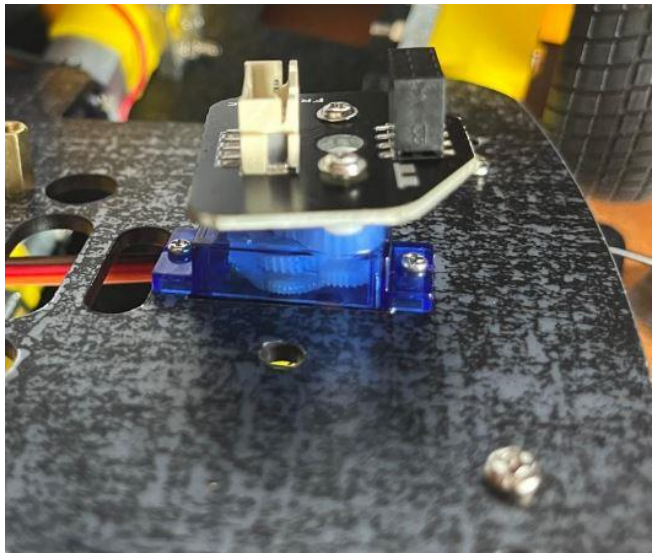
• **Sensor de tracking (seguidor de Línea)**: Instalé el módulo de seguimiento de líneas debajo del chasis del coche utilizando un tornillo M3*8 y una tuerca M3. Este módulo consta de tres sensores infrarrojos integrados que permiten al coche autónomo seguir una línea trazada en el suelo de manera precisa.



(Figura 3-9. Imagen de seguidor de línea)

(Figura 3-10. Imagen de los sensores de tracking de línea por debajo)

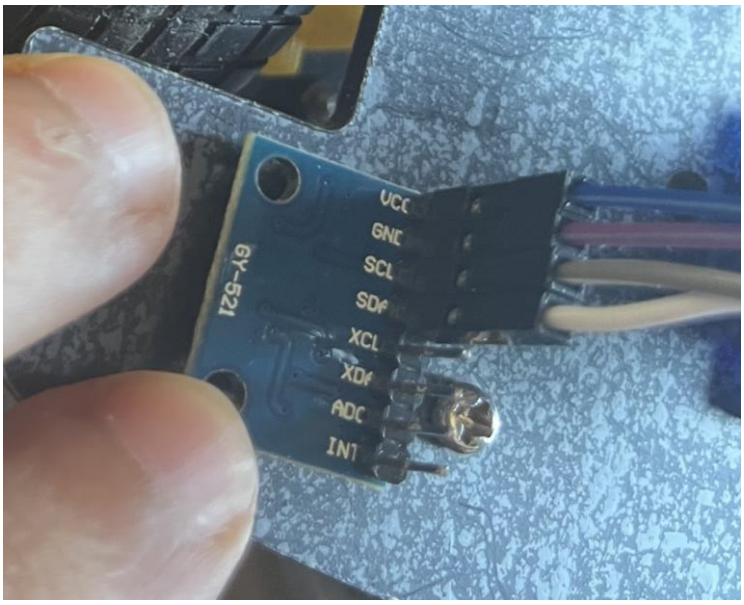
• **Sensor de ultrasonido**: El sensor de ultrasonido se montó encima del conector del módulo sónico, aprovechando el soporte y espacio diseñado para este propósito. Aunque no se utiliza el servo en la configuración actual, el montaje del sensor de ultrasonido se realizó asegurando que esté bien posicionado para medir distancias sin obstáculos.



(Figura 3-11. Imagen del conector de modulo sónico)

(Figura 3-12. Imagen del sensor de ultrasonido)

• **MPU6050**: Conecté el MPU6050 al Arduino mediante la interfaz I2C, estableciendo la comunicación necesaria para medir la temperatura y otros datos específicos del movimiento, que se explicarán detalladamente en la sección 3.3 sobre las conexiones de cables.



(Figura 2-13. Imagen del MPU6050)

Finalmente, instalé **el módulo de Bluetooth** en un conector I2C dedicado de la placa auxiliar. Este módulo se utiliza para establecer la comunicación inalámbrica con dispositivos externos, permitiendo el control y la supervisión del coche autónomo a través de aplicaciones móviles u otros medios compatibles.

3.1.2 Interacción de los componentes

El Arduino Uno es el cerebro central del sistema del coche autónomo, coordinando la interacción entre todos los componentes clave. Este microcontrolador robusto y versátil está programado para recibir datos de los sensores, procesar la información mediante algoritmos específicos, y enviar comandos precisos a los actuadores para controlar el movimiento del vehículo. Utiliza una variedad de puertos digitales y analógicos para establecer conexiones directas con los diferentes componentes del sistema.

Además, es el responsable de regular de manera inteligente de la potencia que llega a los actuadores y sensores. A través de sus salidas PWM (modulación por ancho de pulso) y la gestión de los niveles de voltaje, el Arduino Uno coordina eficazmente la potencia suministrada por el módulo de batería, asegurando un funcionamiento óptimo y eficiente de todos los componentes.

A continuación, voy a explicar cómo se interaccionan los componentes entre ellos:

- **Primeramente, los sensores infrarrojos (seguidor de línea):** Estos sensores están constantemente escaneando el suelo en busca de una línea negra sobre un fondo blanco. Cuando detectan la línea, envían señales al Arduino Uno indicando su posición relativa. Basándose en esta retroalimentación, el Arduino ajusta la velocidad y dirección de los motores DC para mantener el coche autónomo siguiendo la trayectoria definida. Es capaz de modular el voltaje suministrado a los motores mediante PWM (modulación por ancho de pulso), lo cual permite un control fino y preciso del movimiento, además tiene otra interacción con los **motores**, el **MPU650** y el **buzzer** que explico a continuación.
- **En segundo lugar, se encuentra el sensor de ultrasonido:** Montado estratégicamente para detectar obstáculos delante del coche autónomo, el sensor de ultrasonido mide la distancia hasta el objeto más cercano. Cuando detecta un obstáculo, transmite datos de distancia al Arduino, que inmediatamente evalúa la situación. Dependiendo de la proximidad del obstáculo, el Arduino puede tomar acciones como reducir la velocidad del vehículo gradualmente o detenerlo por completo.

Simultáneamente, activa el **buzzer activo** para emitir alertas sonoras escalonadas, alertando al operador o a otros usuarios cercanos sobre la presencia del obstáculo.

- **En tercer lugar, el MPU6050:** Este módulo proporciona datos precisos de medición de temperatura, en zonas específicas del escenario, como aquellas donde **los sensores infrarrojos** detectan completamente una línea negra, es decir, cuando todos los sensores de infrarrojos detectan un '1' lógico. Esto se logra mediante una breve pausa del sistema (aproximadamente medio segundo), además durante esta breve pausa el **buzzer** emite un breve sonido de aviso de medición y el Arduino uno envía los datos a través del **puerto serial** y, opcionalmente, a través del **módulo Bluetooth**.

Por último, añadir más información de lo anteriormente comentado al principio de esta subsección sobre que el Arduino uno es el responsable de regular de manera inteligente de la potencia que llega a los actuadores:

- **Motores DC:** Controlados mediante señales PWM generadas por el Arduino Uno, los motores DC reciben variaciones controladas de voltaje. Esto permite al sistema ajustar la velocidad y la dirección del coche autónomo con gran precisión. Por ejemplo, para avanzar o retroceder, el Arduino regula el voltaje suministrado a los motores DC mediante ciclos de PWM, donde la duración del pulso determina la velocidad del motor. Este proceso no solo optimiza el rendimiento del vehículo, sino que también conserva la vida útil de los motores al evitar sobrecargas.

- **Buzzer activo:** Integrado en la placa auxiliar, el buzzer activo se utiliza para emitir alertas sonoras en respuesta a diversas condiciones detectadas por los sensores. Cuando el Arduino Uno detecta un obstáculo mediante el sensor de ultrasonido, por ejemplo, modula el voltaje enviado al buzzer para generar sonidos audibles con una frecuencia y volumen específicos. Esta modulación de voltaje asegura que el buzzer emita señales claras y reconocibles, indicando al operador o a los usuarios cercanos la proximidad de un obstáculo y la necesidad de tomar medidas preventivas.

- **Módulo de batería:** Utilizando un portapilas con dos pilas 18650 en serie, el módulo de batería proporciona la energía necesaria para el funcionamiento continuo del coche autónomo. El Arduino Uno supervisa constantemente el nivel de voltaje de las baterías a través de sus entradas analógicas. Cuando el voltaje de las baterías alcanza un umbral bajo predeterminado, el Arduino puede tomar decisiones como reducir la velocidad de los motores. Esta capacidad de gestión inteligente asegura que el sistema opere de manera confiable y prolonga la autonomía del coche autónomo entre recargas.

3.1.3. Modularidad y escalabilidad

El diseño del coche autónomo se basa en principios de modularidad y escalabilidad, asegurando flexibilidad para futuras expansiones y mejoras. A continuación, se detallan los aspectos clave:

1. **Arquitectura modular:**

   El sistema está compuesto por módulos independientes que desempeñan funciones específicas y se interconectan de manera eficiente. Cada módulo, como los sensores, actuadores y el microcontrolador, está diseñado para ser fácilmente intercambiable y actualizable. Esto facilita la adaptación del coche autónomo a diferentes entornos y requisitos específicos de aplicación.

2. **Interfaz estándar:**

   Se ha implementado una interfaz estándar entre los módulos para asegurar la compatibilidad y la comunicación efectiva. Por ejemplo, los sensores utilizan protocolos de comunicación estándar

(como I2C o UART) para enviar datos al Arduino Uno, mientras que los actuadores reciben comandos precisos a través de señales PWM. Esta estandarización simplifica el proceso de integración de nuevos componentes y permite la incorporación de tecnologías emergentes con facilidad.

3. **Flexibilidad en la configuración:**

   El sistema está diseñado para ser altamente configurable según las necesidades específicas del proyecto. Los puertos digitales y analógicos del Arduino Uno permiten la conexión directa y la configuración de múltiples dispositivos periféricos. Además, el uso de una placa auxiliar facilita la disposición física y eléctrica de los componentes, optimizando el espacio y mejorando la eficiencia del sistema.

4. **Escalabilidad:**

   La arquitectura del coche autónomo está preparada para escalar en términos de capacidades y rendimiento. Esto se logra mediante la incorporación de capacidades de procesamiento adicionales, como la expansión de memoria, la integración de microcontroladores más potentes, que pueden manejar cargas de trabajo más complejas y mejorar el tiempo de respuesta del sistema, la introducción de nuevos sensores y actuadores que nos den datos y acciones nuevas o desarrollar diferentes algoritmos para ir puliendo un producto.

5. **Actualizaciones de software y hardware:**

   Se ha implementado un sistema robusto de actualización de software y hardware para mantener la modularidad y escalabilidad del proyecto a lo largo del tiempo. Las actualizaciones de firmware del Arduino Uno y otros componentes se realizan de manera sencilla a través de conexiones inalámbricas o por cable, garantizando la compatibilidad con las últimas tecnologías y mejoras en el rendimiento.

6. **Compatibilidad con futuras expansiones:**

   El diseño modular y escalable del coche autónomo asegura su compatibilidad con futuras expansiones y desarrollos tecnológicos. Esto incluye la capacidad de integrar nuevos sensores, actuadores o módulos de procesamiento sin requerir modificaciones significativas en la arquitectura principal. De este modo, el proyecto puede evolucionar y adaptarse a nuevas aplicaciones y requisitos con facilidad.

3.2. Selección y justificación de componentes

La selección de los componentes del coche autónomo se realizó principalmente a partir del kit "Freenove 4WD Car Kit", el cual ofrece una solución completa y modular para el desarrollo de proyectos de robótica. Este kit estaba disponible tanto con una Raspberry Pi como con un Arduino Uno. Se optó por el Arduino Uno debido a su facilidad de programación y manejo, así como a su alta compatibilidad con una amplia variedad de módulos y componentes adicionales, lo que facilita la expansión y el desarrollo de proyectos más complejos. Además, el Arduino Uno tiene un robusto entorno de desarrollo y una comunidad extensa, lo que proporciona un gran soporte técnico y numerosos recursos.

3.2.1 Microcontrolador: Arduino Uno

Selección: El Arduino Uno fue seleccionado como el microcontrolador principal del "Freenove 4WD Car

Kit".

Justificación:

- **Facilidad de Programación:** El Arduino Uno es conocido por su facilidad de programación, lo que permite un desarrollo más rápido y eficiente.
- **Compatibilidad:** Ofrece una amplia compatibilidad con diversos sensores y actuadores, facilitando la expansión del proyecto.
- **Comunidad de Soporte:** La extensa comunidad de Arduino proporciona abundante documentación, recursos y soporte técnico, lo que facilita la resolución de problemas y la implementación de mejoras.

3.2.2 Placa auxiliar

Selección: La placa auxiliar incluida en el kit facilita las conexiones entre los diversos componentes, optimizando la disposición física y eléctrica del sistema.

Justificación:

- **Conexiones Simplificadas:** Incluye elementos pre-soldados que simplifican la instalación de sensores y actuadores.
- **Interfaz de Expansión:** Proporciona interfaces específicas, como la conexión para el módulo de Bluetooth, permitiendo la fácil integración de módulos adicionales.

3.2.3 Sensores de entrada

1. **Sensores Infrarrojos (Seguidor de Línea):**

Selección: Los sensores infrarrojos del kit son ideales para detectar líneas negras sobre fondos blancos, permitiendo al coche seguir una trayectoria predefinida.

Justificación:

- **Precisión:** Proporcionan lecturas precisas y rápidas, esenciales para mantener el coche en la trayectoria correcta.
- **Facilidad de integración:** Son compatibles con el Arduino Uno y se configuran fácilmente a través de la placa auxiliar.

2. **Sensor de ultrasonido:**

Selección: El sensor de ultrasonido del kit fue seleccionado por su capacidad para detectar obstáculos a una distancia considerable y con precisión.

Justificación:

- **Detección de obstáculos:** Permite al coche evitar colisiones mediante la medición de distancias y la activación de sistemas de alerta y frenado.
- **Compatibilidad:** Fácil integración con el Arduino Uno y otros componentes del sistema.

3. **MPU6050 (acelerómetro y giroscopio):**

Selección: Aunque el MPU6050 no estaba incluido en el kit, se eligió para proporcionar datos de movimiento y temperatura adicionales.

Justificación:

- **Disponibilidad:** Este componente ya estaba disponible de un proyecto personal anterior, lo que facilitó su incorporación sin costo adicional.
- **Interfaz I2C:** Su uso mediante la interfaz I2C añade complejidad y valor educativo al proyecto, demostrando la capacidad del sistema para integrar y manejar múltiples protocolos de comunicación.
- **Datos de movimiento y temperatura:** Proporciona mediciones precisas de aceleración, giroscopio y temperatura, útiles para implementar algoritmos avanzados de control de movimiento y monitoreo ambiental.

3.2.4 Actuadores

1. **Motores DC:**

Selección: Se seleccionaron motores DC controlados mediante PWM para permitir el movimiento del coche, incluidos en el kit.

Justificación:

- **Control preciso:** Permiten un control fino de la velocidad y dirección del coche autónomo.
- **Versatilidad:** Son compatibles con el Arduino Uno y pueden ser fácilmente controlados mediante señales PWM.

2. **Buzzer activo:**

Selección: El buzzer activo integrado en la placa auxiliar fue seleccionado para proporcionar alertas sonoras.

Justificación:

- **Alertas Sonoras:** Emite sonidos para notificar eventos críticos como la detección de obstáculos.
- **Facilidad de Uso:** Su integración en la placa auxiliar simplifica su uso y control.

3.2.5 Módulo de Bluetooth

Selección: El módulo de Bluetooth incluido en el kit permite la comunicación inalámbrica con dispositivos externos.

Justificación:

- **Control remoto:** Permite la supervisión y control del coche autónomo a través de aplicaciones móviles u otros dispositivos compatibles.
- **Integración fácil:** Compatible con la placa auxiliar y el Arduino Uno, lo que facilita su implementación.

3.2.6 Sistema de alimentación

Selección: El sistema de alimentación se basa en un módulo de batería portátil que utiliza dos pilas 18650 en serie, incluido en el kit.

Justificación:

- **Potencia suficiente:** Proporciona la energía necesaria para todos los componentes del sistema.
- **Portabilidad:** Las pilas 18650 son recargables y tienen una alta capacidad de energía, ideal para aplicaciones móviles.

3.2.7 Chasis del coche

Selección: El chasis del coche incluido en el kit proporciona una estructura física robusta y adecuada para montar todos los componentes.

Justificación:

- **Durabilidad:** Proporciona una base estable y duradera para todos los componentes del coche autónomo.
- **Facilidad de montaje:** Diseñado para facilitar el montaje y la organización de los componentes.

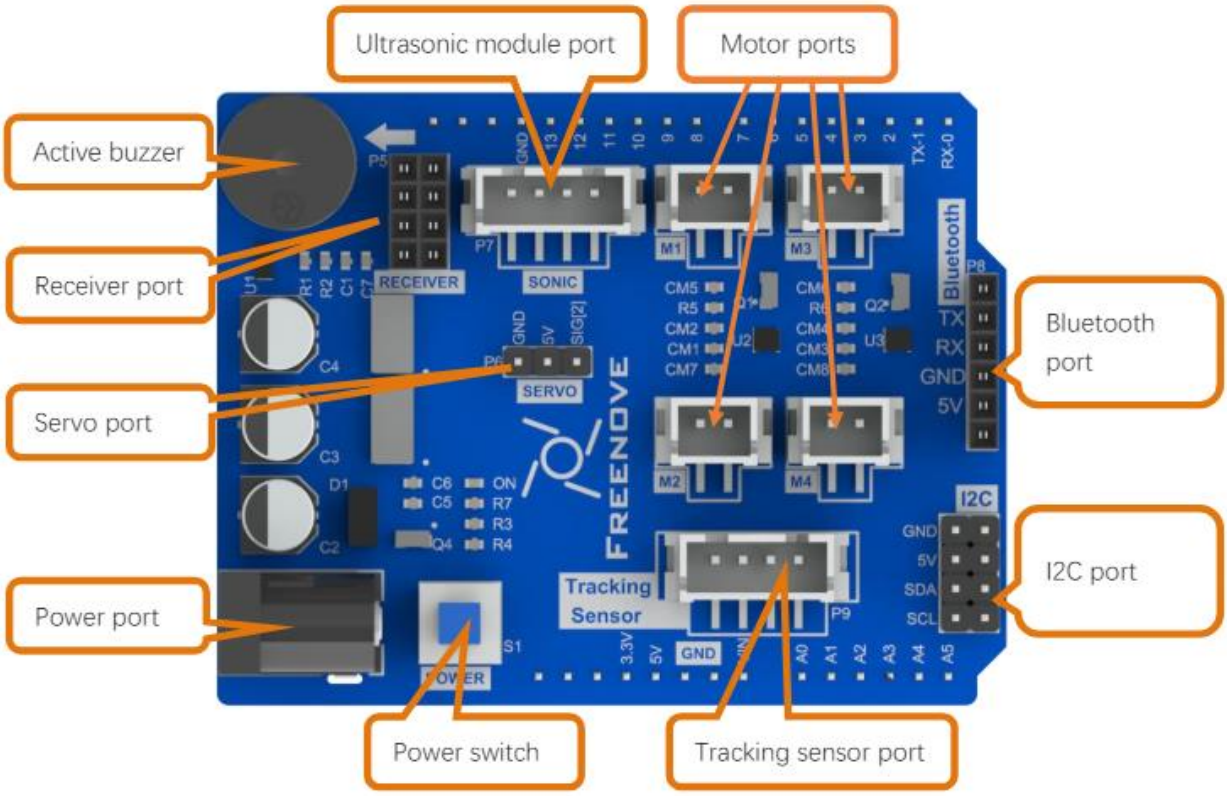
3.2 Bloques principales y conexiones detalladas

1. **Microcontrolador Arduino Uno**
    - **Entradas:**
        - Sensores de infrarrojos: conectados a los pines analógicos A1, A2, A3
        - Sensor de ultrasonido: pin Trigger conectado a D7, pin Echo conectado a D8
        - MPU6050: pines I2C A4 (SDA) y A5 (SCL)
    - **Salidas:**
        - Motores DC:
            - Dirección: pin D3 (derecha) y pin D4 (izquierda)
            - Velocidad controlada por PWM: pin D5 (derecha) y pin D6 (izquierda)
        - Buzzer activo: pin analógico A0
        - Módulo Bluetooth: pines UART D0 (RX) y D1 (TX)
    - **Alimentación:**
        - Conectado al módulo de batería (Vin y GND)
2. **Sensores de infrarrojos (x3)**
    - **Conexiones:**
        - Cada sensor está conectado a un pin analógico del Arduino (A1, A2, A3)
        - Integrados en una placa del kit, conectados a la placa auxiliar.
3. **Sensor de ultrasonido**
    - **Conexiones:**
        - Pin Trigger conectado a D7
        - Pin Echo conectado a D8
        - Conectado al conector de módulo sónico en la placa auxiliar.
4. **MPU6050**
    - **Conexiones:**
        - SDA conectado al pin A4 del Arduino
        - SCL conectado al pin A5 del Arduino
        - Alimentación conectada a Vcc y GND del Arduino.
5. **Motores DC (x4)**
    - **Conexiones:**
        - Dirección: pin D3 (derecho) y pin D4 (izquierdo)
        - Velocidad (PWM): pin D5 (derecho) y pin D6 (izquierdo)
        - Alimentación del controlador de motor conectada a la batería.
        - Conectados mediante cables a enchufes presoldados en la placa auxiliar.
6. **Buzzer activo**
    - **Conexiones:**

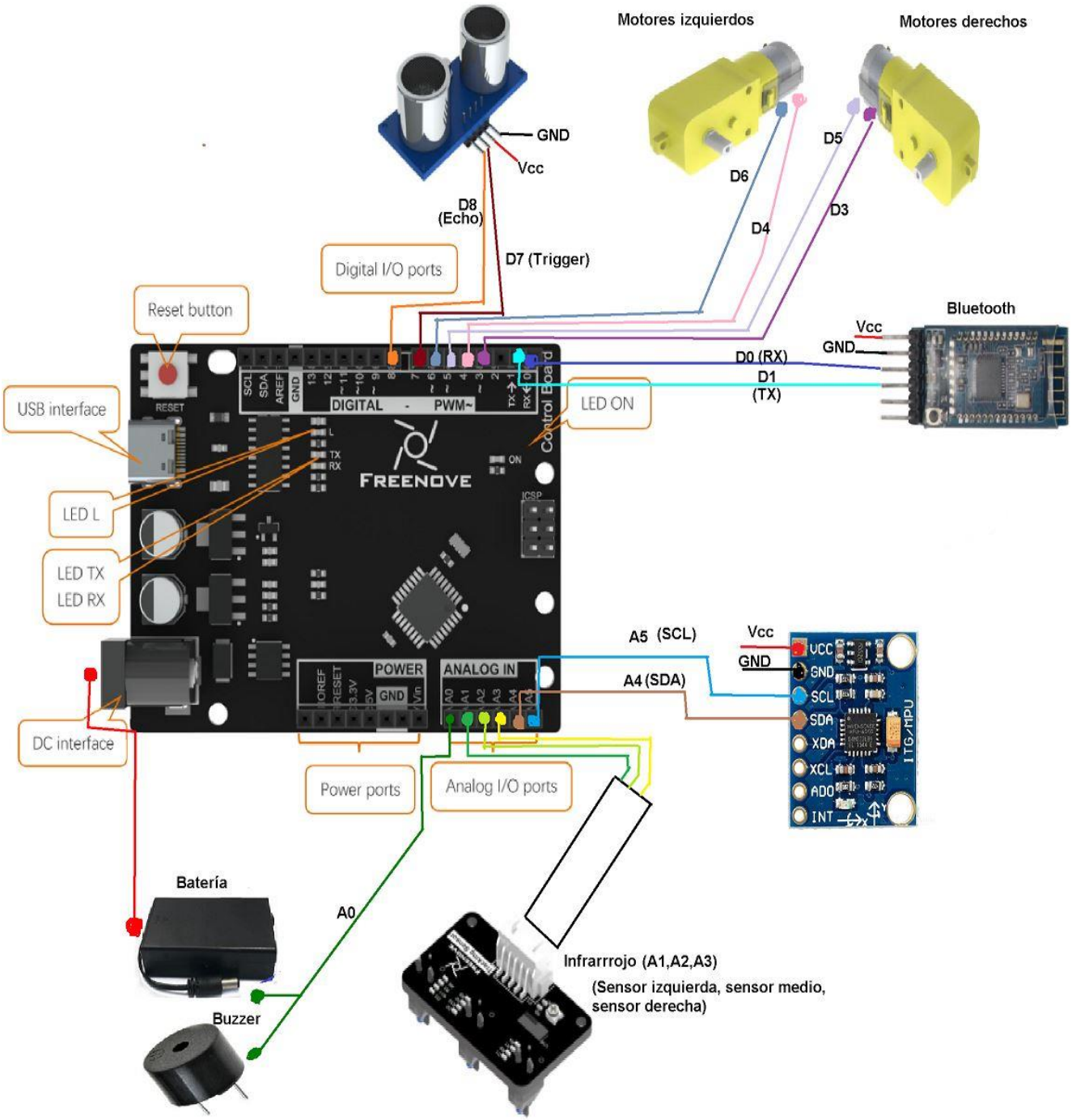
- Pin de señal conectado a A0
- Presoldado en la placa auxiliar.

7. **Módulo Bluetooth**
   - **Conexiones:**
     - TX conectado a D0 (RX) del Arduino
     - RX conectado a D1 (TX) del Arduino
     - Alimentación conectada a 3.3V y GND del Arduino
     - Conectado directamente a la placa auxiliar.
8. **Módulo de batería**
   - **Conexiones:**
     - Salida de voltaje conectada a Vin del Arduino y al controlador de motor
     - GND compartido con el Arduino y el controlador de motor.

Para facilitar la comprensión de las conexiones detalladas anteriormente, se ha preparado una representación visual que muestra cómo están interconectados los componentes principales del proyecto. A continuación, se presenta una imagen con anotaciones que destacan las conexiones físicas entre el Arduino Uno, los sensores, los actuadores y otros módulos:



(Figura 3-14. Placa auxiliar explicada)



(Figura 3-15. Esquema de conexiones entre componentes)

4 IMPLEMENTACIÓN PRACTICA

En esta sección se detalla la ejecución práctica del proyecto, desglosando los aspectos clave del desarrollo y la integración de los componentes y algoritmos. A continuación, se describen los puntos principales abordados en esta fase:

4.1 Desarrollo del algoritmo de seguimiento de línea

El seguimiento de línea es una de las tareas más importantes para el coche autónomo, ya que le permite mantener su trayectoria sobre un camino predefinido. Para implementar esta funcionalidad, he desarrollado un algoritmo de seguimiento de línea utilizando los sensores infrarrojos incluidos en el kit "Freenove 4WD Car Kit".

Antes de profundizar en el código del algoritmo, es importante señalar que, como se ha detallado en los fundamentos teóricos, en las subsecciones "2.1. Seguimiento de línea con sensores infrarrojos" y "2.6 Sistema de movimiento", primero se debe definir la asignación de pines para los sensores de infrarrojos y los actuadores de los motores. Aquí, nos enfocaremos en el desarrollo específico del algoritmo, omitiendo la repetición de información previamente explicada.

A continuación, se presenta el código del algoritmo de seguimiento de línea en Arduino:

```
//definición de diferentes niveles de velocidades
int tk_VoltageCompensationToSpeed;  //definición de la compensación de voltaje
#define TK_TURN_SPEED_LV4           (195 + tk_VoltageCompensationToSpeed)
#define TK_TURN_SPEED_LV3           (155 + tk_VoltageCompensationToSpeed)
#define TK_TURN_SPEED_LV2           (-130 + tk_VoltageCompensationToSpeed)
#define TK_TURN_SPEED_LV1           (-150 + tk_VoltageCompensationToSpeed)
#define TK_STOP_SPEED          0
#define TK_FORWARD_SPEED        (95 + tk_VoltageCompensationToSpeed    )

void loop() {
      trackLine();
}

//función del algoritmo seguido para el seguimiento de línea
void trackLine() {
  u8 trackingSensorVal = 0;
  trackingSensorVal = getTrackingSensorVal();

  switch (trackingSensorVal) {
    case 0:   // 000
      motorRun(TK_FORWARD_SPEED, TK_FORWARD_SPEED);
      break;
    case 7:   // 111
      motorRun(TK_STOP_SPEED, TK_STOP_SPEED);
      delay(500);
```

```
      motorRun(TK_FORWARD_SPEED, TK_FORWARD_SPEED);
      break;*/
    case 1:   // 001
      motorRun(TK_TURN_SPEED_LV4, TK_TURN_SPEED_LV1);
      break;
    case 3:   // 011
      motorRun(TK_TURN_SPEED_LV3, TK_TURN_SPEED_LV2);
      break;
    case 2:   // 010
    case 5:   // 101
      motorRun(TK_FORWARD_SPEED, TK_FORWARD_SPEED);
      break;
    case 6:   // 110
      motorRun(TK_TURN_SPEED_LV2, TK_TURN_SPEED_LV3);
      break;
    case 4:   // 100
      motorRun(TK_TURN_SPEED_LV1, TK_TURN_SPEED_LV4);
      break;
    default:
      break;
  }
}
```

4.1.1 Descripción del algoritmo

1. **Lectura de Sensores**: El algoritmo comienza leyendo los valores de los sensores de infrarrojos utilizando la función getTrackingSensorVal(). Estos sensores proporcionan una lectura binaria indicando si están detectando la línea negra (1) o el fondo blanco (0).
2. **Decisiones Basadas en Sensores**:
    - **Caso 0 (000)**: Ninguno de los sensores detecta la línea. Esto generalmente indica que el coche está completamente fuera de la línea. En este caso, el coche avanza recto para intentar volver a encontrar la línea.
    - **Caso 7 (111)**: Todos los sensores detectan la línea. Este caso especial se usa para detener el coche brevemente y luego reanudar el avance. Esto podría ser útil en ciertas aplicaciones donde se necesita una pausa en puntos específicos del recorrido, como en nuestro caso para realizar una medición de temperatura.
    - **Caso 1 (001)**: Solo el sensor derecho detecta la línea, indicando que el coche debe girar a la izquierda para realinearse con la línea.
    - **Caso 3 (011)**: Los sensores derecho y medio detectan la línea, indicando un pequeño desvío hacia la derecha y el coche necesita corregir ligeramente a la derecha.
    - **Caso 2 (010)** y **Caso 5 (101)**: El sensor medio o ambos sensores lateral y medio detectan la línea. En ambos casos, el coche debe avanzar recto.
    - **Caso 6 (110)**: Los sensores izquierdo y medio detectan la línea, indicando un pequeño desvío hacia la izquierda y el coche necesita corregir ligeramente a la izquierda.
    - **Caso 4 (100)**: Solo el sensor izquierdo detecta la línea, indicando que el coche debe girar a la derecha para realinearse con la línea.
3. **Control de Motores**: Dependiendo de la lectura de los sensores, el algoritmo ajusta la velocidad y dirección de los motores utilizando la función motorRun(). Esto se logra modulando la velocidad de los motores derecho e izquierdo mediante PWM (modulación por ancho de pulso), lo que permite giros precisos y un control suave del movimiento del coche.
4. **Función loop():** En cada ciclo del loop(), se llama a la función trackLine(). Esto asegura que el coche ajuste continuamente su trayectoria en tiempo real, basándose en las lecturas actuales de los

sensores de infrarrojos. La función loop() es el corazón del programa Arduino, donde se ejecutan repetidamente todas las instrucciones que se encuentran dentro de ella

4.1.2 Estrategias de control

- **Avance Recto**: Cuando los sensores indican que el coche está bien alineado con la línea, se mantiene una velocidad constante hacia adelante.
- **Giros**: Los giros se manejan ajustando la velocidad relativa de los motores derecho e izquierdo. Por ejemplo, para un giro a la izquierda, se disminuye la velocidad del motor izquierdo mientras se mantiene o aumenta la velocidad del motor derecho.
- **Detección de Línea Completa (111)**: En ciertas aplicaciones, es necesario detener el coche en puntos específicos. Al detectar todos los sensores la línea, se detiene el coche brevemente antes de continuar.

4.2 Integración de la detección de obstáculos

La integración de la detección de obstáculos es crucial para asegurar la seguridad del coche autónomo, permitiendo evitar colisiones con objetos o personas en su camino. Para entender completamente el algoritmo de detección de obstáculos, es recomendable revisar los puntos "2.2. Detección de obstáculos con sensor ultrasonido" y el "Sistema de alertas".

A continuación, se presenta el desarrollo del algoritmo implementado en Arduino para la detección de obstáculos y las estrategias de control asociadas.

4.2.1 Descripción del algoritmo

El siguiente código de Arduino define el comportamiento del coche cuando se encuentra con obstáculos a diferentes distancias:

```
#define OBSTACLE_DISTANCE           30
#define OBSTACLE_DISTANCE_MID       20
#define OBSTACLE_DISTANCE_LOW       10
void loop() {
    u8 trackingSensorVal = 0;
    trackingSensorVal = getTrackingSensorVal(); // Get sensor value
    int distance = getSonar();
    if (distance < OBSTACLE_DISTANCE_LOW) {
        motorRun(TK_STOP_SPEED, TK_STOP_SPEED);
        setBuzzer(true);
    }else if (distance < OBSTACLE_DISTANCE_MID) {
        setBuzzer(true);
        delay(20);
        setBuzzer(false);
        trackLine();
    }else if (distance < OBSTACLE_DISTANCE && distance > OBSTACLE_DISTANCE_MID)
{
        setBuzzer(true);
        delay(100);
        setBuzzer(false);
        trackLine();
    } else {
        setBuzzer(false);
```

```
        trackLine();
    }
    delay(10);
}
```

Este código realiza las siguientes acciones detalladas:

- **getSonar():** Obtiene la distancia medida por el sensor de ultrasonido.
- **Comparaciones de Distancia:** Se definen tres umbrales de distancia: OBSTACLE_DISTANCE_LOW (10 cm), OBSTACLE_DISTANCE_MID (20 cm) y OBSTACLE_DISTANCE (30 cm). Dependiendo de la distancia medida por el sensor, el coche autónomo ajusta su comportamiento para evitar colisiones. Cuando la distancia es menor que OBSTACLE_DISTANCE_LOW, el coche se detiene por completo y activa un buzzer continuamente para alertar sobre un obstáculo inminente. Si la distancia está entre OBSTACLE_DISTANCE_LOW y OBSTACLE_DISTANCE_MID, el buzzer emite un sonido intermitente mientras el coche continúa siguiendo la línea. Por último, si la distancia es mayor que OBSTACLE_DISTANCE_MID, el coche sigue su trayectoria sin activar el buzzer, asegurando un movimiento fluido y seguro en entornos con obstáculos.
- **setBuzzer(true/false):** Activa y desactiva el buzzer según la distancia del obstáculo detectado.
- **trackLine():** Continúa el movimiento del coche siguiendo la línea después de manejar la detección de obstáculos

4.2.2 Estrategias de control

Para asegurar una respuesta adecuada y segura a la presencia de obstáculos, el algoritmo implementa las siguientes estrategias de control:

1. **Parada de Emergencia:** Cuando se detecta un obstáculo a una distancia muy corta (menor a 10 cm), el coche se detiene inmediatamente para evitar colisiones. El buzzer se activa constantemente para alertar de la situación.
2. **Alerta de Proximidad Media:** Cuando se detecta un obstáculo a una distancia media (menor a 20 cm), el coche sigue su camino, pero activa el buzzer de manera intermitente con una frecuencia alta. Esto alerta a posibles personas o vehículos cercanos de la proximidad del coche.
3. **Alerta de Proximidad Lejana:** Cuando se detecta un obstáculo a una distancia larga (menor a 30 cm), el coche sigue su camino, pero activa el buzzer de manera intermitente con una frecuencia baja. Esto proporciona una advertencia temprana de la presencia del coche en el área.
4. **Continuación Normal:** Si no se detectan obstáculos a una distancia menor de 30 cm, el coche sigue su camino normalmente sin activar el buzzer, permitiendo una navegación fluida y sin interrupciones.

4.3 Incorporación de la medición de temperatura

La incorporación de la medición de temperatura es un componente adicional del coche autónomo que le permite recopilar y transmitir datos de temperatura. Esto puede ser útil para monitorear el entorno en el que opera el coche, así como para asegurar su correcto funcionamiento. Para entender completamente el algoritmo de medición de temperatura, es recomendable revisar los puntos "2.3. Medición de temperatura", “2.7. Módulo Bluetooth” y los anteriormente mencionados.

A continuación, se presenta el desarrollo del algoritmo implementado en Arduino para la medición de temperatura y las estrategias de control asociadas.

4.3.1 Descripción del algoritmo

El siguiente código de Arduino define el comportamiento del coche cuando se encuentra con una línea gruesa (donde todos los sensores de infrarrojos detectan negro) y realiza una medición de temperatura:

```
void loop() {
    u8 trackingSensorVal = 0;
    trackingSensorVal = getTrackingSensorVal();
    int distance = getSonar();
    if (trackingSensorVal == 7){
      setBuzzer(true);
      motorRun(TK_STOP_SPEED, TK_STOP_SPEED);
      float temperature = getMPU6050Temperature();
      Serial.print("Temperatura del MPU6050: ");
      Serial.print(temperature);
      Serial.println(" °C");
      BTSerial.print("Temperatura: ");
      BTSerial.print(temperature);
      BTSerial.println(" °C");
      if (isnan(temperature) || temperature < -20 || temperature > 100) {
        motorRun(TK_STOP_SPEED, TK_STOP_SPEED);
        setBuzzer(true);
        Serial.println("Error: Lectura de temperatura no válida");
        while (1);
        }
      delay(500);
      setBuzzer(false);
      motorRun(TK_FORWARD_SPEED, TK_FORWARD_SPEED);
    }
    if (distance < OBSTACLE_DISTANCE_LOW) {
        motorRun(TK_STOP_SPEED, TK_STOP_SPEED);
        setBuzzer(true);
    }else if (distance < OBSTACLE_DISTANCE_MID) {
        setBuzzer(true);
        delay(20);
        setBuzzer(false);
        trackLine();
    }else if (distance < OBSTACLE_DISTANCE && distance > OBSTACLE_DISTANCE_MID)
{
        setBuzzer(true);
        delay(100);
        setBuzzer(false);
        trackLine();
    } else {
        setBuzzer(false);
        trackLine();
    }
    delay(10);
}
```

Este código realiza las siguientes acciones detalladas:

1. **Detección de línea gruesa (111):**
    - **getTrackingSensorVal():** Obtiene el valor de los sensores de infrarrojos para determinar si todos los sensores detectan una línea negra gruesa.
    - **setBuzzer(true):** Activa el buzzer para señalizar la detección.
    - **motorRun (TK_STOP_SPEED, TK_STOP_SPEED):** Detiene el movimiento del coche.
    - **getMPU6050Temperature ():** Obtiene la temperatura actual del sensor MPU6050.
    - **Serial.print():** Imprime la temperatura en el monitor serial.
    - **BTSerial.print():** Envía la temperatura por Bluetooth.
    - **Validación de Temperatura:** Comprueba si la temperatura está dentro del rango válido (-20 °C a 100 °C). Si no lo está, detiene el coche y activa el buzzer continuamente.
    - **delay(500):** Espera 500 milisegundos antes de reanudar.
    - **setBuzzer(false):** Desactiva el buzzer.
    - **motorRun(TK_FORWARD_SPEED, TK_FORWARD_SPEED):** Reanuda el movimiento hacia adelante.
2. **Detección de obstáculos:**
    - **getSonar():** Obtiene la distancia medida por el sensor de ultrasonido.
    - **Comparaciones de Distancia:** Evalúa la distancia medida para determinar la presencia y la cercanía de obstáculos.
    - **setBuzzer(true/false):** Activa y desactiva el buzzer según la distancia del obstáculo detectado.
    - **trackLine():** Continúa el movimiento del coche siguiendo la línea después de manejar la detección de obstáculos.

4.3.2 Estrategias de control

Para asegurar que el coche autónomo pueda recopilar y transmitir datos de temperatura de manera efectiva y segura, el algoritmo implementa las siguientes estrategias de control:

1. **Detención para medición:** Cuando los sensores de infrarrojos detectan una línea gruesa, el coche se detiene para realizar la medición de temperatura. Esto asegura que la lectura sea estable y precisa.
2. **Transmisión de datos:** La temperatura medida se imprime en el monitor serial y se envía a través del módulo Bluetooth. Esto permite la monitorización remota de la temperatura, lo cual es útil para aplicaciones que requieren el seguimiento de las condiciones ambientales.
3. **Validación de lectura:** La temperatura medida se validan para asegurar que está dentro de un rango aceptable. Si la lectura es inválida, el coche se detiene y el buzzer se activa de forma continua para indicar un error. Esto previene posibles problemas causados por lecturas incorrectas del sensor.
4. **Reanudación de movimiento:** Después de realizar y transmitir la medición de temperatura, el coche reanuda su movimiento siguiendo la línea, asegurando que no haya interrupciones prolongadas en su trayecto.

Estas estrategias aseguran que el coche autónomo pueda recopilar y transmitir datos de temperatura de manera confiable, mientras continúa operando de manera segura y eficiente.

5 RESULTADOS EXPERIMENTALES

5.1. Pruebas de funcionamiento y validación

Durante el desarrollo del proyecto, se llevaron a cabo una serie de experimentos para evaluar el rendimiento y validar el sistema del coche autónomo en diferentes configuraciones y escenarios simulados. A continuación, se detallan los resultados y aprendizajes obtenidos de cada experimento:

Experimento 1: Evaluación inicial del seguimiento de línea

Para la primera prueba, se configuró un escenario casero utilizando folios unidos y dispuestos en forma de un circuito en forma de '0'. La línea del circuito estaba delineada con cinta negra sobre un fondo blanco. El objetivo principal era verificar la capacidad del coche autónomo para seguir la línea trazada. Sin embargo, este experimento reveló desafíos significativos. La fricción con los folios afectaba la adherencia del coche al suelo, causando desviaciones inesperadas. Además, los pliegues en el papel blanco confundían a los sensores de infrarrojos, resultando en lecturas incorrectas que afectaban el seguimiento preciso de la línea.

(Figura 5-1. Imagen del escenario del experimento 1)

Experimento 2: Optimización del escenario de prueba

Para mejorar la precisión, se modificó el escenario colocando directamente la cinta negra sobre el suelo. Este ajuste proporcionó una superficie más uniforme y permitió una detección más fiable de la línea negra. Aunque este ajuste mostró mejoras significativas, aún se observaron desafíos en las esquinas del circuito, donde el coche a veces tendía a desviarse debido a la altura inadecuada de los sensores sobre el chasis del coche.

Experimento 3: Reubicación de los sensores y validación

Se decidió reubicar los sensores por debajo del chasis del coche para reducir la altura de detección. Esta modificación fue crucial, ya que el coche comenzó a seguir la línea de manera más consistente y precisa. La reubicación demostró la importancia de la colocación adecuada de los sensores, aunque fuera solo un mm para optimizar el rendimiento del algoritmo de seguimiento de línea.

Experimento 4: Integración de la detección de obstáculos

Con el seguimiento de línea validado, se procedió a integrar la detección de obstáculos en el mismo escenario de prueba. Se ajustaron las distancias de detección para asegurar que el coche respondiera de manera efectiva ante la presencia de obstáculos mientras seguía la línea trazada. Este paso fue exitoso desde el inicio, resaltando la capacidad del sistema para manejar múltiples funciones simultáneamente.

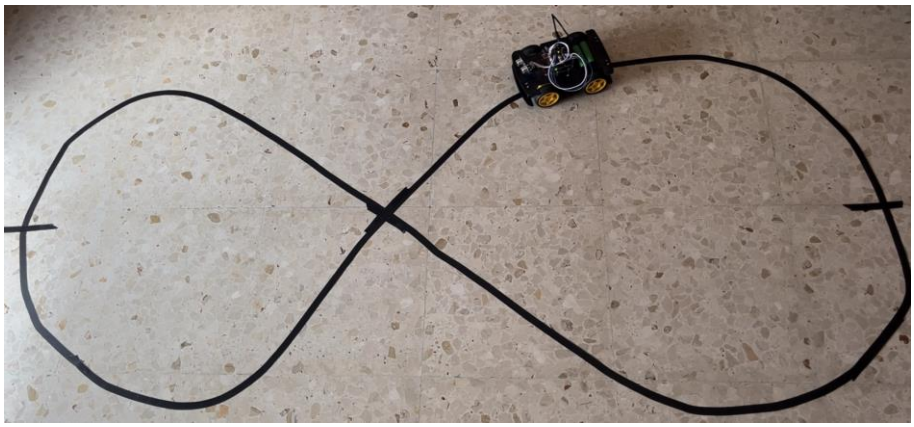
Experimento 5: Incorporación de líneas perpendiculares y mediciones de temperatura

Para aumentar la complejidad del escenario, se añadieron secciones con líneas negras perpendiculares dentro del circuito en forma de '0'. Esta adición permitió probar el algoritmo bajo condiciones más desafiantes, donde además de seguir la línea principal, el coche realizaba mediciones de temperatura en puntos estratégicos del circuito. Tras ajustar el algoritmo para manejar estas nuevas condiciones, el experimento fue exitoso, validando la capacidad del coche autónomo para realizar múltiples tareas de forma eficiente.

(Figura 5-2. Imagen del escenario del experimento 5)

Experimento 6: Desafío final con circuito en forma de '8'

Para evaluar la robustez del sistema, se diseñó un circuito en forma de '8' que incluía cuatro paradas para medir la temperatura en diferentes ubicaciones. Este experimento demostró la versatilidad del algoritmo y la capacidad del coche para navegar de manera efectiva en entornos complejos, manteniendo un seguimiento preciso de la línea y realizando mediciones de temperatura sin dificultades significativas.



(Figura 5-3. Imagen del escenario del experimento 6)

5.2. Evaluación de la precisión y fiabilidad

En este apartado, se analiza la precisión y la fiabilidad del sistema desarrollado a través de pruebas específicas y análisis detallados:

- **Precisión del Seguimiento de Línea:** Se evaluó la capacidad del coche autónomo para seguir la línea negra con precisión. Se observaron las correcciones realizadas por el algoritmo para mantener la trayectoria deseada en diferentes configuraciones de circuito y condiciones ambientales. La modificación de la ubicación de los sensores infrarrojos debajo del chasis mejoró significativamente la adherencia del coche al seguir la línea, minimizando los errores causados por pliegues en los folios y variaciones en la superficie.
- **Fiabilidad en la Detección de Obstáculos:** Se realizaron pruebas para verificar la fiabilidad del sistema en la detección y evitación de obstáculos. Se ajustaron las distancias de detección y se evaluaron las tasas de detección correcta. Aunque las mediciones mostraron variaciones dependientes del entorno, el sistema demostró ser efectivo al activar alertas sonoras progresivamente intensas a medida que se acercaba a los obstáculos, asegurando así la detención anticipada del coche y evitando colisiones.
- **Consistencia en la Medición de Temperatura:** Se examinó la consistencia y precisión de las mediciones de temperatura realizadas por el coche autónomo en varios puntos del circuito. Aunque se observaron pequeñas variaciones de 0.1 a 0.3 grados Celsius, las mediciones fueron consistentemente fiables y coincidieron dentro del rango esperado. Esto validó la exactitud del sensor de temperatura integrado, crucial para la monitorización ambiental y la seguridad del sistema.

5.3. Análisis de desempeño y eficiencia

Este apartado se enfoca en evaluar el desempeño general del sistema y su eficiencia operativa:

- **Velocidad y tiempo de respuesta:** Se midió la velocidad de respuesta del coche autónomo al seguir la línea y detectar obstáculos. El sistema demostró una respuesta rápida y eficaz al ajustar la dirección y la velocidad, garantizando un movimiento suave y seguro incluso en entornos dinámicos. La optimización de las estrategias de control contribuyó a mejorar aún más la eficacia del sistema en diferentes condiciones de circuito.
- **Consumo de recursos:** Se registró el consumo de energía durante las pruebas para evaluar la eficiencia energética del sistema. Se compararon varias configuraciones y estrategias de control para optimizar el uso de la batería y prolongar la autonomía del coche autónomo. La implementación de medidas de eficiencia energética resultó en un uso más efectivo de los recursos, asegurando una operación prolongada sin comprometer el desempeño.
- **Robustez y estabilidad:** Se analizó la capacidad del sistema para mantener un funcionamiento robusto y estable a lo largo de pruebas extensivas. Se realizaron pruebas bajo diferentes cargas de trabajo y variaciones ambientales para verificar la estabilidad del coche autónomo. Aunque el sistema demostró una alta robustez en condiciones controladas, se identificaron desafíos en superficies con baja fricción y en circuitos con pliegues o irregularidades, afectando ocasionalmente la precisión del seguimiento de línea y la detección de obstáculos.

6 DISCUSIÓN Y CONCLUSIONES

6.1. Limitaciones y Recomendaciones para Futuros Trabajos

Aunque el sistema ha demostrado ser robusto en condiciones ideales, existen algunas limitaciones identificadas durante las pruebas experimentales. Las principales limitaciones incluyen la variabilidad en la precisión de la detección de obstáculos a diferentes distancias y la sensibilidad a condiciones ambientales variables como la iluminación y la temperatura. Además, se observó que las mediciones de temperatura, aunque consistentes, presentaron pequeñas variaciones que podrían afectar la precisión en aplicaciones sensibles.

Para futuros trabajos, se recomienda explorar las siguientes áreas:

- **Optimización de algoritmos:** Mejorar los algoritmos de seguimiento de línea y detección de obstáculos para adaptarse mejor a condiciones adversas y variables.
- **Calibración y ajustes:** Realizar ajustes adicionales en los sensores para mejorar la precisión y consistencia de las mediciones de temperatura, así como la detección de obstáculos a diferentes distancias.
- **Incorporación de sensores adicionales:** Explorar la integración de sensores adicionales para mejorar la percepción del entorno, como cámaras o sensores de proximidad más avanzados.
- **Mapeo del entorno:** Implementar una nueva funcionalidad para realizar un mapeado de los caminos recorridos por el coche. Este mapeo permitirá registrar las medidas recogidas durante el trayecto, facilitando la creación de un mapa detallado del entorno y las condiciones ambientales.
- **Desarrollo de una aplicación:** Crear una aplicación móvil o web que permita recoger todos los datos generados por el coche y enviarle órdenes específicas. Esta herramienta proporcionaría una interfaz de usuario amigable para monitorear y controlar el coche de manera más eficiente.
- **Machine learning:** Introducir algoritmos de aprendizaje automático (machine learning) para mejorar la precisión y adaptabilidad del sistema. Estos algoritmos permitirán que el coche aprenda de sus experiencias y se vuelva más inteligente con el tiempo.

6.2. Costes del Trabajo

- **Costes en materiales**

El desarrollo del proyecto incluyó los siguientes costes de materiales:

Nombre componente	Unidades/Pack	Coste unitario (euros)	Total (euros)
Paquete de folios	1 pack	1.00	1.00
Cintas adhesivas negras	4 rollo	1.20	4.80
Cintas adhesivas blancas	2 rollos	1.20	2.40
Cinta adhesiva transparente	1 rollo	1.20	1.20
Freenove 4WD Car Kit	1 unidad	59.95	59.95
MPU6050	1 unidad	3.00	3.00
Cables de Arduino hembra a hembra	Pack múltiple	3.00	3.00
Total			**75.35**

(Tabla 6–1 Tabla de los costes de los materiales del proyecto)

Estos costes fueron fundamentales para la realización de las pruebas experimentales y el desarrollo funcional del coche autónomo, demostrando una inversión eficiente en recursos para alcanzar los objetivos del proyecto.

- **Costes del programador junior**

Además de los costes en materiales, se evaluó el coste de un programador junior que desempeñaría las funciones de programación de Arduino y QA al realizar las pruebas mencionadas.

- **Desarrollo del Código**

1. Comprensión de Requisitos y Preparación (4-6 horas)**:**
    - Revisión de los requisitos del proyecto.
    - Configuración del entorno de desarrollo.
2. Implementación del Código (16-24 horas)**:**
    - Escribir y probar el código principal.
    - Integración de sensores (MPU6050, sensores de línea, ultrasonido, etc.).
    - Configuración de la comunicación Bluetooth.
3. Depuración y Optimización (8-12 horas)**:**
    - Identificación y corrección de errores.
    - Optimización del código para eficiencia y precisión.

- **Pruebas del sistema**

1. Preparación de escenarios de prueba (4-6 horas)**:**
    - Configuración de circuitos de prueba.
    - Asegurar que los escenarios sean adecuados para la validación.
1. Realización de pruebas (24-32 horas)**:**
    - Realización de los seis experimentos detallados.
    - Documentación de resultados y ajustes necesarios entre pruebas.
1. Análisis y documentación de rresultados (8-12 horas)**:**
    - Análisis de datos de prueba.
    - Redacción del informe de resultados y aprendizajes.

- **Estimación total de tiempo**

- Total, de Desarrollo del Código**:** 28-42 horas
- Total, de Pruebas y Validación**:** 36-50 horas
- Tiempo Total Estimado**:** 64-92 horas

- **Costo aproximado**

La tarifa horaria para un programador junior puede variar dependiendo de la ubicación geográfica, la empresa, y la experiencia del programador. Sin embargo, una tarifa común podría ser entre 15 y 30 euros por hora.

- **Cálculo del costo total**

  - Mínimo Tiempo (64 horas) a 15 euros/hora + 75.35**:** 1035.35 euros
  - Máximo Tiempo (92 horas) a 30 euros/hora + 75.35**:** 2835.35euros

6.3. Conclusión final del proyecto

Los resultados experimentales han demostrado que el sistema desarrollado posee capacidades robustas para el seguimiento de línea, la detección de obstáculos y la medición de temperatura en entornos controlados. La implementación efectiva de sensores infrarrojos y ultrasonido ha permitido al coche autónomo navegar con precisión siguiendo una línea negra en diferentes configuraciones de circuito. Además, al añadir la función de la detección de obstáculos y medición de temperatura ha enriquecido significativamente la capacidad del sistema para responder proactivamente a su entorno.

Se observó que la precisión del seguimiento de línea y la detección de obstáculos dependen en gran medida de la uniformidad y las condiciones del entorno. En escenarios con superficies irregulares o baja fricción, se identificaron limitaciones que afectaron temporalmente el rendimiento del sistema. Sin embargo, ajustes como el reposicionamiento de los sensores infrarrojos debajo del chasis y mejoras en el diseño del circuito contribuyeron a mitigar estos desafíos y mejorar la estabilidad operativa del coche autónomo. En conjunto, este proyecto no solo ha logrado sus objetivos iniciales, sino que también ha proporcionado una base sólida para futuras mejoras y aplicaciones prácticas en el campo de la robótica autónoma y la automatización industrial.

La inversión en recursos ha demostrado ser eficiente, permitiendo alcanzar los objetivos del proyecto y presentar un sistema funcional con potencial para adaptarse y evolucionar en respuesta a nuevos desafíos y necesidades. Además, la aplicabilidad práctica de este proyecto resalta su relevancia y potencial impacto en la industria, ofreciendo una solución accesible y efectiva para la automatización de tareas logísticas y de manejo de materiales.

7 REFERENCIAS

[1] I. Temboury, «El uso de robots en las pymes,» Expansión, 2016. [Online]. Available: https://www.expansion.com/pymes/2016/10/16/58010ec2e5fdea14668b45b5.html.

[2] Prometec, «Sensores infrarrojos,» Prometec, [Online]. Available: https://www.prometec.net/infrarrojos/.

[3] ETC2, «Ultrasonic Sensor Module HC-SR04,» Datasheet, [Online]. Available: https://pdf1.alldatasheet.com/datasheet-pdf/download/1132204/ETC2/HCSR04.html.

[4] Maker Guides, «How to use an MPU6050 3-axis accelerometer and 3-axis gyrosensor with Arduino,» Maker Guides, [Online]. Available: https://www.makerguides.com/how-to-use-an-mpu6050-3-axis-accelerometer-and-3-axis-gyrosensor-with-arduino/.

[5] Circuit Basics, «How to Use Active and Passive Buzzers on the Arduino,» Circuit Basics, [Online]. Available: https://www.circuitbasics.com/how-to-use-active-and-passive-buzzers-on-the-arduino/.

[6] Programar Fácil, «PWM con Arduino: Salida analógica,» Programar Fácil, [Online]. Available: https://programarfacil.com/blog/arduino-blog/pwm-con-arduino-analogico/.

[7] Naylamp Mechatronics, «Configuración del módulo Bluetooth HC-05 usando comandos AT,» Naylamp Mechatronics, [Online]. Available: https://naylampmechatronics.com/blog/24_configuracion-del-modulo-bluetooth-hc-05-usando-comandos-at.html.

[8] Freenove, «Freenove 4WD Car Kit User Manual,» Freenove, [Online]. Available: https://store.freenove.com/products/fnk0041.

8 ANEXO

8.1. Código fuente completo del proyecto:

```
#include <Wire.h>
#include <MPU6050.h>
#include <SoftwareSerial.h>

// Define el objeto SoftwareSerial para la comunicación con el módulo Bluetooth
SoftwareSerial BTSerial(0, 1); // RX, TX

#define MOTOR_DIRECTION 1
#define PIN_DIRECTION_LEFT 4
#define PIN_DIRECTION_RIGHT 3
#define PIN_MOTOR_PWM_LEFT 6
#define PIN_MOTOR_PWM_RIGHT 5
#define PIN_SONIC_TRIG 7
#define PIN_SONIC_ECHO 8
#define PIN_BATTERY A0
#define PIN_BUZZER A0
#define PIN_TRACKING_LEFT A1
#define PIN_TRACKING_CENTER A2
#define PIN_TRACKING_RIGHT A3

#define MOTOR_PWM_DEAD 10
#define TK_STOP_SPEED 0
#define TK_FORWARD_SPEED (95 + tk_VoltageCompensationToSpeed)

// Definición de diferentes velocidades
int tk_VoltageCompensationToSpeed; // Definicion de compensacion de velocidad segun voltaje
#define TK_TURN_SPEED_LV4 (195 + tk_VoltageCompensationToSpeed)
#define TK_TURN_SPEED_LV3 (155 + tk_VoltageCompensationToSpeed)
#define TK_TURN_SPEED_LV2 (-130 + tk_VoltageCompensationToSpeed)
#define TK_TURN_SPEED_LV1 (-150 + tk_VoltageCompensationToSpeed)
#define OBSTACLE_DISTANCE 30
#define OBSTACLE_DISTANCE_MID 20
#define OBSTACLE_DISTANCE_LOW 10
#define MAX_DISTANCE 1000
#define SONIC_TIMEOUT (MAX_DISTANCE * 60)
#define SOUND_VELOCITY 340 // velocidad de la luz: 340m/s

// Inicialización del objeto MPU6050
MPU6050 mpu;

float batteryVoltage = 0;
bool isBuzzered = false;
```

```
void setup() {
  // Iniciar la comunicación serial para depuración y Bluetooth
  Serial.begin(9600);
  // Iniciar la comunicación I2C
  Wire.begin();
  // Iniciar el sensor MPU6050
  mpu.initialize();
  // Iniciar la comunicación serial del módulo Bluetooth
  BTSerial.begin(9600);
  // Verificar si el sensor está conectado correctamente
  if (!mpu.testConnection()) {
    Serial.println("MPU6050 no conectado correctamente");
    while (1);
  }
  pinsSetup(); // Set up pins
  getTrackingSensorVal(); // Calculate Voltage speed Compensation
}

void loop() {
  u8 trackingSensorVal = 0;
  trackingSensorVal = getTrackingSensorVal(); // Get sensor value
  int distance = getSonar();
  if (trackingSensorVal == 7) {
    setBuzzer(true);
    motorRun(TK_STOP_SPEED, TK_STOP_SPEED); // Car stop
    // Leer temperatura del MPU6050
    float temperature = getMPU6050Temperature();
    // Imprimir temperatura en el monitor serial
    Serial.print("Temperatura del MPU6050: ");
    Serial.print(temperature);
    Serial.println(" °C");
    // Enviar temperatura por Bluetooth
    BTSerial.print("Temperatura: ");
    BTSerial.print(temperature);
    BTSerial.println(" °C");
    // Verificar si la lectura de temperatura es válida
    if (isnan(temperature) || temperature < -20 || temperature > 100) {
      // Si la temperatura no es válida, detener el coche y activar el buzzer
      motorRun(TK_STOP_SPEED, TK_STOP_SPEED);
      setBuzzer(true);
      Serial.println("Error: Lectura de temperatura no válida");
      while (1); // Quedarse en un bucle infinito
    }
    delay(500);
    setBuzzer(false);
    motorRun(TK_FORWARD_SPEED, TK_FORWARD_SPEED);
  }
  if (distance < OBSTACLE_DISTANCE_LOW) {
    motorRun(TK_STOP_SPEED, TK_STOP_SPEED);
```

```
    setBuzzer(true);
  } else if (distance < OBSTACLE_DISTANCE_MID) {
    setBuzzer(true);
    delay(20);
    setBuzzer(false);
    trackLine();
  } else if (distance < OBSTACLE_DISTANCE && distance > OBSTACLE_DISTANCE_MID) {
    setBuzzer(true);
    delay(100);
    setBuzzer(false);
    trackLine();
  } else {
    setBuzzer(false);
    trackLine();
  }
  delay(10); // Introduce a delay to slow down the loop execution
}

void trackLine() {
  u8 trackingSensorVal = 0;
  trackingSensorVal = getTrackingSensorVal(); // Get sensor value

  switch (trackingSensorVal) {
    case 0:   // 000
      motorRun(TK_FORWARD_SPEED, TK_FORWARD_SPEED); // Car move forward
      break;
    case 7:   // 111
      motorRun(TK_STOP_SPEED, TK_STOP_SPEED); // Car stop
      delay(500);
      motorRun(TK_FORWARD_SPEED, TK_FORWARD_SPEED);
      break;
    case 1:   // 001
      motorRun(TK_TURN_SPEED_LV4, TK_TURN_SPEED_LV1); // Car turn left
      break;
    case 3:   // 011
      motorRun(TK_TURN_SPEED_LV3, TK_TURN_SPEED_LV2); // Car turn right
      break;
    case 2:   // 010
    case 5:   // 101
      motorRun(TK_FORWARD_SPEED, TK_FORWARD_SPEED);  // Car move forward
      break;
    case 6:   // 110
      motorRun(TK_TURN_SPEED_LV2, TK_TURN_SPEED_LV3); // Car turn left
      break;
    case 4:   // 100
      motorRun(TK_TURN_SPEED_LV1, TK_TURN_SPEED_LV4); // Car turn right
      break;
    default:
      break;
```

```
  }
}

// Función para leer la temperatura del MPU6050
float getMPU6050Temperature() {
  int16_t tempRaw = mpu.getTemperature();
  return tempRaw / 340.0 + 36.53; // Fórmula de conversión de temperatura
}

void tk_CalculateVoltageCompensation() {
  getBatteryVoltage();
  float voltageOffset = 7 - batteryVoltage;
  tk_VoltageCompensationToSpeed = 30 * voltageOffset;
}

// Cuando se detecta línea negra en un sensor, el valor del sensor será 1, o sino lo detecta es 0.
u8 getTrackingSensorVal() {
  u8 trackingSensorVal = 0;
  trackingSensorVal = (digitalRead(PIN_TRACKING_LEFT) == 1 ? 1 : 0) << 2 |
(digitalRead(PIN_TRACKING_CENTER) == 1 ? 1 : 0) << 1 | (digitalRead(PIN_TRACKING_RIGHT)
== 1 ? 1 : 0) << 0;
  return trackingSensorVal;
}

void pinsSetup() {
  // Definición de pines del motor
  pinMode(PIN_DIRECTION_LEFT, OUTPUT);
  pinMode(PIN_MOTOR_PWM_LEFT, OUTPUT);
  pinMode(PIN_DIRECTION_RIGHT, OUTPUT);
  pinMode(PIN_MOTOR_PWM_RIGHT, OUTPUT);
  // Definicion de los pines del ultrasonido
  pinMode(PIN_SONIC_TRIG, OUTPUT);
  pinMode(PIN_SONIC_ECHO, INPUT);
  // Deficiones de los pines del sensor de tracking
  pinMode(PIN_TRACKING_LEFT, INPUT);
  pinMode(PIN_TRACKING_RIGHT, INPUT);
  pinMode(PIN_TRACKING_CENTER, INPUT);
  setBuzzer(false);
}

void motorRun(int speedl, int speedr) {
  int dirL = 0, dirR = 0;
  if (speedl > 0) {
    dirL = 0 ^ MOTOR_DIRECTION;
  } else {
    dirL = 1 ^ MOTOR_DIRECTION;
    speedl = -speedl;
  }
```

```
  if (speedr > 0) {
    dirR = 1 ^ MOTOR_DIRECTION;
  } else {
    dirR = 0 ^ MOTOR_DIRECTION;
    speedr = -speedr;
  }
  speedl = constrain(speedl, 0, 255);
  speedr = constrain(speedr, 0, 255);
  digitalWrite(PIN_DIRECTION_LEFT, dirL);
  digitalWrite(PIN_DIRECTION_RIGHT, dirR);
  analogWrite(PIN_MOTOR_PWM_LEFT, speedl);
  analogWrite(PIN_MOTOR_PWM_RIGHT, speedr);
}

bool getBatteryVoltage() {
  if (!isBuzzered) {
    pinMode(PIN_BATTERY, INPUT);
    int batteryADC = analogRead(PIN_BATTERY);
    if (batteryADC < 614) { // 3V/12V ,Voltage read: <2.1V/8.4V
      batteryVoltage = batteryADC / 1023.0 * 5.0 * 4;
      return true;
    }
  }
  return false;
}

void setBuzzer(bool flag) {
  isBuzzered = flag;
  pinMode(PIN_BUZZER, flag);
  digitalWrite(PIN_BUZZER, flag);
}

void alarm(u8 beat, u8 repeat) {
  beat = constrain(beat, 1, 9);
  repeat = constrain(repeat, 1, 255);
  for (int j = 0; j < repeat; j++) {
    for (int i = 0; i < beat; i++) {
      setBuzzer(true);
      delay(100);
      setBuzzer(false);
      delay(100);
    }
    delay(500);
  }
}

void resetCarAction() {
  motorRun(0, 0);
  setBuzzer(false);
```

```
}

float getSonar() {
  unsigned long pingTime;
  float distance;
  digitalWrite(PIN_SONIC_TRIG, HIGH);
  delayMicroseconds(10);
  digitalWrite(PIN_SONIC_TRIG, LOW);
  pingTime = pulseIn(PIN_SONIC_ECHO, HIGH, SONIC_TIMEOUT);
  if (pingTime != 0)
    distance = (float)pingTime * SOUND_VELOCITY / 2 / 10000;
  else
    distance = MAX_DISTANCE;
  return distance;
}
```