Trabajo Fin de Grado
Ingeniería de Tecnologías Industriales

Robot Auxiliar de Guiado basado en TMS320F28335 de Texas Instruments.

Autor: Manuel Serrano Rodríguez
Tutor: Federico Barrero García

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

Sevilla, 2020

Trabajo Fin de Grado
Ingeniería de Tecnologías Industriales

Robot Auxiliar de Guiado basado en TMS320F28335 de Texas Instruments.

Autor:
Manuel Serrano Rodríguez

Tutor:
Federico Barrero García
Profesor titular

Dpto. de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2020

Trabajo Fin de Grado: Robot Auxiliar de Guiado basado en TMS320F28335 de Texas Instruments.

Autor: Manuel Serrano Rodríguez

Tutor: Federico Barrero García

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

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2020

El Secretario del Tribunal

Agradecimientos

Mis agradecimientos van dirigidos hacia todas aquellas personas que me han apoyado y animado a lo largo de la carrera. Quisiera mencionar en especial a mi familia, que desde el principio confió en mí cuando decidí empezar esta etapa y me ha animado en los momentos más difíciles de ella.

Como no mencionar también a mis compañeros y amigos del grado, que entienden mejor que nadie lo que supone la entrega de este trabajo. Hemos vivido y compartido unos años de carrera inolvidables, llenos de risas, buenos momentos y algún que otro bache en el camino.

Por último, agradecer a los docentes de la escuela todo lo que me han enseñado a lo largo de estos años. Muchos me han inspirado para llegar a ser un buen ingeniero y me han inculcado que el trabajo duro y esfuerzo siempre tienen su recompensa.

Manuel Serrano Rodríguez

Sevilla, 2020

Resumen

Desde hace años se considera al robot seguidor de líneas como uno de los más sencillos creados por el hombre. Se dice que es el *"Hola Mundo"* de la robótica. Sin embargo, es debido a esta sencillez que también es uno de los robots más utilizados en industria, investigación e incluso entretenimiento.

En este proyecto, nuestro objetivo es desarrollar un sistema de guiado de personas en entornos hospitalarios mediante un robot seguidor de líneas. Además, contará con distintas funcionalidades adicionales que iremos detallando a lo largo de la memoria.

Para ello, usaremos un procesador de señales digitales (de ahora en adelante, *DSP*) de *Texas Instruments*, concretamente el modelo TMS320F28335. Mediante la adición de diversos sensores y su posterior programación, dotaremos a nuestro robot de un funcionamiento adecuado al objetivo de este proyecto.

Por último, analizaremos los resultados obtenidos y estudiaremos posibles ampliaciones destinadas a mejorar el funcionamiento y rendimiento del robot.

Abstract

For many years now, the line follower robot is considered as one of the simples man-made robots. It is said to be the *"Hello World"* of robotics. However, it is due to this simplicity that it is also one of the most widely used robots in industry, research and even entertainment.

In this project, our objective is to develop a guiding system for people in hospital settings using a line follower robot. In addition, several extra functionalities will be implemented. These will be detailed as we move further into our project.

We will make use of a digital signal proccesor (hereinafter, *DSP*) from *Texas Instruments*, specifically the TMS320F28335 model. By adding different sensors and the subsequent programming we will achieve the desired operation of the robot.

Finally, we will analyze the obtained results and discuss the possible extensions aimed at improving the robot's performance and operation.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

A^*	Conjugado
c.t.p.	En casi todos los puntos
c.q.d.	Como queríamos demostrar
■	Como queríamos demostrar
e.o.c.	En cualquier otro caso
e	número e
IRe	Parte real
IIm	Parte imaginaria
sen	Función seno
tg	Función tangente
arctg	Función arco tangente
sen	Función seno
$\sin^x y$	Función seno de x elevado a y
$\cos^x y$	Función coseno de x elevado a y
Sa	Función sampling
sgn	Función signo
rect	Función rectángulo
Sinc	Función sinc
∂y ∂x	Derivada parcial de y respecto
$x_\circ$	Notación de grado, x grados.
Pr(A)	Probabilidad del suceso A
SNR	Signal-to-noise ratio
MSE	Minimum square error
:	Tal que
<	Menor o igual
>	Mayor o igual
\	Backslash
⇔	Si y sólo si

1 INTRODUCCIÓN

Hoy en día, la robótica está presente en muchos ámbitos de nuestra vida. Es la capacidad de crear, con un poco de inventiva, innumerables diseños que desempeñan diversas funciones. El propósito de estos robots suele ser facilitar trabajos realizados por el ser humano o directamente reemplazarlos en aquellas tareas demasiado tediosas o peligrosas. Entre estos diseños, nos centraremos en el *AGV (Automated Guided Vehicle)* o vehículo autoguiado.

1.1 El Vehículo de Guiado Automático

Un AGV es un robot móvil autónomo que, por lo general, recorre un camino previamente definido. Suelen utilizarse en entornos industriales, como fábricas o almacenes, para transportar materiales pesados. Minimizando la interacción humana con los bienes transportados, los AGV pueden reducir considerablemente el daño de materiales y el número de accidentes laborales. El uso de vehículos autoguiados se popularizó en la segunda mitad del siglo XX.

Figura 1-1. Ejemplo de AGV en la factoría *Renault España,* en Sevilla.

Entre los distintos métodos que usan estos robots para la navegación, destacan tres:

- **Guiado por láser:** Requiere la instalación de espejos reflectantes en las paredes o elementos del entorno por el que se desplazará el vehículo. El AGV porta una unidad láser rotatoria que actúa tanto de transmisor como receptor. Esto permite al sistema de navegación triangular la posición del robot.
- **Guiado óptico:** Puede ser instalado sin la modificación previa del entorno o infraestructura. Se instalan cámaras que grabaran el recorrido del vehículo, teniendo en cuenta los obstáculos en el camino para definir una ruta. El robot genera un mapeado 2D o 3D del entorno en el que está trabajando. Éste es el sistema más flexible y adaptativo.
- **Guiado por seguimiento de bandas:** Este es el método más expandido para usos industriales, pero a la vez es el menos flexible, ya que la ruta del vehículo queda limitada a la banda instalada en el suelo. Dicha banda puede ser de dos tipos: magnéticas (ejemplo de la figura 1-1) o de colores. La banda de colores es inicialmente menos costosa, pero carece de la ventaja de estar incrustada en áreas de alto tráfico donde puede dañarse o ensuciarse.

1.2 Motivaciones

Ahora que hemos definido lo que son los vehículos de guiado automático podemos empezar a plantearnos el objetivo de nuestro proyecto. ¿Es el ámbito industrial el único que puede beneficiarse de estos robots? ¿Cómo podríamos aplicarlos a otros campos?

Imaginemos un entorno en el que es necesario desplazarse según los deseos de un individuo. Por ejemplo, un centro comercial. En caso de que se quisiera un libro, la persona acudiría a la librería; si se quisiera ropa, a una tienda de moda; o quizás necesite un teléfono móvil nuevo, por lo que iría a una tienda de electrónica. ¿Pero qué pasaría si esa persona tuviera complicaciones para valerse por sí misma? O a lo mejor simplemente prefiere valerse de algo que le permita desplazarse directamente hasta su destino, sin tener que preguntar direcciones o dar rodeos.

Un AGV podría cumplir esta necesidad, mediante un registro de cada una de las rutas hasta los posibles destinos para, posteriormente, guiar a personas hasta ellos. Un lugar con una alta afluencia de personas que podría hacer un buen uso de esta funcionalidad es un hospital. La presencia de personas mayores o discapacitadas en entornos hospitalarios no es extraña. En la actualidad, un trabajador podría realizar fácilmente la tarea de guiar a estas personas hasta la consulta correspondiente, ¿pero no estamos avanzando hacia un futuro cada vez más automatizado?

Por lo tanto, nuestro proyecto tratará de dar una nueva utilidad a los AGV, transformándolos en sistemas auxiliares capaces de guiar personas a través de una clínica o entorno hospitalario.

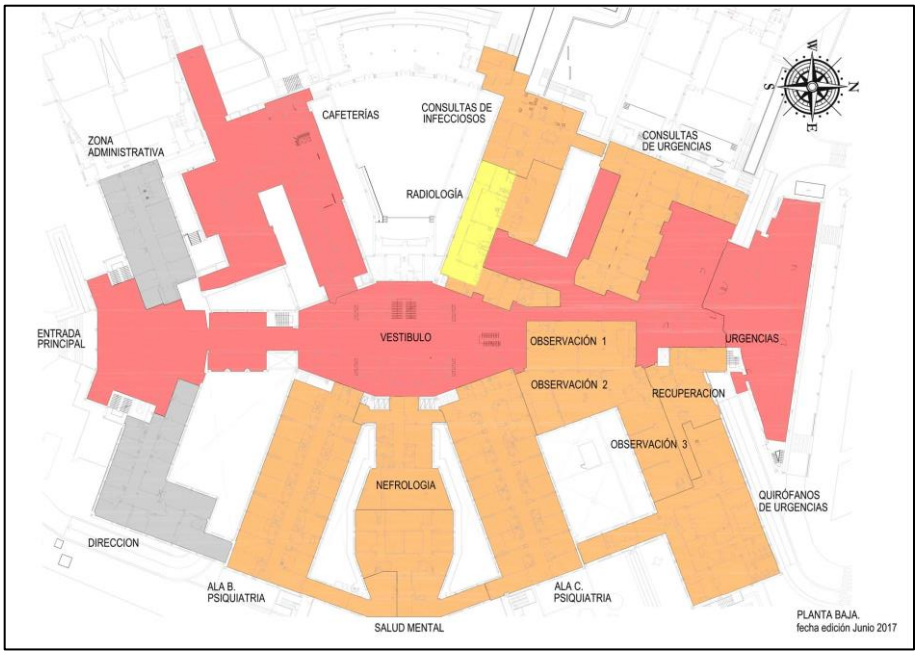

Figura 1-2. Mapa de la planta baja del hospital Virgen Macarena, en Sevilla.

1.3 Planteamiento

El proyecto parte de la idea de un robot seguidor de líneas, al que se le agregarán diversas funcionalidades para que sea capaz de guiar a los pacientes hasta distintas áreas del hospital. El control se basa en el microcontrolador TMS320F28335 de *Texas Instruments*, programado en lenguaje C, mediante el entorno *Code Composer Studio*. Esta elección se debe a que ya hemos manipulado con anterioridad este *DSP* en la asignatura de Electrónica Industrial, por lo que ya contamos con ciertas nociones sobre el funcionamiento de este dispositivo.

Así mismo, el robot seguidor de líneas básico no bastaría por sí mismo, por lo que debemos considerar la adición de diversos sensores que añadan funcionalidades necesarias para el guiado. Además, deberemos diseñar y construir una estructura adecuada para el vehículo, que permita un funcionamiento cómodo y eficiente, y determinar cómo se efectuará la comunicación entre el paciente y el robot.

Por último, crearemos un circuito que nos permita mostrar todas las aptitudes que hayamos implementado en el robot. Ha de tenerse en cuenta, que, debido al alcance de este TFG, nuestro robot será una versión relativamente simple, con una escala reducida y omitiendo muchas pequeñas particularidades que habría que considerar para su contraparte real.

1.4 Objetivos

1. **Definir comportamiento:** Nuestra primera tarea será definir como queremos que actúe nuestro robot y que funcionalidades queremos implementar.
2. **Elección de componentes:** En función de lo anterior, definiremos los componentes que darán forma al robot, incluyendo sensores, actuadores y chasis.
3. **Estudiar el funcionamiento de los sensores:** Recopilar información acerca de los sensores que vamos a usar y determinar cómo queremos que influyan en nuestro robot.
4. **Acoplamiento de componentes:** Se trata de enlazar todos los componentes para que el robot quede "montado". Para ello, usaremos una placa de circuito impreso o *PCB (Press Circuit Board),* que debemos diseñar previamente.
5. **Diseño del circuito:** Deberemos crear un entorno por el que se maneje nuestro robot, mostrando todas las funcionalidades implementadas en el apartado de programación.
6. **Programación:** Es la creación del código que dará vida al robot.
7. **Ajuste del robot:** Realizaremos distintas comprobaciones para verificar el correcto funcionamiento del robot. Serán necesarios pequeños retoques a la programación para que el robot pueda manejarse de forma más fluida por el entorno diseñado.
8. **Análisis de resultados:** Veremos en qué áreas destaca el robot y en cuáles podría mejorar su comportamiento. Esto incluye posibles ampliaciones y mejoras.
9. **Conclusiones.**

2 DISEÑO DEL ROBOT

El robot seguidor de líneas como tal es un robot sencillo, cuyo funcionamiento es generalmente la detección de una línea negra sobre un fondo blanco mediante sensores infrarrojos, formados por un LED infrarrojo y un fototransistor. Sin embargo, mediante la adición de otros componentes, podemos lograr un funcionamiento más complejo y adecuado a nuestro planteamiento.

2.1 Funcionalidades

Lo primero es definir que comportamiento queremos que efectúe nuestro robot.

2.1.1 Tracción

Aunque durante mucho tiempo los humanos hemos tratado de perfeccionar el desplazamiento mediante patas de un robot, emulando nuestro propio movimiento o el de animales, esto supone una gran complejidad de control debido al elevado número de grados de libertad y consumo energético.

Para este proyecto, nuestro objetivo debería ser diseñar un robot con ruedas. No solo es más fácil implementar el mecanismo de una rueda, si no que elimina la necesidad de diseñar un sistema de equilibrio siempre que tres o más estén presentes. Además, la locomoción con ruedas es relativamente eficiente, incluso a altas velocidades.

2.1.1.1 Motores

Como usaremos baterías, los dispositivos que proporcionen la fuerza motriz para desplazar nuestro robot serán por lo general alimentados por corriente continua. Se suelen utilizar tres tipos de motores: motores de corriente continua, motores paso a paso y servomotores.

Nosotros usaremos motores de corriente continua, que suministran velocidades finales muy elevadas al aplicar una diferencia de potencial en sus extremos. Sin embargo, proporcionan una velocidad de giro muy alta para un par de potencia muy bajo, lo que obliga a colocar una reductora que regule dicha velocidad de giro a cambio de una potencia mayor.

Una particularidad de estos motores es que no se pueden clavar y hay que aplicar una inversión de la polarización para frenarlos. Esto se realiza mediante lo que en electrónica conocemos como un Puente H. Este circuito permite a un motor DC girar en ambos sentidos, avance y retroceso, así como la capacidad de frenar.

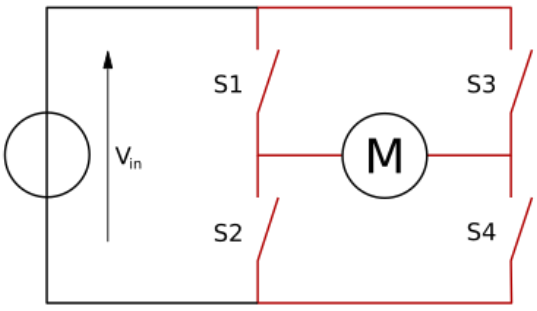

Figura 2-1. Circuito del puente H (representado en rojo).

El circuito consta de 4 interruptores, permitiendo un funcionamiento tal que:

- Cuando S1 y S4 están cerrados y S2 y S3 abiertos, se aplica una tensión positiva al motor, haciéndolo girar en el sentido de avance.
- Cuando S2 y S3 están cerrados y S1 y S4 abiertos, se aplica una tensión negativa al motor, haciéndolo girar en sentido inverso (retroceso).
- Cuando todos los interruptores están abiertos, el motor se detiene bajo su propia inercia.
- Si se cierran S2 y S4 y se abren S1 y S3 (o viceversa), el motor frena bruscamente.

Tabla 2-1. Tabla de verdad de Puente H.

S_1	S_2	S_3	S_4	Resultados
1	0	0	1	Giro en avance
0	1	1	0	Giro en retroceso
0	0	0	0	Punto muerto
1	0	0	0	
0	1	0	0	
0	0	1	0	
0	0	0	1	
0	1	0	1	Frenado
1	0	1	0	
x	x	1	1	Cortocircuito
1	1	x	x	

El puente H vendrá integrado en uno de los componentes que formarán parte del robot: el controlador o *driver* de los motores. Además, dicho componente será también capaz, haciendo uso de las funciones PWM del microcontrolador, de regular la velocidad de giro del motor.

2.1.1.2 Ruedas

A continuación, debemos estudiar la tracción y estabilidad en determinados terrenos, así como control y maniobrabilidad para determinar qué tipo de rueda vamos a usar.

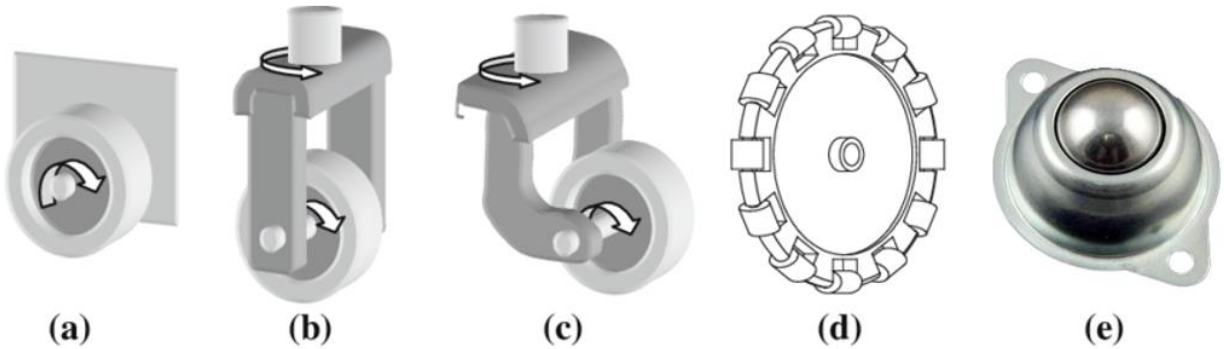

Figura 2-2. Tipos de ruedas para robots móviles.

Según el tipo de rueda, podemos encontrar:

- Rueda fija: Rueda estándar con dos grados de libertad. Puede girar en dos sentidos y generalmente su centro está unido al chasis del robot. El ángulo entre el plano de la rueda y el robot es constante. Figura 2-1-1 a).
- Rueda orientable: Están montadas en una horquilla que va sujeta al robot. Estas ruedas son capaces de rotar en su eje vertical respecto a la estructura del robot, por lo que se usan normalmente para otorgar estabilidad. Distinguimos dos tipos:
    - Orientable centrada: El eje vertical de la rueda pasa también por su centro. Figura 2-1-1 b).
    - Orientable no centrada: También conocida como rueda castor o rueda loca. Su eje vertical no está alineado con el centro de la rueda. Pueden presentar un problema particular denominado *aleteo*, que provoca un movimiento de zigzag o vaivén, observable por ejemplo en los carritos de la compra cuando no tienen peso. Figura 2-1-1 c).
- Rueda omnidireccional: También se conoce como rueda sueca. Está formada por una serie de rodillos en la circunferencia de la rueda que la permiten moverse en cualquier dirección sin mucho esfuerzo. Figura 2-1-1 d).
- Rueda de bola: Contienen una bola esférica, generalmente de metal o nylon, colocada dentro de un soporte. Permiten un movimiento de 360º y suelen usarse para equilibrar al robot. Sin embargo, su alta tracción requiere de una mayor potencia para moverlas. Figura 2-1-1 e).

2.1.1.3 Configuración

Ahora que ya hemos definido los diferentes tipos de ruedas, podemos determinar que configuración queremos que use nuestro robot según la ubicación y número de éstas.

Para un robot seguidor de líneas nos interesa lograr una mayor maniobrabilidad, por lo que hemos decidido que el desplazamiento de se hará mediante dos ruedas fijas situadas en un eje común, controladas individualmente y responsables de la tracción, y una tercera rueda loca que proporcionará estabilidad.

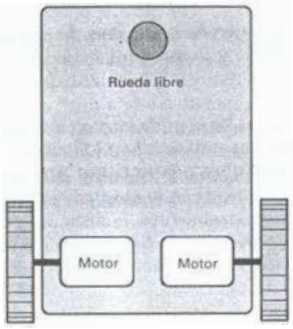

Figura 2-3. Estructura de nuestro robot seguidor de líneas.

El cambio de dirección se consigue mediante la modificación de la velocidad relativa de las ruedas derecha e izquierda. Esta configuración es una variante del modelo de locomoción diferencial de la robótica móvil.

2.1.2 Comunicación

En el contexto en el que será usado este robot, la interacción entre el robot y un usuario se basa en la elección de un destino por parte de este último, para que el primero efectúe una ruta previamente asignada a dicho destino. Teniendo en cuenta que muchos de los pacientes de un hospital son personas mayores, probablemente no se sientan cómodas manipulando una interfaz visual y prefieran comunicarle directamente en voz alta al robot el destino deseado.

En cambio, habrá personas que si hicieran uso de una interfaz visual (digamos, un panel táctil), en el que se muestre una lista con los destinos disponibles y las distintas opciones de control del robot. Es por ello, que proponemos dos métodos de interacción simultáneamente funcionales:

- Control por voz: El robot es capaz de reconocer órdenes habladas simples, como destino (p. ej. *"Quiero ir a dermatología"*), velocidad (p. ej. *"Más despacio."*) o detención (p. ej. *"Para."*).
- Control por interfaz táctil: Una serie de botones táctiles permiten comunicarle al robot el destino y tipo de movimiento deseados.

La comunicación se realizará entre nuestro dispositivo móvil, que hará las funciones tanto de reconocimiento de voz como de interfaz táctil, y el robot mediante conexión *bluetooth*, una tecnología inalámbrica utilizada para intercambiar datos entre dispositivos a corta distancia mediante un enlace por radiofrecuencia en la banda *ISM (Industrial, Scientific and Medical)* de los 2,4 GHz.

A pesar de que muchas personas pueden tener la noción de que es una tecnología anticuada y lenta, el bluetooth cuenta con la enorme ventaja de estar integrado en la mayoría de nuestros dispositivos, lo que lo hace ideal para la comunicación inalámbrica con nuestro robot.

2.1.3 Funcionamiento

2.1.3.1 Seguimiento de línea negra

Aunque se le pueden colocar multitud de sensores, el más clave en el diseño de un robot seguidor de líneas es el sensor de infrarrojos, que es el encargado de detectar la línea a seguir.

Estos sensores están compuestos de dos partes: un LED infrarrojo (emisor) y un fototransistor (receptor) colocados uno al lado del otro. El LED emite una luz infrarroja, invisible para los humanos, que al ser reflejada regresará al fototransistor. Una superficie blanca refleja la luz, mientras que una superficie negra la absorbe, impidiendo que esta llegue al fototransistor.

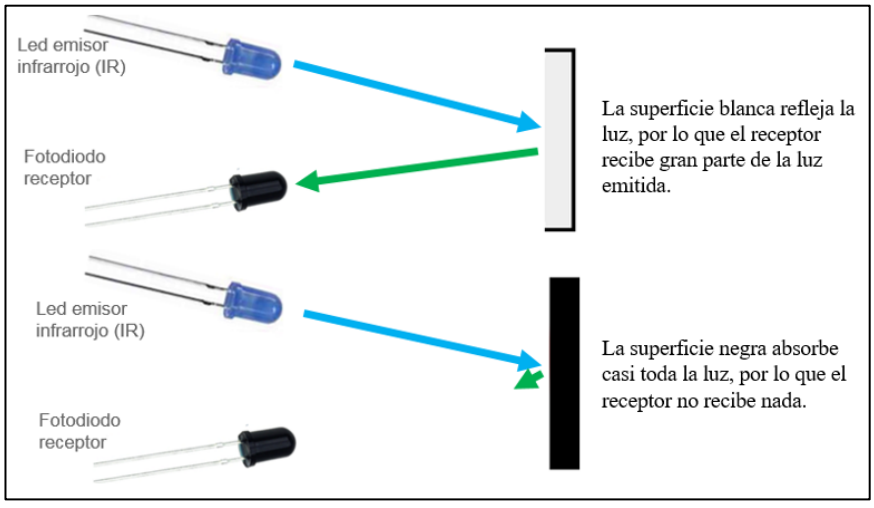

Figura 2-4. Funcionamiento de un sensor de infrarrojos.

La idea pues parte de la capacidad del sensor IR de distinguir una superficie blanca de una negra. Colocando dos sensores, ajustados a la anchura de la línea, podemos detectar cuando el mismo se ha desviado de la línea negra y corregir su movimiento en consecuencia.

Por ejemplo, si se activa el sensor izquierdo, significará que el vehículo se ha desviado hacia la izquierda de la línea, por lo que deberá efectuar un giro hacia la derecha, hasta que el sensor izquierdo vuelva a desactivarse.

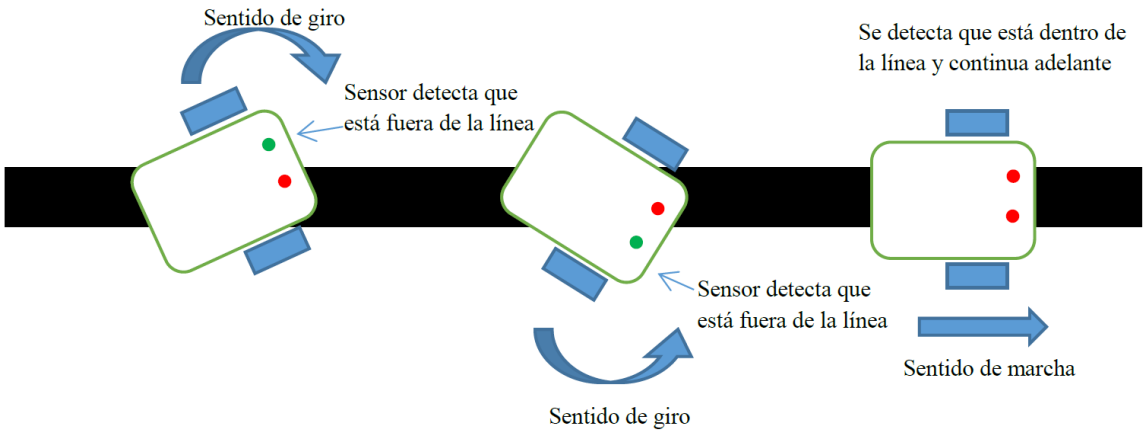

Figura 2-5. Principio de funcionamiento del robot seguidor de líneas.

El giro se efectúa parando la rueda contraria, semejante al movimiento de un tanque. El movimiento de este robot será moderadamente brusco, pero podríamos aumentar su fluidez añadiendo más sensores IR, colocados de forma que la detección de giros y bifurcaciones sea más precisa. Sin embargo, para este proyecto, colocaremos solamente dos sensores, tal y como se ve en la figura 2-1-4.

2.1.3.2 Detección de color

Para que el robot detecte la llegada a su destino o sea capaz de decidir qué camino tomar si se topa con una bifurcación, nos serviremos de un sensor RGB, que realiza una medición digital del color. Usando "estaciones" de distinto color el robot puede discernir en qué parte del circuito se encuentra y que camino debe coger para llegar a su destino.

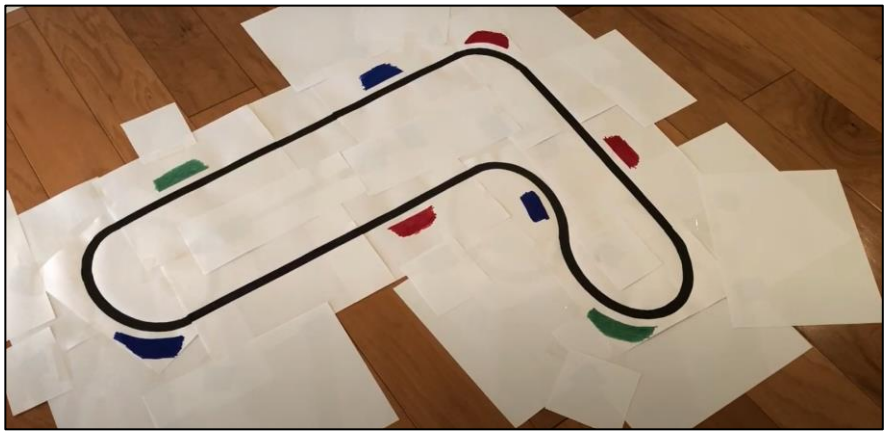

Figura 2-6. Circuito para robot seguidor de líneas con estaciones de color.

La idea es que el circuito sea de la forma de la figura 2-1-5, con franjas de colores que el sensor RGB (colocado en un lateral del robot) pueda detectar. Sin embargo, en nuestro circuito, cada estación será de un color diferente, para evitar confundir a nuestro robot, y el camino no será lineal, si no que contará con bifurcaciones y desvíos.

2.1.3.3 Detección de obstáculos

Finalmente, el robot podrá detectar obstáculos en el camino mediante un sensor de proximidad situado en la parte frontal. Su funcionamiento es similar al del sensor de infrarrojos, pero en este caso, en vez de ser luz, se trata de ultrasonidos.

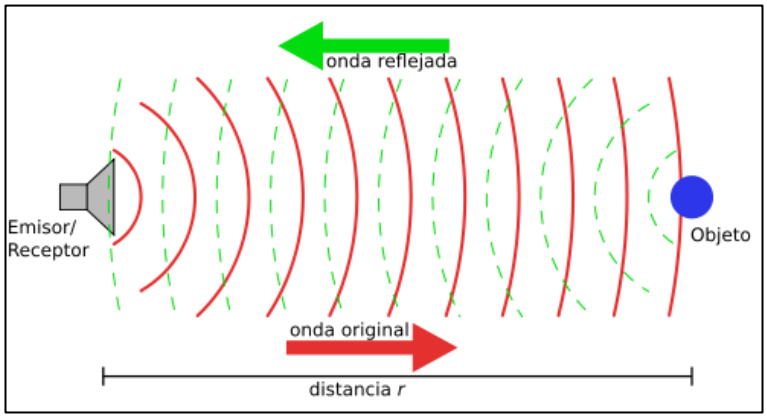

Figura 2-7. Funcionamiento de un sensor de ultrasonidos.

Se basa en el mismo principio que usamos cuando tiramos una piedra a un pozo para medir su profundidad. El sensor emite ultrasonidos y cuando rebotan son captados por el receptor. Tras una serie de cálculos, podemos determinar la distancia recorrida por ese eco.

2.2 Componentes

2.2.1 Microprocesador TMS320F28335 de Texas Instruments.

Texas Instruments (conocida también como *TI*) es una empresa especializada en semiconductores, circuitos integrados y procesadores digitales de señales o *DSP*. Nosotros usaremos el DSP modelo TMS320F28335, un microprocesador optimizado para el procesamiento, detección y actuación en aplicaciones de control exigentes.

Tabla 2-2. Características de TSM320F28335

Características	**TSM320F28335**
Velocidad de procesamiento	150 MHz
Memoria Flash	256K x 16 bits
Memoria RAM (*SARAM*)	34K x 16 bits
Memoria ROM	1K x 16 bits
Canales *PWM*	6
Conversor analógico – digital (12 bits)	16 canales
Temporizadores CPU (32 bits)	3
Comunicación *SCI*	Sí
Comunicación *I2C*	Sí
Interrupciones externas	8
Pines E/S (*GPIO*)	88
Interfaz externa (*XINTF*)	16/32 bits
Llave de seguridad para memoria	128 bits
Puertos periféricos	3

Además de las citadas en la tabla 2-1, este DSP cuenta con muchas otras características adicionales (se pueden consultar en su *datasheet*), pero las hemos omitido al no ser tan relevantes para nuestro proyecto. Es un microcontrolador bastante potente, con unas características superiores a las necesarias para este trabajo. Podríamos incluso desaconsejar su uso, ya que el coste aumentaría innecesariamente, pero es una opción muy cómoda y eficaz para el desarrollo y testeo de nuestro robot.

Se usarán también herramientas de desarrollo en forma de tarjeta de control y estación de acoplamiento para este DSP: modelos TMDSCNCD28335 y TMDSDOCK28335. Proporcionan energía al microprocesador y permiten fácilmente el prototipado mediante una amplia área de pruebas. El acceso a señales clave del dispositivo se puede efectuar a través de una serie de pines disponibles.

Figura 2-8. Estación de acoplamiento (izquierda) y TSM320F28335 aislado (derecha).

El entorno de desarrollo integrado que nos proporciona el fabricante, *Code Composer Studio,* comprende un conjunto de herramientas para la programación y depuración de aplicaciones, incluyendo un compilador C/C++ y una serie de *linkers* necesarios para trabajar con los microcontroladores de Texas Instruments.

2.2.2 Chasis y Motores

Para nuestro proyecto, se usará una estructura especialmente pensada para el montaje de un robot seguidor de líneas, capaz de proporcionar equilibrio y que permita cambiar de dirección rápidamente. Hemos elegido el modelo "Smart Car Robot Kit" de la marca *The Perseids* por su simplicidad mecánica y porque incluye dos motorreductores con sus respectivas ruedas.

Figura 2-9. Smart Car Robot Kit de *The Perseids.*

Ambas ruedas tienen un diámetro de 65 mm y un grosor de 28mm, facilitando una velocidad final y tracción adecuadas. Además, incorpora una tercera rueda loca, que permite al robot girar libremente, como si fuera un tanque, y cambiar rápidamente de dirección.

Tenemos dos motores DC que incluyen una pequeña caja reductora, sacrificando una velocidad de giro más elevada por una potencia mayor.

Figura 2-10. Motores DC con caja reductora.

Por último, se incluye una estación para cuatro pilas AA que actuarán como fuente de alimentación para el controlador de los motores. Cada motor será alimentado a 3V (6V en total).

2.2.3 Controlador de motores

Una parte fundamental de este proyecto es el correcto control de cada uno de los dos motores. Será necesario ajustar sus revoluciones, así como la conmutación de su dirección de giro. Para ello, usaremos el controlador *DRV8835 Dual Motor Driver* de la marca *Pololu*. Aunque su uso está optimizado para procesadores de la serie *Raspberry Pi,* es plenamente funcional y configurable con nuestro microcontrolador.

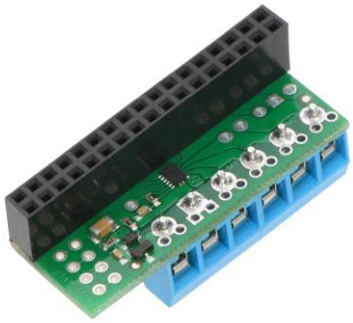

Figura 2-11. Pololu DRV8835 Dual Motor Driver.

El funcionamiento de este controlador se basa en el circuito de Puente H explicado en el apartado 2.1.1. Permite la puesta en marcha, parada, conmutación de la dirección de giro y ajuste de velocidad de dos motores DC simultáneamente.

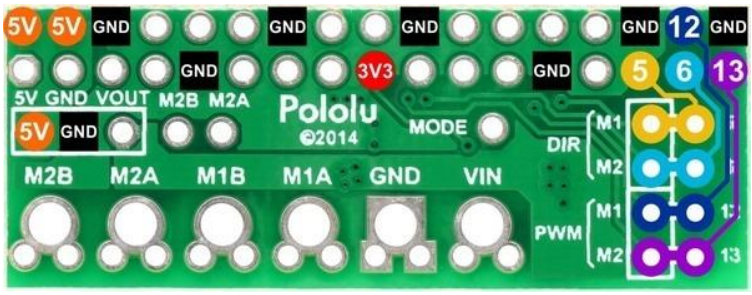

Figura 2-12. Conexionado de Pololu DRV8835 Dual Motor Driver.

Tabla 2-3. Características de *Pololu DRV8835 Dual Motor Driver.*

Características	**Pololu DRV8835**
Control mediante Puente H	2 motores DC
Tensión alimentación motores	1,5V – 11V
Tensión alimentación controlador	2V – 7V
Corriente de salida	1,2A / motor
Corriente de pico	1,5A / motor
Ajuste de PWM	Hasta 250 kHz
2 modos de control	PHASE/ENABLE o IN/IN
Protección contra tensión inversa	Sí
Protección contra sobreintensidades	Sí
Protección contra sobre temperaturas	8
Dimensiones	4,30 × 1,65 cm
Peso	2,3 g

Lo primero será conectarlo con nuestro microcontrolador TMS320F28335 y con ambos motores. Las posibles conexiones se muestran en la figura 2-4.

- Se alimentará a la placa por el pin de 3.3V señalado en rojo.
- Se conecta a tierra por cualquiera de los pines en negro señalizados con "GND".
- Los pines 5 y 6 controlan la dirección de los motores 1 y 2, respectivamente.
- Los pines 12 y 13 controlan la velocidad de los motores 1 y 2, respectivamente.

En base a las señales recibidas por estos pines, se alimentarán los bornes de los motores con los valores de tensión necesarios para que se cumplan las especificaciones deseadas:

- Los pines M2B, M2A, M1B y M1A irán conectados a los bornes de los motores.
- Los pines de VIN y GND irán conectados a la fuente de alimentación externa (4 pilas – 6V).

Tabla 2-4. Conexionado de Pololu DRV8835 Dual Motor Driver.

TMS320F28335 Experimenter's Kit	**DRV8835 Dual Motor Driver**	**Motores**	**Baterías**
3V3	3V3 (rojo)	-	-
GND	GND (negro)	-	-
GPIO2	13 (morado)	-	-
GPIO6	6 (cyan)	-	-
GPIO10	12 (azul)	-	-
GPIO15	5 (amarillo)	-	-
-	M2B y M2A	Derecho	-
-	M1B y M2A	Izquierdo	-
-	GND	-	Negativo
-	VIN	-	Positivo

Por último, como se puede ver en la tabla 2-2, el controlador tiene dos modos. Nosotros usaremos el modo PHASE/ENABLE, cuyo funcionamiento se describe en la tabla 2-3.

Tabla 2-5. Funcionamiento del modo PHASE/ENABLE.

Funcionamiento del modo PHASE/ENABLE				
xPHASE	**xENABLE**	**MxA**	**MxB**	**Operación**
1	PWM	Low	PWM	Retroceso/freno a velocidad *%PWM*
0	PWM	PWM	Low	Avance/freno a velocidad *%PWM*
X	0	Low	Low	Freno

2.2.4 Módulo Bluetooth

La comunicación bluetooth será la encargada de transmitirle la información de control deseada por el usuario al microcontrolador. Para ello, usaremos el módulo bluetooth HC-06.

Figura 2-13. Módulo Bluetooth HC-06.

Generalmente, un dispositivo bluetooth puede actuar como "maestro", iniciando la comunicación con otro dispositivo, o como "esclavo", recibiéndola. En nuestro caso, el módulo actuará como esclavo, recibiendo información de nuestro dispositivo móvil que enviará al microcontrolador a través de los puertos GPIO.

Este módulo está formado por dos placas superpuestas: una es el transceptor, que se alimenta a 3,3V, y la otra es un regulador de tensión, que le permite trabajar en un rango de 3,6V a 6V. Además, incorpora un LED para indicar el estado de la conexión y una pequeña memoria capaz de recordar dispositivos previamente emparejados para así evitar tener que introducir la contraseña en futuras conexiones.

Al contrario que su hermano, el HC-05, este módulo BT solo puede utilizarse en modo esclavo y presenta únicamente 4 pines (frente a los 6 del HC-05), simplificando su uso. Utiliza el protocolo Bluetooth v2.0 + EDR (*Enhanced Data Rate)*, que aumenta la velocidad de transmisión de datos.

Tabla 2-6. Características del HC-06.

Características	**HC-06**
Tensión de alimentación	3,6V – 6V
Consumo de corriente (emparejado)	30mA – 40mA
Pinout	4 pines
Chip de radio	CSR BC417143
Baudios por defecto	9600
Frecuencia	2,4GHz, banda ISM
Potencia de emisión	4 dBm, clase 2
Alcance	5m – 10m
Sensibilidad	-80 dBm
Velocidad asincrónica	2Mbps/160Kbps
Velocidad sincrónica	1Mbps/1Mbps
Seguridad	Autenticación y encriptación
Dimensiones	1,52 × 3,57 cm

Los pines son los siguiente:

- Vcc es el pin de alimentación, en nuestro caso, a 5V.
- GND es el pin de tierra.
- TXD es el pin de transmisión de datos. Todos los datos recibidos por bluetooth en el dispositivo se transmiten por este pin al microcontrolador. Debe ir conectado a un GPIO configurado para recepción de datos.
- RXD es el pin de recepción de datos. A través de este pin se transmiten los datos del microcontrolador al HC-06, que luego serán enviados por bluetooth al dispositivo emparejado. Debe ir conectado a un GPIO configurado para transmisión de datos.

Dado que en nuestro proyecto solo queremos enviar información desde el dispositivo móvil al módulo BT, dejaremos sin conectar el pin de RXD.

Tabla 2-7. Conexionado de los pines del HC-06.

TMS320F28335 Experimenter's Kit	Módulo HC-06
5V	Vcc
GND	GND
GPIO28	TXD
-	RXD

2.2.5 Sensores

Los sensores son aquellos componentes encargados de transmitir al microcontrolador información del exterior que condicionará el movimiento del robot. Nosotros vamos a usar tres tipos de sensores distintos:

- **Sensor de infrarrojos:** Modelo TCRT5000. Es el encargado de detectar la línea negra que debe seguir el robot.
- **Sensor de RGB:** Modelo TCS34725. Capaz de distinguir distintas tonalidades de colores.
- **Sensor de ultrasonidos:** Modelo HC-SR04. Es el encargado de detectar obstáculos que bloquean el camino.

Una particularidad de estos sensores es que están diseñados para su uso con placas de experimentación de la marca *Arduino*. La mayoría de ellos incluyen librerías de programación e instrucciones para conectar los pines con estas placas que hacen de su implementación una tarea prácticamente trivial. Sin embargo, nuestro controlador no puede hacer uso de esas librerías y somos nosotros quienes debemos configurar sus pines manualmente para que puedan comunicarse con estos sensores, que a su vez debemos manipular mediante la información suministrada en sus respectivos *datasheets*.

Esto hace que la correcta implementación de estos sensores con el robot sea mucho más complicada y uno de los grandes retos de este proyecto.

2.2.5.1 Sensor de infrarrojos – TCRT5000

El modelo elegido ha sido el módulo TCRT5000, que incorpora al propio sensor IR una placa lógica que incluye LEDs y pines que permiten el conexionado con un microcontrolador.

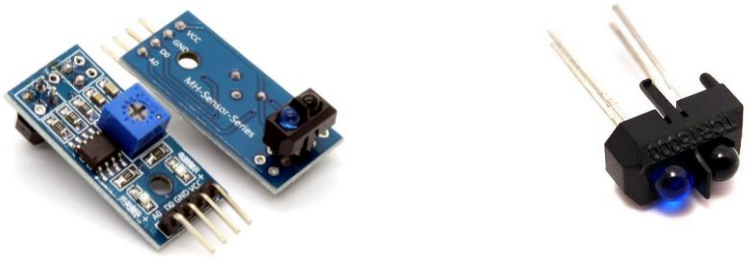

Figura 2-14. Módulo TCRT5000 (izquierda) y sensor aislado (derecha).

Puesto que ya explicamos el funcionamiento de este sensor en el apartado 2.1.3.1, procederemos a listar las características de este módulo.

Tabla 2-8. Características del Módulo TCRT5000

Características	**Módulo TCRT5000**
Sensor óptico reflexivo TCRT5000	
Comparador de tensión amplia LM393	
Potenciómetro digital para ajuste de sensibilidad	
Salida de conmutación digital (0 y 1)	
Salida analógica	
Pinout	4 pines
Indicadores LED	Alimentación y detección
Orificio de perno fijo para facilitar instalación	
Rango de detección	1mm – 8mm
Capacidad de conducción	< 15mA
Tensión de alimentación	3.3V – 5V
Dimensiones	3,2 × 1,4 cm

El módulo cuenta con 4 pines:

- Vcc es el pin de alimentación, que irá conectado a 3,3V.
- GND es el pin de tierra.
- A0 es la salida analógica, que no utilizaremos.
- D0 es la salida digital. Marcará "1" cuando no detecte ningún obstáculo (en nuestro caso, cuando detecte la línea negra) y "0" cuando sí lo detecte (blanco).

Dos de estos sensores serán necesarios en el montaje de nuestro robot.

Tabla 2-9. Conexionado del sensor TCRT5000.

TMS320F28335 Experimenter's Kit	Sensor IR TCRT5000 (1)	Sensor IR TCRT5000 (2)
3V3	Vcc	Vcc
GND	GND	GND
-	A0	A0
GPIO7	D0	-
GPIO11	-	D0

2.2.5.2 Sensor de color RGB – TCS34725

Usaremos el modelo TCS34725 de la marca *Adafruit*. Este es, posiblemente, el mejor sensor de reconocimiento de color para proyectos como el nuestro. Incluye un detector RGB y de luz clara, así como un bloqueador de IR integrado, lo que permite mediciones de color moderadamente precisas. Además, incorpora un LED de luz neutra que facilita la medición en entornos con iluminación deficiente.

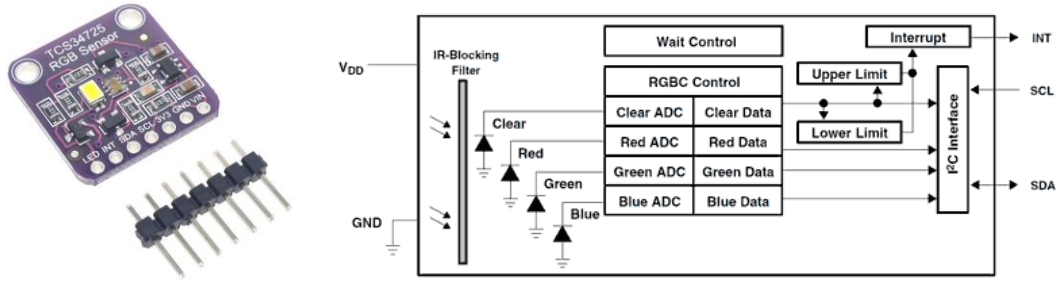

Figura 2-15. Sensor TCS34725 de *Adafruit* y funcionamiento.

Este sensor actúa mediante una matriz 3 × 4 de fotodiodos, con filtros de luz roja, verde y azul y sin filtros (luz clara). 4 conversores analógico-digital integrados convierten las corrientes amplificadas de los fotodiodos a valores digitales de 16 bits. El rango de medición es ajustable en tiempo y ganancia.

La comunicación con el microcontrolador se realiza mediante protocolo I^2C (hasta 400 kHz) y se ofrece la posibilidad de generar una señal de interrupción, mediante umbrales de valores previamente definidos por el usuario.

Tabla 2-10. Características del sensor TCS34725.

Características	**TCS34725**
Tensión de alimentación	3,3V – 5V
Rango dinámico	3.800.000:1
LED neutro	4150 K
Comunicación I^2C	Dirección 0x29
4 canales de medición	RGB y luz clara
Pinout	7 pines
Peso	3,23g
Dimensiones	2,04 × 2,03 cm
Pin de interrupción	Sí
Filtro de luz IR	Sí

La placa cuenta con los siguientes pines:

- VIN es el pin de alimentación a 5V.
- GND es el pin de tierra.
- 3V3 es el pin de alimentación a 3,3V.
- SCL es el pin de *System Clock* para la comunicación I^2C con el microcontrolador. Es la línea de pulsos de reloj que sincronizan el sistema.
- SDA es el pin de *System Data* para la comunicación I^2C con el microcontrolador. Es la línea por la que se mueven los datos entre los dispositivos.
- INT es el pin que genera la interrupción si se sobrepasan los umbrales de valores establecidos.
- LED es el pin que controla el encendido o apagado del LED de luz neutra. El LED está encendido por defecto y debemos conectar este pin a tierra si deseamos apagarlo.

Tabla 2-11. Conexionado del sensor TCS34725.

TMS320F28335 Experimenter's Kit	Sensor de Color TCS34735
5V	VIN
GND	GND
-	3V3
GPIO33	SCL
GPIO32	SDA
-	INT
-	LED

2.2.5.3 Sensor de ultrasonidos – HC-SR04

El modelo a usar será el HC-SR04, un sensor de proximidad basado en el rebote de ultrasonido y que suele utilizarse como transductor para la detección de obstáculos. Este módulo incluye transmisor, receptor y circuito de control.

Figura 2-16. Sensor de ultrasonidos HC-SR04.

Su funcionamiento se basa en una serie de pulsos ultrasónicos generados por el emisor que, al rebotar, llegarán al receptor. Midiendo el tiempo transcurrido entre ambos sucesos y sabiendo que la velocidad del sonido es de 340 m/s se puede determinar la distancia a la que se encuentra el obstáculo, en caso de que lo hubiera.

Tabla 2-12. Características del sensor HC-SR04.

Características	**HC-SR04**
Tensión de alimentación	5V
Corriente trabajando	15mA
Corriente en reposo	< 2mA
Rango de medida	2cm – 400cm
Frecuencia de ultrasonidos	40 kHz
Ángulo efectivo	< 15º
4 canales de medición	RGB y luz clara
Resolución media	3mm
Dimensiones	4,5 × 2 × 1,5 cm
Pinout	4 pines

Los pines son los siguientes:

- Vcc es el pin de alimentación a 5V.
- GND es el pin de tierra.
- TRIG el pin correspondiente al disparador (emite el pulso ultrasónico).
- ECHO es el pin correspondiente al receptor (recibe el pulso ultrasónico).

Tabla 2-13. Conexionado del sensor HC-SR04.

TMS320F28335 Experimenter's Kit	**HC-SR04**
5V	Vcc
GND	GND
GPIO13	TRIG
GPIO14	ECHO

2.3 Placa de Circuito Impreso (PCB)

Para conectar los componentes con nuestro microcontrolador podríamos usar cables puente, diseñados para proyectos electrónicos, y combinarlos con una *protoboard* para mayor estabilidad. Sin embargo, pese a que es una buena opción para el prototipado de los sensores, en el modelo final podría ocasionar problemas de ruido eléctrico, además de pérdidas de potencia.

Por lo tanto, vamos a diseñar una placa que vaya directamente enganchada a la estación de acoplamiento del microcontrolador y que disponga "internamente" de todos los circuitos necesarios para conectar los componentes a través de una serie de pines hembra o macho. Esto es lo que se conoce como placa de circuito impreso o PCB *(Printed Circuit Board)*. Un PCB está formado por placas de sustrato no conductor sobre las que se realiza la interconexión de componentes electrónicos a través de rutas o pistas de un material conductor.

El diseño se hará mediante el software *Altium Designer 15* de la empresa australiana *Altium Unlimited.* Este programa es conocido por su capacidad de lograr buenos resultados para circuitos complejos de forma relativamente rápida. Las principales funcionalidades que usaremos son:

- Módulo de visor de PCBs y ficheros *gerber*, que permiten al ingeniero de diseño visualizar los circuitos impresos dibujados.
- Editor de esquemáticos con integración de librerías de componentes, sean de creación propia o previamente registradas.
- Chequeo de reglas de diseño en tiempo real para asegurar la compatibilidad de los modelos 3D de los componentes y la placa.
- Rutado interactivo basado en el diseño previamente registrado en el esquemático.

2.3.1 Diseño de Elementos

Nuestra primera tarea será crear una librería con los elementos que irán conectados a la placa. Posteriormente, incorporaremos los diseños ya creados de esta librería al de nuestro PCB.

Realmente, por "elementos" nos referimos a los pines necesarios para enganchar el componente real. Por ejemplo, en el caso del módulo bluetooth HC-06, que tiene 4 pines macho, el elemento que irá conectado a la placa será una hilera de 4 pines hembra que permitirán, a su vez, conectar el HC-06. Esto permite mayor flexibilidad para conectar y desconectar los componentes, además de su reutilización para futuros proyectos, que si los conectásemos directamente a la placa.

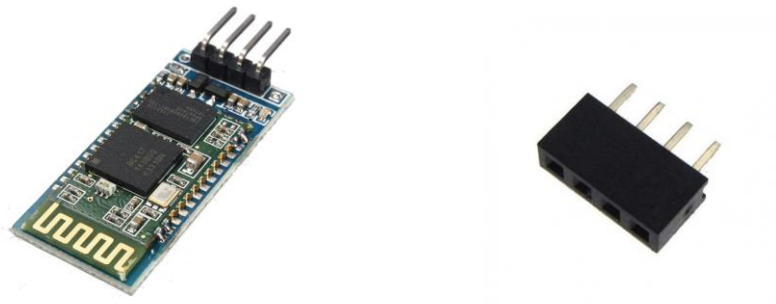

Figura 2-17. Sensor HC-06 con 4 pines macho (izquierda) y cabezal de 4 pines hembra (derecha).

Además, también contaremos como elementos los pines de la estación de acoplamiento que serán conectados a la placa. Al igual que con los sensores, no irán directamente soldados a ella, sino encajados en cabezales hembra que sí estarán soldados al PCB. Debido a la forma del *Experimenter's Kit*, diseñaremos un elemento para los pines del lado izquierdo y otro para los del derecho, ya que debe quedar un hueco en el centro para la *controlCARD*. Aunque la mayoría de los pines a enganchar no se vayan a utilizar, se hará así para otorgar mayor estabilidad al PCB.

Cada elemento estará formado por una o varias hileras de *pads through-hole*, superficies de cobre con un agujero en el centro que nos servirán para soldar los cabezales en los que conectaremos los pines de los componentes.

Dicho pues, nos quedan los siguientes elementos a diseñar:

- **HC-06:** Hilera de 4 *pads*.
- **DRV8835 Dual Motor Driver:** 2 hileras de 17 *pads* cada una.
- **TCRT5000:** Hilera de 4 *pads*.
- **TCS34725:** Hilera de 7 *pads*.
- **HC-SR04:** Hilera de 4 *pads*.
- **GPIOs izquierda:** Son los *pads* para los pines del lado izquierdo de la estación de acoplamiento, incluyendo GNDs y alimentaciones. En rojo en la figura 2-9.
- **GPIOs derecha:** Igual que GPIOs izquierda, pero en el lado derecho de la estación de acoplamiento. En azul en la figura 2-9.

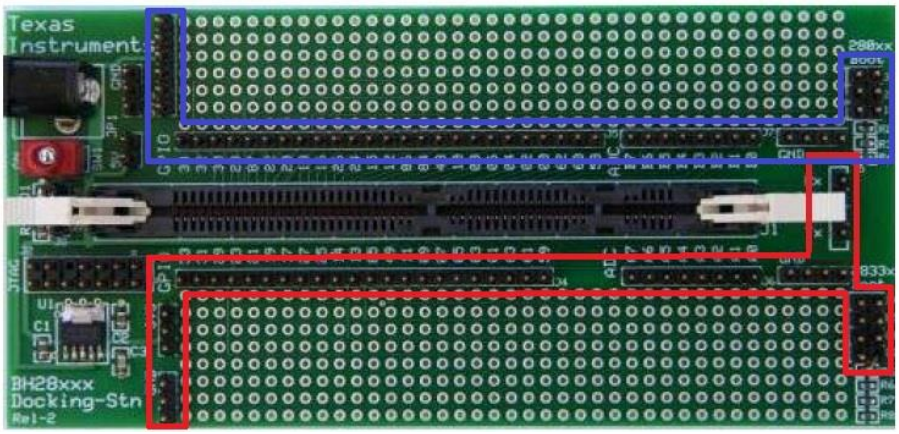

Figura 2-18. Pines del TMDSDOCK28335 que irán conectados al PCB.

Como el proceso es análogo para cada uno de los componentes, aquí mostraremos solo el de uno de ellos, el HC-06.

Lo primero será crear un modelo del componente, lo que se denomina *footprint* o huella. Nosotros usaremos *pads* de 1,524mm de radio con agujeros de 1mm de radio. La distancia entre los centros de los *pads* debe ser 2,54mm, la cual es una separación estandarizada para todos los componentes que vamos a usar. Finalmente, los encerramos en una línea de la capa *top-overlay*, que delimita el tamaño de nuestro elemento.

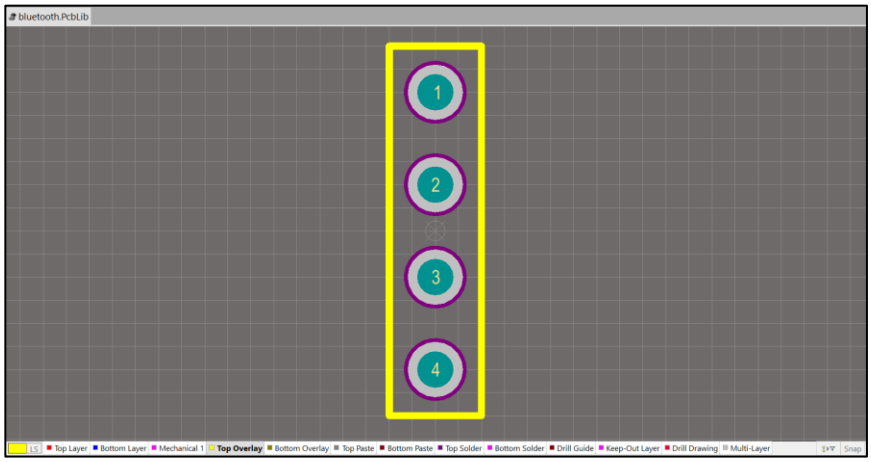

Figura 2-19. Modelo para los *pads* del HC-06.

A continuación, crearemos un esquemático que enlazaremos a la *footprint* ya creada. El esquemático es el encargado de darle nombre tanto al elemento como a cada uno de sus pines, como podemos ver en la figura 2-11. Cada pin se corresponde con uno del HC-06. En la esquina inferior derecha se puede apreciar el modelo creado previamente.

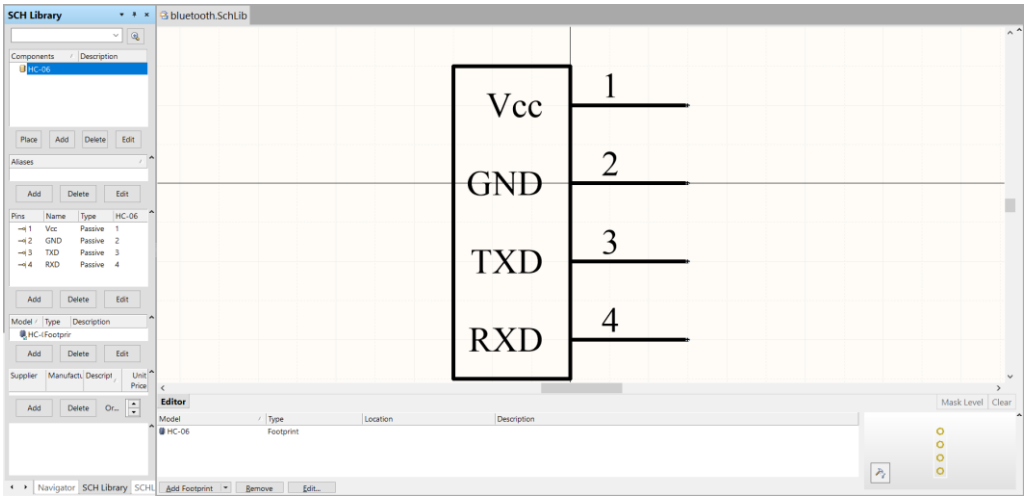

Figura 2-20. Esquemático del componente HC-06.

Finalmente, compilamos para asegurarnos de la ausencia de errores y el elemento pasará a formar parte de la librería que hayamos especificado previamente. Tener una librería con nuestros componentes nos permite incorporarlos fácilmente más tarde al diseño de nuestro PCB.

2.3.2 Diseño de PCB

Ahora que ya tenemos creados los componentes que vamos a utilizar, es hora de empezar a diseñar el PCB. Esta vez, empezaremos por el esquemático, que no es más que una representación visual de las interconexiones de los elementos del circuito. Su objetivo es proporcionar una información clara para que se pueda reproducir y/o modificar el diseño adecuadamente.

Cada elemento o componente debe ir identificado de forma unívoca y sus conexiones se harán mediante *labels*¸ etiquetas que nombran la naturaleza de la conexión. Por ejemplo, los pines de *Vcc* y 3,3V irán conectados por la etiqueta *3V3.* Se deben conectar puertos, alimentación y tierra.

Nótese en la figura 2-11 que muchos pines quedan sin conectar, pues la mayoría solo actuarán de soporte para el PCB sin ninguna funcionalidad real.

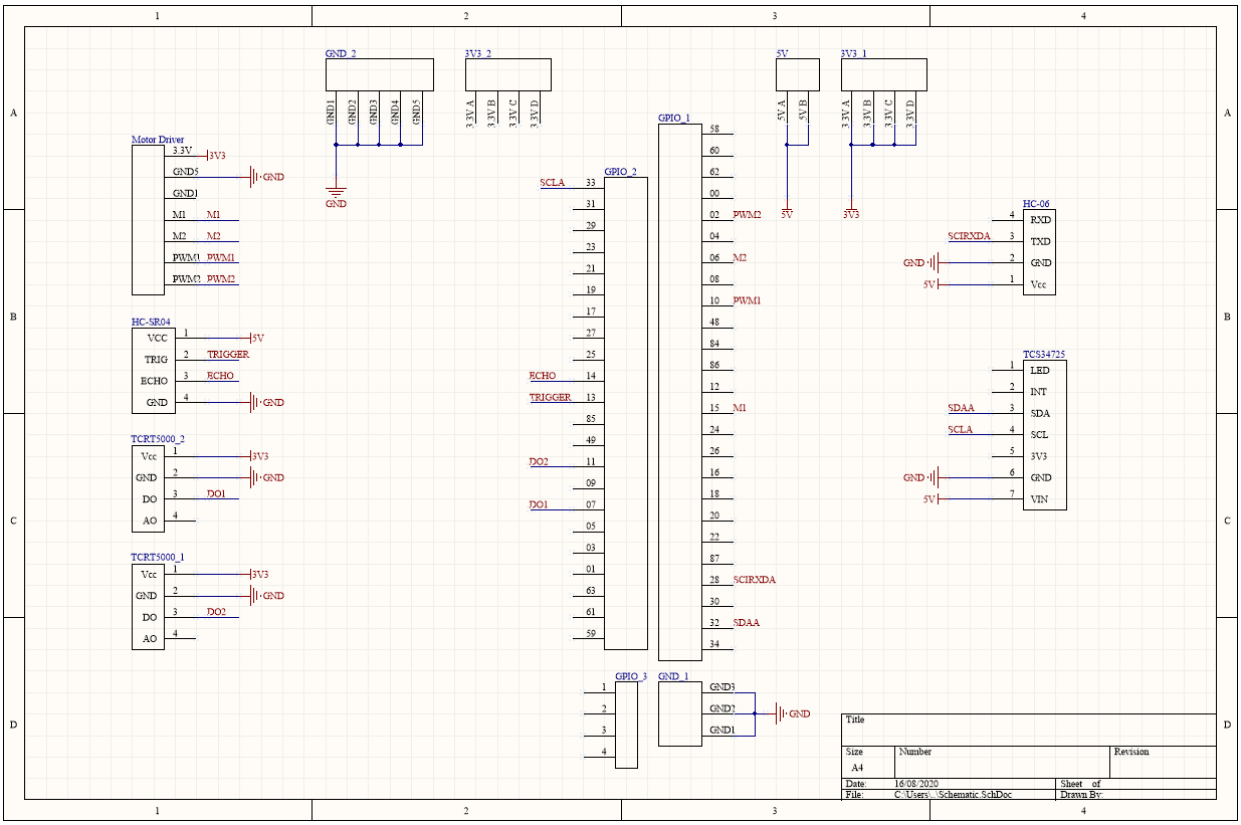

Figura 2-21. Esquemático del PCB.

A continuación, y usando como base el esquemático, creamos el *layout*, el archivo donde se realiza el diseño físico del PCB. El software genera una primera visión de la placa, donde aparecen todos los componentes y las conexiones existentes entre sus pines.

Colocamos los componentes según la disposición que queramos para nuestra placa y trazamos las pistas que la recorrerán. En nuestro caso, crearemos un PCB de doble cara, es decir, tendremos dos *layers* en las que colocar las trazas. El motivo es que las pistas de una misma cara no pueden cruzarse bajo ningún concepto, por lo que la doble cara nos facilita mucho el trabajo a la hora de realizar el rutado. Incluso podemos empezar una pista en una cara y, mediante una *vía*, pasarla a la cara opuesta.

Algunas trazas, como las de alimentación, contarán con un mayor grosor para permitir el paso de la corriente máxima que va a circulas por ellas. Es conveniente también evitar el trazado de ángulos de 90º y mantener una distancia de separación de al menos 0,3mm entre pistas paralelas.

Además, crearemos lo que se denomina un *plano de tierra*, una lámina de cobre conectada a los pines de GND y que sirve como camino de retorno para la corriente de muchos componentes diferentes, eliminando la necesidad de crear pistas de tierra adicionales. Esta capa cubrirá todas las zonas del PCB que no estén ocupadas por trazas y ayudará a reducir el ruido eléctrico, asegurando que la conexión a tierra de todos los componentes esté en el mismo potencial de referencia.

Finalmente, anotamos el nombre de los componentes usando la capa *top-overlay* y delimitamos el tamaño de nuestro PCB mediante la capa *keep-out*¸ dibujando nosotros mismos la forma que queremos que adopte. El software se encargará de comprobar que se cumplen las reglas de diseño y que no hay incoherencias entre el *layout* y el esquemático.

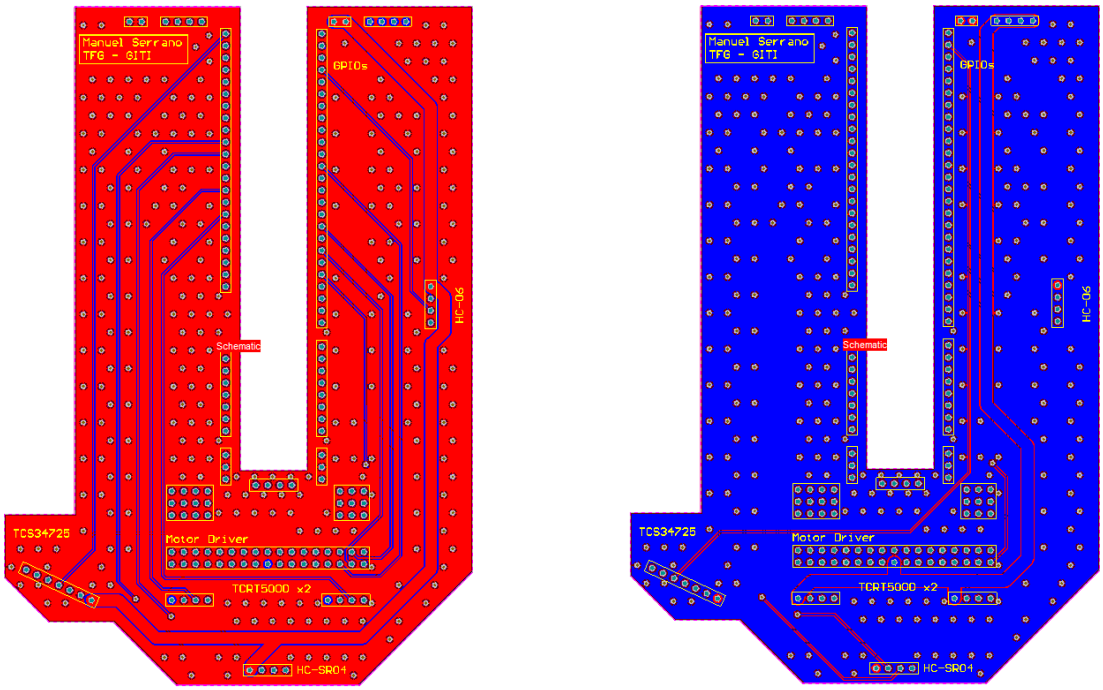

Figura 2-22. Layout del PCB. En rojo la capa superior (*top-layer)* y en azul la capa inferior *(bottom-layer)*.

2.3.3 Fabricación del PCB

Una vez que tenemos nuestro esquemático y *layout* creados y libres de errores, generaremos una serie de ficheros denominados *GERBERS*. Estos archivos contienen información geométrica de las diferentes capas del PCB y son necesarios para el fabricante.

Nosotros hemos recurrido a *JLCPCB*, un fabricante especializado en la producción de PCB en lotes pequeños, de alta calidad y a precios muy económicos. El único inconveniente es que su sede se encuentra en China, por lo que, a pesar de que el tiempo de fabricación no suele superar los 4 días, hay que tener en cuenta el tiempo y los gastos de envío.

Debemos entrar en su página web, *jlcpcb.com*, registrarnos y subir los archivos GERBER. En caso de que estén correctos, la página nos enseñará una vista previa del PCB, con diversas opciones a elegir, como color, número de piezas o el acabado superficial.

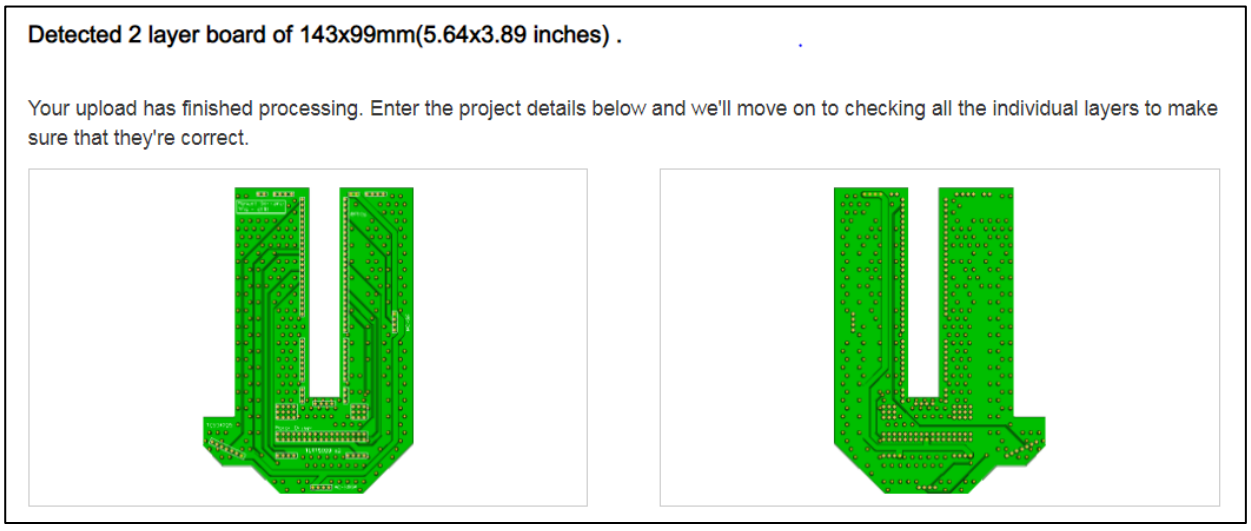

Figura 2-23. Vista previa de nuestro PCB en *jlcpcb.com*.

Ahora que ya tenemos el PCB en nuestras manos comprobaremos que encaja con el *Experimenter's Kit* de nuestro microcontrolador. Podemos ver en la figura 2-14 que el acabado es bastante bueno y su forma le permite acoplarse perfectamente al DSP.

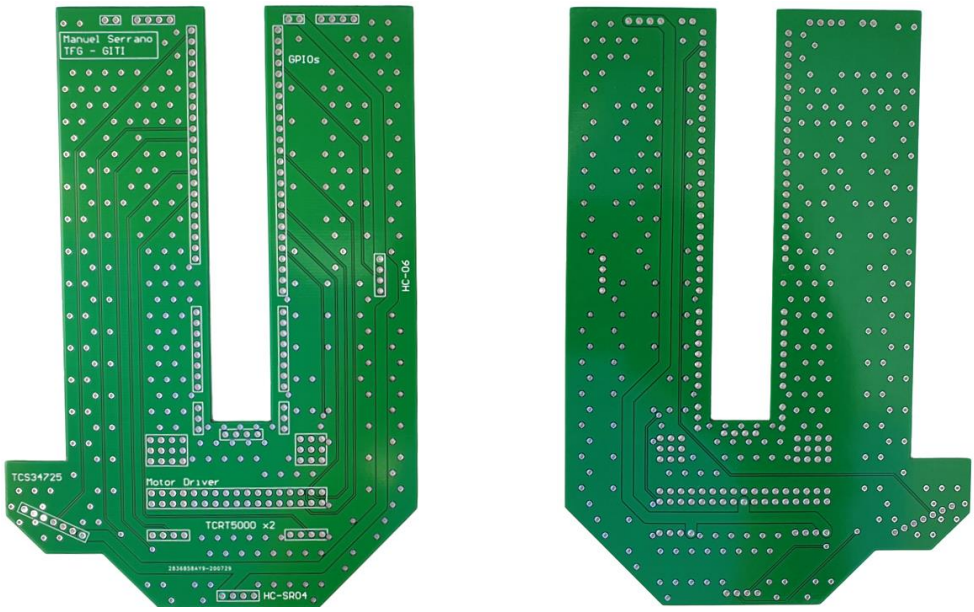

Figura 2-24. PCB finalizado. A la izquierda cara superior y a la derecha cara inferior.

Lo único que queda es soldar los cabezales de los pines que usaremos para conectar nuestros componentes al PCB.

2.4 Montaje y Diseño Final

El diseño final de nuestro robot se puede dividir en 3 niveles:

- En el nivel central encontramos el *Experimenter's Kit* del microcontrolador con su PCB ya integrado. Conectamos al PCB el driver de los motores y, en la parte frontal, el sensor de proximidad. En un lateral situaremos el módulo bluetooth, también conectado al PCB.
- En el nivel superior se sitúa la caja de 4 pilas, elevada mediante unos separadores de latón y cuyos cables irán conectados al driver de los motores del nivel central.
- En el nivel inferior hay dos motores conectados a sus respectivas ruedas y al driver. En la parte frontal encontramos dos sensores IR y el sensor RGB, todos situados casi a ras del suelo (sujetos por separadores de latón) para que puedan desempeñar mejor sus funciones. Los sensores IR irán separados por una distancia de 6mm.

Hemos logrado un diseño compacto, casi libre de cableado y bastante ligero. Además, los sensores se conectan perfectamente al PCB de forma que puedan realizar su trabajo en la mejor condición posible.

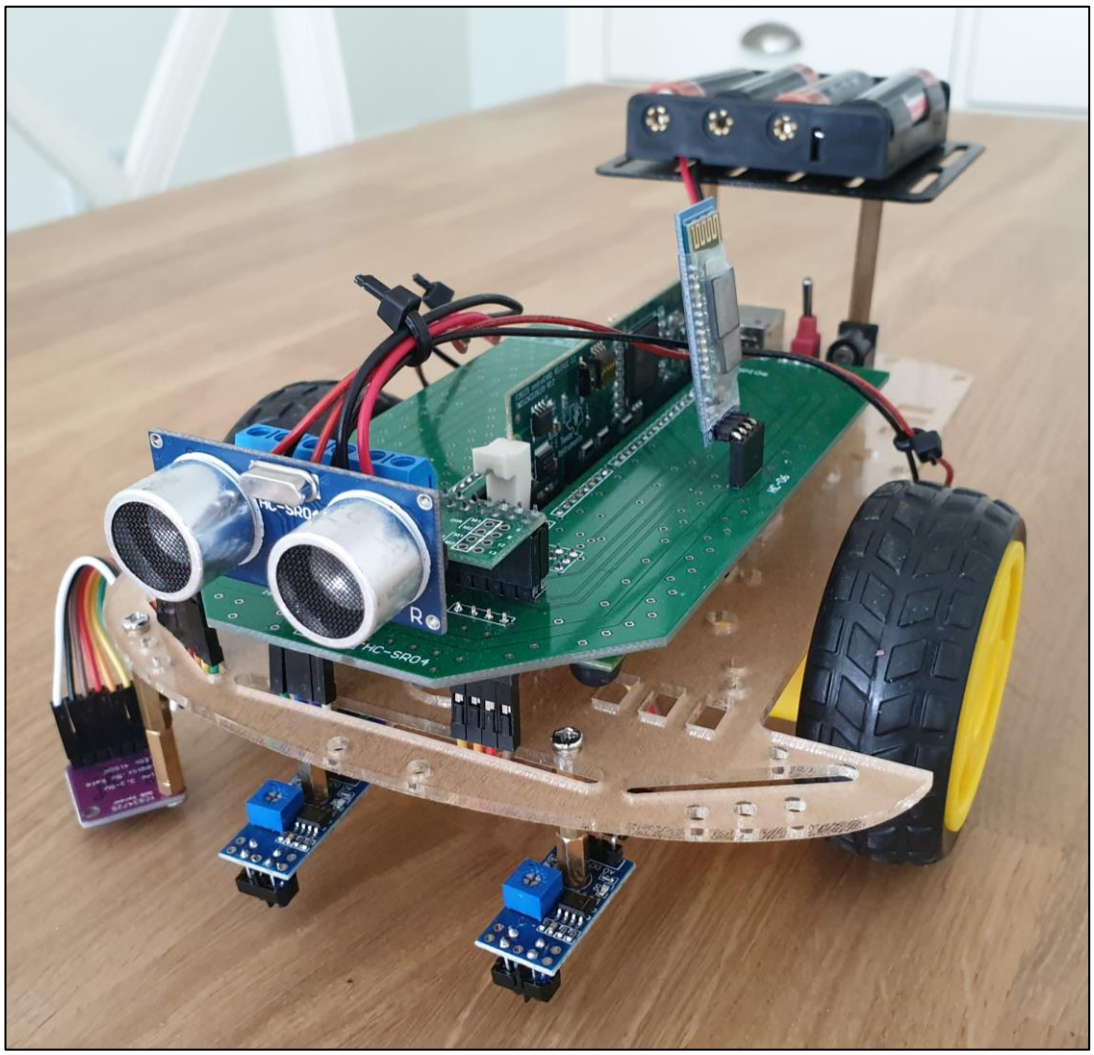

Figura 2-25. Diseño final del robot.

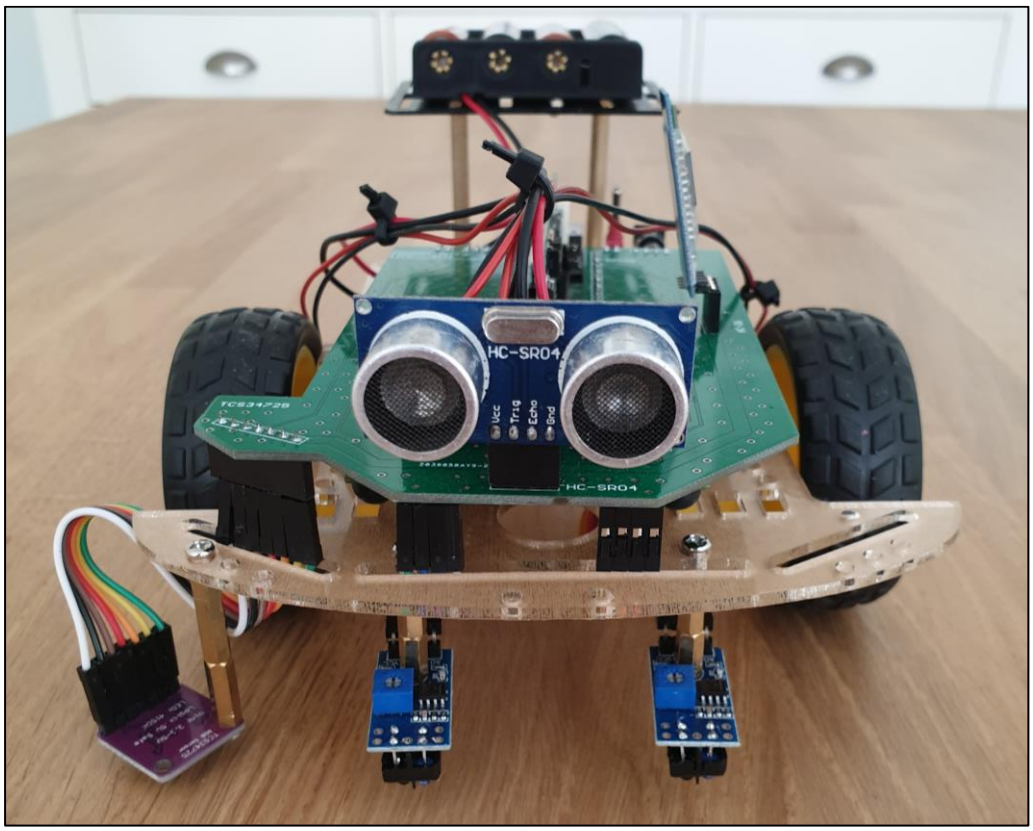

Figura 2-26. Diseño final del robot. Vista frontal.

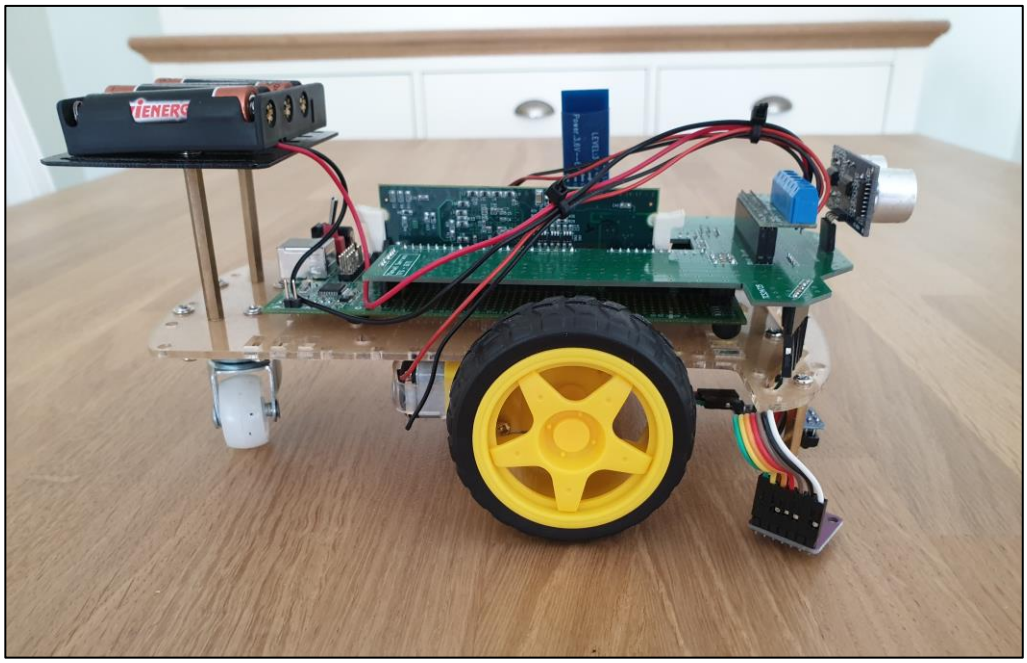

Figura 2-27. Diseño final del robot. Vista lateral derecha.

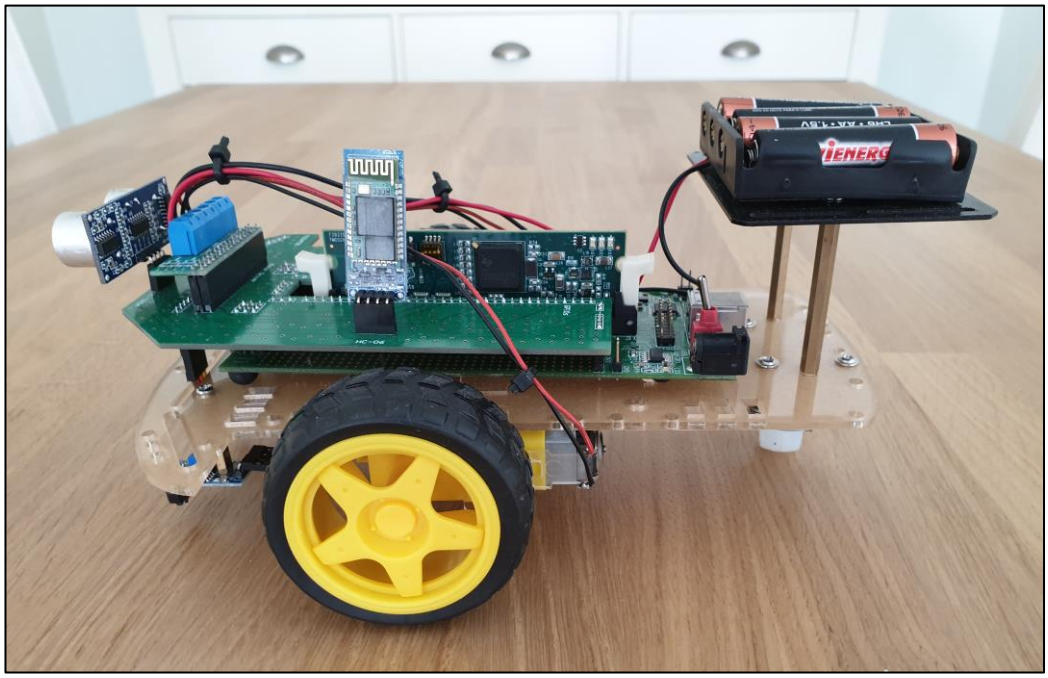

Figura 2-28. Diseño final del robot. Vista lateral izquierda.

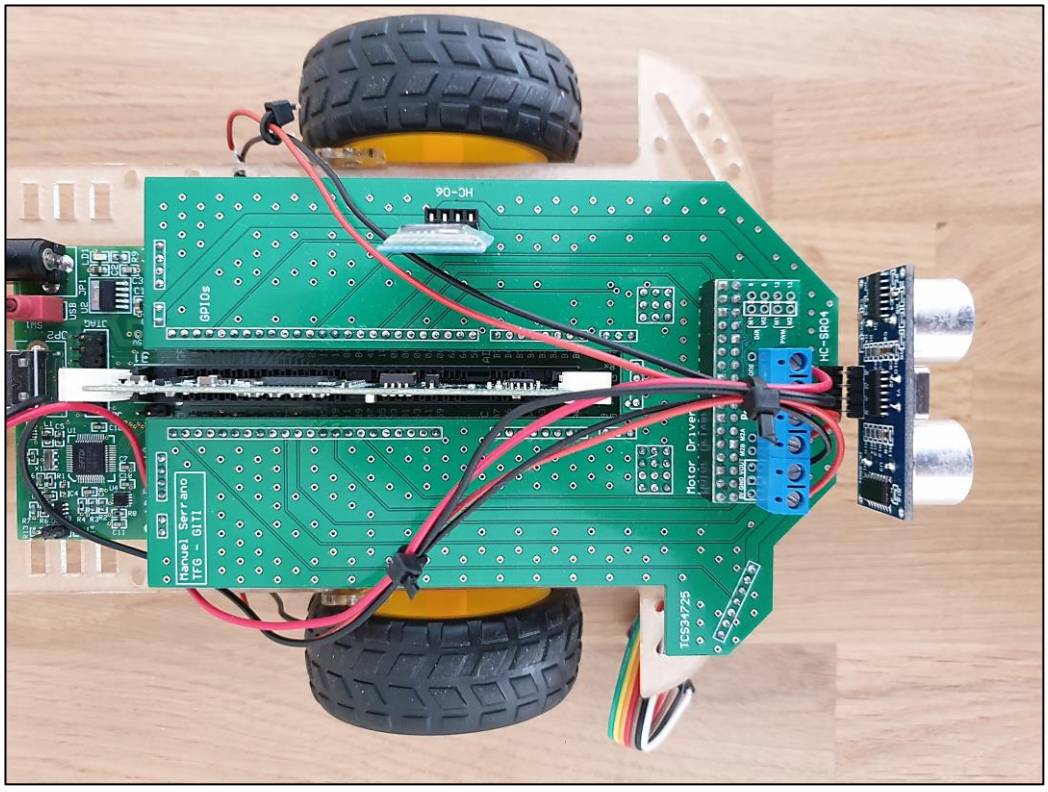

Figura 2-29. Diseño final del robot. Integración del PCB.

Debido a un pequeño error en el diseño del PCB, el driver de los motores está situado al revés de como inicialmente se había pensado, ya que las entradas de cable deberían mirar hacia el lado contrario para facilitar la conexión. No obstante, el diseño sigue siendo plenamente funcional.

2.5 Circuito

Nuestra siguiente tarea será crear un circuito que saque a relucir las funcionalidades del robot. Como ya establecimos en el apartado 2.1.3, el circuito consistirá en líneas negras sobre un fondo blanco. Además, en algunas zonas y bifurcaciones situaremos una pequeña área de color a la derecha de la línea negra (debido a la posición de sensor RGB en el robot), a las que llamaremos "estaciones". Cada estación se corresponderá con un área médica.

Aunque en este apartado establecemos directamente las propiedades del circuito, las elecciones como colores, grosor de línea o bifurcaciones han pasado un extenso proceso de prueba y error que explicaremos en el apartado 3: Programación y Pruebas.

Primero se ha realizado un esbozo en el software *AutoCAD*. El diseño consiste en carreteras de una dirección con diversas curvas y bifurcaciones para comprobar la maniobrabilidad del robot. Cada estación está comunicada con todas las demás (aunque el camino puede que pase por otras estaciones) y también sirven como punto de decisión para girar en algunas bifurcaciones.

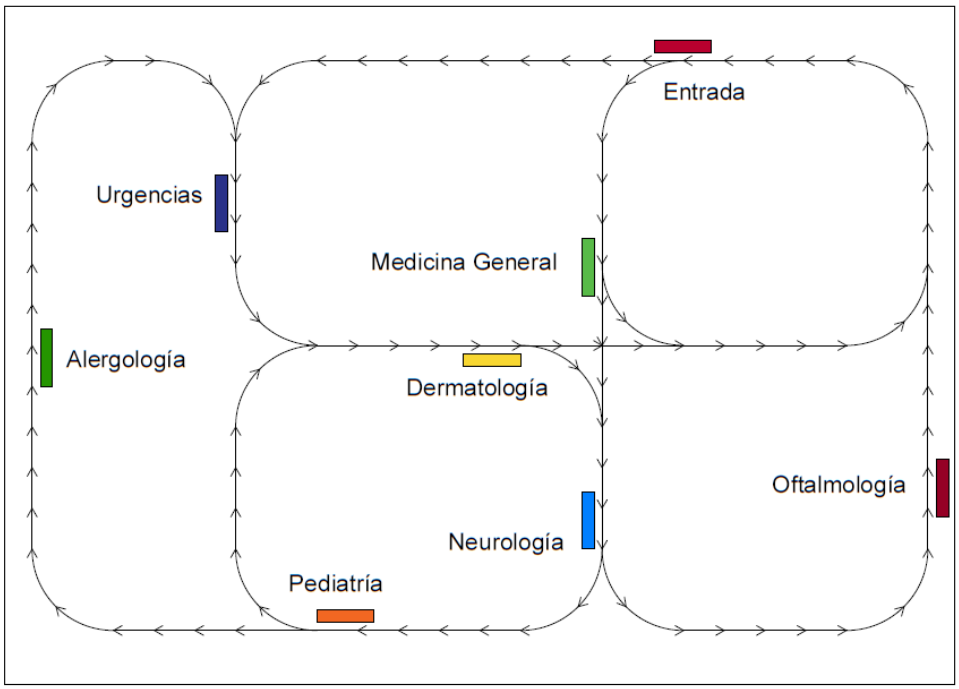

Figura 2-30. Esbozo del circuito.

Para facilitar la posterior programación, cada estación está designada también con un número y quedan tal que:

Tabla 2-14. Estaciones del circuito.

Estación	Color	Área
1	Fucsia	Entrada
2	Lima	Medicina General
3	Marrón	Oftalmología
4	Cyan	Neurología
5	Amarillo	Dermatología
6	Naranja	Pediatría
7	Verde	Alergología
8	Azul	Urgencias

Para el circuito real, se han utilizado los siguientes materiales:

- El fondo consiste en un mantel de plástico blanco de medidas 100×140cm. Se ha elegido este material por ser más resistente y poderse guardar sin deteriorarse. Además, algunos tipos de tinta y pintura se pueden borrar si aplicamos alcohol, lo que nos será útil para procesos de prueba y error.
- Para las líneas negras se ha utilizado pintura de esmalte antioxidante. Su grosor es de 7mm (1mm mayor que la separación entre los sensores IR). Inicialmente se intentó utilizar un marcador permanente negro, sin embargo, con el paso repetido del robot, la tinta tendía a desprenderse y manchar otras partes del circuito. Los sensores IR funcionan también mucho mejor con la pintura, ya que con el marcador fallaban muy a menudo, activándose aún dentro de la línea.
- Las estaciones son trozos recortados de folios de papel de color. En tramos rectos se sitúan tiras de unos 5mm de anchura a la derecha de la línea. En curvas o bifurcaciones se intenta ajustar la estación a la forma de la curva.

En figura 2-31 se puede apreciar el tamaño del circuito respecto al robot.

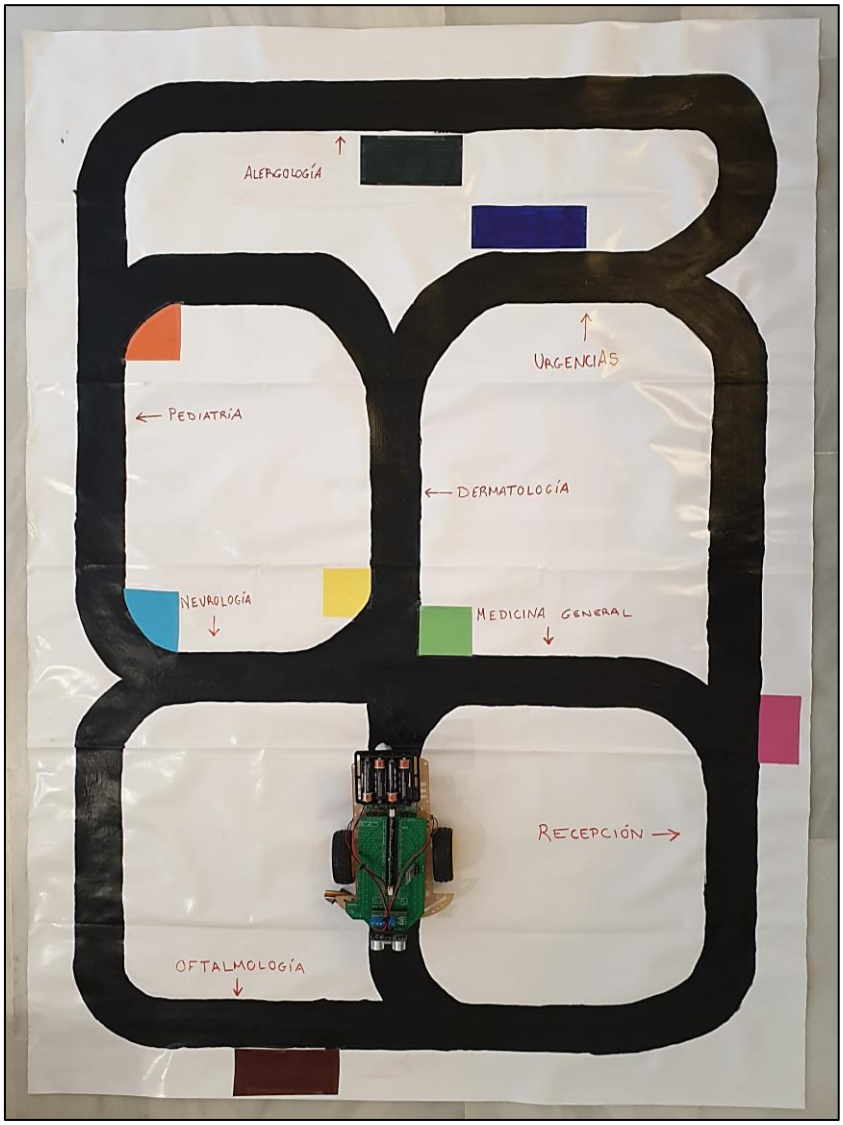

Figura 2-31. Circuito terminado.

3 CONFIGURACIÓN Y PRUEBAS

Para conseguir el funcionamiento deseado de nuestro robot, primero tenemos que configurar y testear todos los módulos que lo componen. Para ello, debemos probar cada componente por separado para luego integrarlos todos a la vez.

Ésta es una de las tareas más complejas de este proyecto, ya que la mayoría de los componentes están diseñados para microcontroladores *Arduino*. En internet se pueden encontrar fácilmente librerías y fragmentos de código, así como infinidad de tutoriales que hacen de la conexión con *Arduino* una tarea trivial.

Sin embargo, la información respecto a la integración de estos componentes con sistemas de *Texas Instruments* es increíblemente escasa, por lo que tendremos que valernos de los *datasheets* y algunos ejemplos que podemos encontrar en su página web.

El entorno que usaremos para programar será *Code Composer Studio*, un software que ofrece la propia empresa y que incluye multitud de librerías que facilitan la programación en la familia de microcontroladores de *Texas Instruments.* El lenguaje de programación a usar será C.

Nuestros componentes a configurar son:

- El controlador de los motores *DRV8835 Dual Motor Driver,* de forma que pueda controlar ambos motores a la vez. Haciendo también uso del módulo PWM (*Pulse Width Modulation*) que viene incorporado en el microcontrolador, debemos controlar la velocidad de giro de los motores.
- El sensor RGB *TCS34725,* para que sea capaz de detectar, diferenciar y registrar colores de forma más o menos continuada. El sensor utiliza protocolo de comunicación I^2C, por lo que será necesario configurarlo también antes de realizar la conexión.
- Ambos sensores de infrarrojos *TCRT5000,* para que manden una señal de interrupción cada vez que se detecte el fondo blanco en vez de la línea negra.
- El sensor de proximidad *HC-SR04*¸ que debe detectar obstáculos en el camino a menos de 10cm de distancia. El funcionamiento de este sensor es un poco más complejo que el de los otros, ya que necesita un proceso diferente y cálculos para obtener la medición.
- El módulo bluetooth *HC-06*, encargado de la comunicación entre el dispositivo móvil y el robot. El proceso de transmisión de datos es tedioso y debe pensarse muy bien para que no interfiera con el resto de las funcionalidades del robot.

Todo el código del proyecto puede encontrarse en los anexos con sus debidas anotaciones, pero en este capítulo se intentarán detallar los aspectos más importantes.

3.1 TMS320F28335

El microcontrolador es, por así decirlo, el corazón del robot. Es el núcleo que recoge, transforma y envía todas las señales necesarias para el correcto funcionamiento de este. Cuenta con diversos periféricos que se utilizan para la comunicación con otros módulos y que necesitaremos para conectar el resto de los componentes.

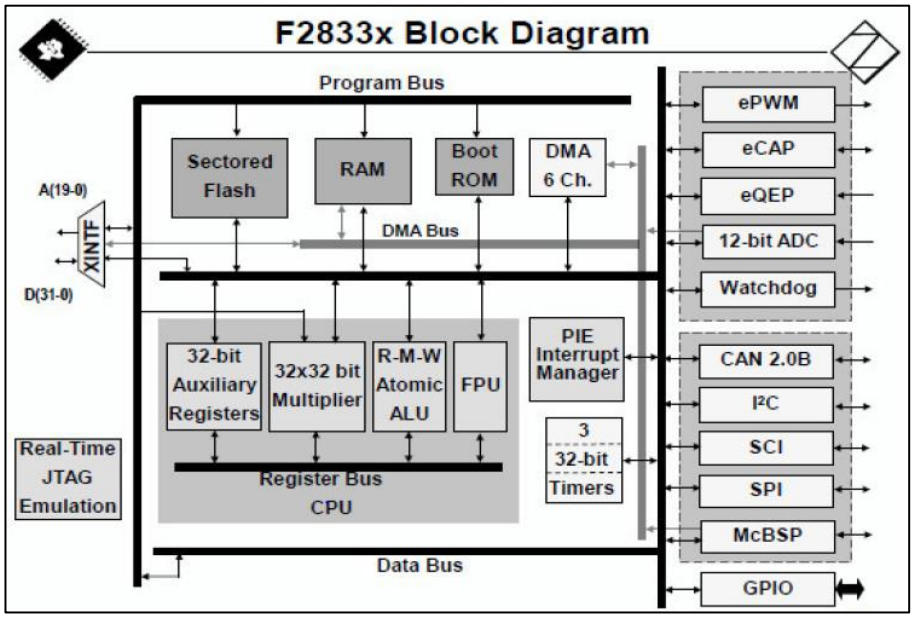

Figura 3-1. Diagrama de bloques de la familia de microcontroladores F28335 de *Texas Instruments.*

3.1.1 GPIOs

El módulo GPIO (*General Purpose Input-Output*) es el encargado de controlar la funcionalidad de los pines que conectan el microcontrolador con dispositivos externos. Estos pines no tienen ningún propósito definido y están inutilizados por defecto. El TMS320F28335 presenta las siguientes características de GPIO:

- Dispone de 88 pines físicos de E/S (*GPIO0 – GPIO87*), agrupados en tres puertos de bits diferentes (*GPIOA, GPIOB, GPIOC*).
- Cada pin de GPIO está multiplexado a 4 funcionalidades, siendo E/S digital la operación por defecto.
- Las otras 3 pueden activarse mediante el acceso al registro de control de GPIOs (*GPxMUXn*).
- Cada puerto dispone de un conjunto de registros para su control individual.

Nosotros haremos uso de 11 GPIOs. Algunos de ellos como E/S digital y otros que controlarán distintos periféricos que detallaremos a continuación.

Tabla 3-1. GPIOs y configuraciones.

GPIO	Puerto	Función
2	GPAMUX1	ePWM2
6	GPAMUX1	Salida Digital
7	GPAMUX1	Entrada Digital
10	GPAMUX1	ePWM6
11	GPAMUX1	Entrada Digital

13	GPAMUX1	Salida Digital
14	GPAMUX1	Entrada Digital
15	GPAMUX1	Salida Digital
28	GPAMUX2	SCIRXDA - SCI
32	GPBMUX1	SDAA – I^2C
33	GPBMUX1	SCLA – I^2C

La configuración de los GPIOs se lleva a cabo en la función **`Gpio_select()`.**

3.1.2 ePWM

El ePWM o modulador por ancho de pulsos es un periférico que se encarga de generar señales cuadradas para modificar los ciclos de trabajo de una señal periódica. En nuestro caso, será necesario su uso para controlar la velocidad de los motores, siendo esta gobernada por su propio bloque lógico.

El TSM320F28335 dispone de 6 generadores ePWM, los cuales generan a su vez dos señales de salida, ePWMxA y ePWMxB, que se relacionan entre sí. Estas señales se transmiten por los pines GPIO que deben ser configurados para ello.

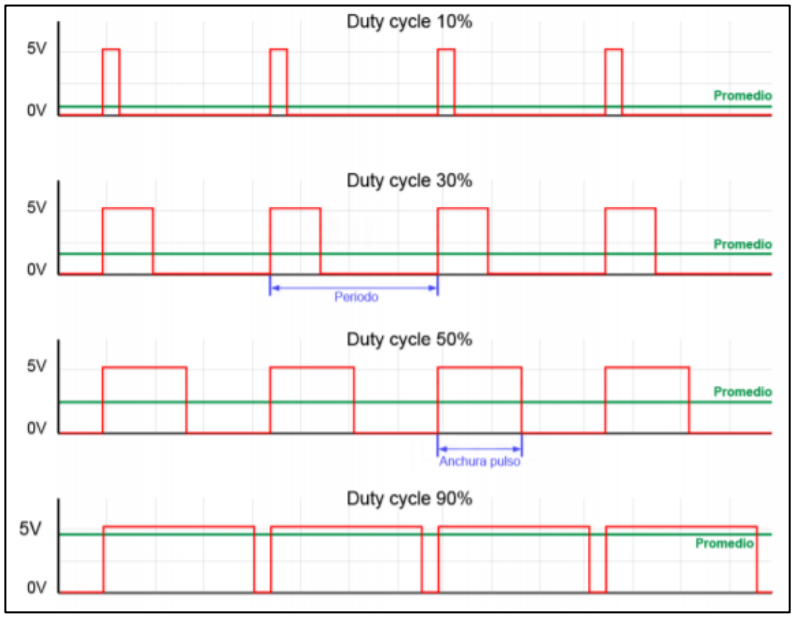

Figura 3-2. Ejemplo de funcionamiento de señales PWM.

El periférico cuenta con los siguientes módulos:

- **Time-Base:** Controla la temporización de la unidad ePWM mediante un timer de 16 bits, generando una señal de tiempo base. Su frecuencia es definida por el usuario y está relacionada con la frecuencia de las dos señales generadas.
- **Counter-Compare:** Controla la fase activa de cada patrón de pulsos y la posición de los puntos de transición de cada ePWM, es decir, controla los registros comparadores (CMPA y CMPB) que determinarán el duty cycle y, por tanto, el valor medio de la señal de cada ePWM.
- **Action-Qualifier:** Se encarga de decidir la acción a realizar en los puntos de comparación establecidos por el módulo counter-compare.
- **DeadBand:** Introduce un tiempo muerto o delay en el flanco de subida y/o bajada de ambas señales.

- **PWM-Chopper:** Permite modular la forma de onda del PWM generado mediante una señal portadora de alta frecuencia.
- **Trip Zone:** Modifica el valor de las señales de salida ePWMxA y ePWMxB en función de las de entrada TZy. Se usa como medida de seguridad ante fallos de sobrecorriente o cortocircuito.
- **Event Trigger:** Determina cuando se produce un evento de comparación y genera una señal de interrupción en determinados casos.

La configuración del módulo ePWM se lleva a cabo en la función **`Setup_ePWM()`.**

3.1.3 SCI

El SCI (*Serial Communication Interface*) es un puerto serie (la información se transmite bit a bit) de comunicación bidireccional para la comunicación asíncrona entre la CPU del microcontrolador y otros dispositivos.

Está formado por 2 líneas de comunicación: SCITXD (transmisión) y SCIRXD (recepción), multiplexadas en los pines GPIO 29 y 35 (SCITXD), y 28 y 36 (SCIRXD). La información que se transmite se denomina marco, que está formado por:

- **Palabra:** Cadena de 1 a 9 bits que contiene la información que se quiere recibir/transmitir.
- **Bit de Start:** Marca el inicio de la palabra. Es opcional.
- **Bit de Paridad:** Indica la paridad de la palabra. Es opcional.
- **1 o 2 bits de Stop:** Marca el final de la palabra.

Además, cada unidad SCI puede generar 2 señales de interrupción: de transmisión o recepción. La configuración del SCI se lleva a cabo en la función **`SCIA_Init()`.**

3.1.4 I^2C

El I^2C (*Inter-Integrated Circuit*) es un protocolo de comunicación que permite conectar varios chips al mismo bus de datos y que todos actúen como maestro. La metodología de comunicación de datos del bus I^2C es en serie y sincrónica. Una de las señales del bus marca el tiempo (pulsos de reloj) y la otra se utiliza para intercambiar datos:

- **SCL** (*System Clock*): Línea de los pulsos de reloj que sincronizan el sistema.
- **SDA** (*System Data*): Línea por la que se mueven los datos entre los dispositivos.

Cada dispositivo conectado al módulo I^2C, incluyendo al microcontrolador, es reconocido por una dirección única y puede funcionar como transmisor o receptor. El periférico I^2C del TMS320F28335 tiene las siguientes características:

- Soporte para transmisiones de formato de 8 bits.
- Modos de direccionamiento de 7 y 10 bits.
- Byte de START.
- Ratio de transmisión de 10 hasta 400 kbps.
- Capacidad de generar una señal de interrupción según determinados eventos.
- Registros de datos que guardan temporalmente la información que se transmite del pin SDA a la CPU y viceversa.
- Preescalador, filtro de ruido y árbitro que controla el flujo de datos entre el módulo I^2C (cuando es maestro) y otro dispositivo maestro.

La configuración del periférico I^2C se lleva a cabo en la función `I2CA_Init()`.

3.1.5 Interrupciones

Una interrupción es un evento que causa que el microcontrolador cese lo que está haciendo y ejecute un fragmento de código denominado rutina de interrupción. La señal de interrupción puede ser enviada por uno de los periféricos del microcontrolador o por un hardware externo.

El TMS320F28335 es capaz de gestionar 16 líneas de interrupción: 2 no enmascarables (alta seguridad y prioridad) y 14 habilitadas por el usuario. El PIE (*Peripheral Interrupt Expansion*) es el encargado de multiplexar las interrupciones INT1 a INT12 para recibir 8 fuentes de interrupción cada una. Así, se consiguen 96 líneas para fuentes de interrupción internas y externas.

A cada línea se le pueden asignar fuentes de interrupción distintas, en función de los periféricos que se vayan a utilizar. En nuestro proyecto, hemos habilitado 5 interrupciones:

Tabla 3-2. Interrupciones a utilizar.

Nombre de Interrupción	Señal de Interrupción	Puerto PIE	Línea	Hardware
XINT1	GPIO11	1	4	Sensor IR
XINT2	GPIO7	1	5	Sensor IR
XINT13	GPIO14	NMI	NMI	Sensor de Ultrasonidos
SCIRXD	SCIRXD	9	1	-
TINT0	TIMER0	1	7	-

XINT1, XINT2 y XINT13 son interrupciones activadas por hardware externo (XINT13 es XNMI configurada como interrupción externa). SCIRXD es la interrupción asociada al periférico SCI, que se activa con la recepción de datos. Por último, tenemos TINT0, que es la interrupción generada cuando el Timer 0 de la CPU acaba su cuenta.

3.2 Motores y PWM

El movimiento del robot es uno de los aspectos más fundamentales del proyecto. Aunque solo cuenta con dos ruedas, el incorrecto funcionamiento de estas (velocidad inadecuada, deslizamiento o desincronización) impedirá que el robot efectúe su trabajo de manera eficaz.

3.2.1 Configuración

Cuatro GPIOs serán necesarios para la puesta en marcha y control de los motores:

- **GPIO 6** será un puerto de salida y será el actuador que controle la **puesta en marcha y dirección** del **motor izquierdo**.
- **GPIO 15** será un puerto de salida y será el actuador que controle la **puesta en marcha y dirección** del **motor derecho**.
- **GPIO 2** actuará como **ePWM2** (*Enhanced Pulse Width Modulation*) y es el encargado de regular la velocidad de giro del **motor derecho**.
- **GPIO 10** actuará como **ePWM6** (*Enhanced Pulse Width Modulation*) y es el encargado de regular la velocidad de giro del **motor izquierdo**.

Nuestro objetivo con el ePWM es generar señales de 1 kHz de frecuencia y cuyo duty cycle variará en función de la velocidad deseada en los motores. Usaremos dos unidades de ePWM (una para cada motor), pero su configuración inicial es idéntica.

La configuración se realiza en la función `Setup_ePWM()`. Nosotros solo vamos a modificar los módulos Time-Base y Action-Qualifier (explicados en el apartado 3.1.2) y fijaremos los preescaladores de tiempo tal que:

- $\mathrm{CLKDIV} = 1$, es el registro de divisiones del reloj base.
- $\mathrm{HSPCLKDIV} = 2$, es el registro de alta velocidad.

Sabiendo que la frecuencia base del TMS320F28335 es 150 MHz, hay que calcular el valor a introducir en el registro correspondiente al periodo de la señal generada, TBPRD, para que la frecuencia de las señales PWM sean de 1 kHz.

$$TBPRD = \frac{1}{2} \times \frac{f_{sys}}{f_{ePWM} \times CLKDIV \times HSPCLKDIV} = \frac{1}{2} \times \frac{150 \times 10^6}{1 \times 10^3 \times 1 \times 2} = 37500$$

A continuación, activamos el modo *up-down*, que genera una señal triangular, primero incrementando el registro contador de tiempo, TBCTR, desde 0 hasta el periodo previamente establecido y luego decrecerá de nuevo hasta 0. Esta señal sirve como comparador para generar las señales PWM, ePWMxA y ePWMxB.

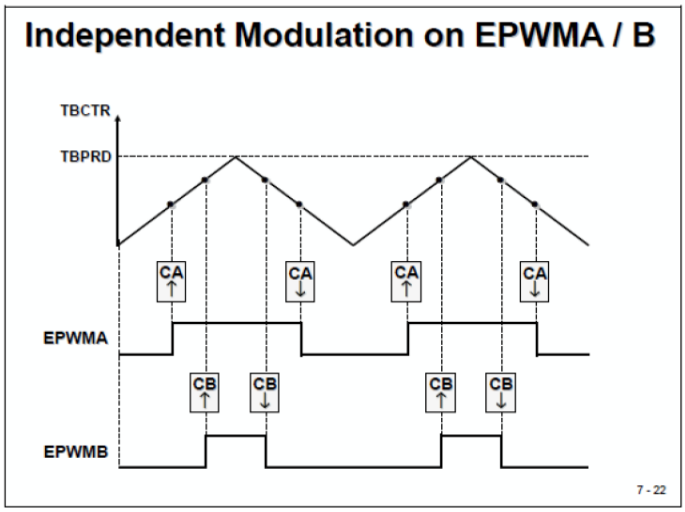

Figura 3-3. Generación de señales PWM a partir de señal triangular.

El registro CMPA será el responsable de controlar el duty cycle de nuestras señales PWM. Debido a como hemos configurado el módulo Action-Qualifier, multiplicando el periodo por valores entre 0 y 1 modificaremos la velocidad de giro de los motores, entonces:

- $CMPA = TBPRD \times 1.0$, el motor girará a la velocidad máxima.
- $CMPA = TBPRD \times 0.0$, el motor estará en estado de freno.

Cuando los GPIO 6 y 15 estén en nivel bajo, los motores girarán en dirección de avance, mientras que si están en nivel alto girarán en dirección de retroceso.

3.2.2 Programación

Definiremos cada estado de movimiento en una función, separando así 7 estados:

Tabla 3-3. Estados de movimiento y funciones.

Estado	Función	GPIO 6	GPIO 15	PWM 2	PWM 6
Avance Lento	`avance1()`	0	0	$TBPRD \times 0.3$	$TBPRD \times 0.3$
Avance Rápido	`avance2()`	0	0	$TBPRD \times 0.5$	$TBPRD \times 0.5$
Giro a la Derecha Lento	`derecha1()`	0	0	$TBPRD \times 0.0$	$TBPRD \times 0.4$
Giro a la Derecha Rápido	`derecha2()`	0	0	$TBPRD \times 0.0$	$TBPRD \times 0.6$
Giro a la Izquierda Lento	`izquierda1()`	0	0	$TBPRD \times 0.4$	$TBPRD \times 0.0$
Giro a la Izquierda Rápido	`izquierda2()`	0	0	$TBPRD \times 0.6$	$TBPRD \times 0.0$
Reposo	`reposo()`	0	0	$TBPRD \times 0.0$	$TBPRD \times 0.0$

Así, cuando se invoque a cualquiera de estas funciones, el robot adoptará el tipo de movimiento descrito por dicha función.

3.2.3 Pruebas

Originalmente, había un nivel de velocidad más, existiendo **`avance3()`**, **`derecha3()`** e **`izquierda3()`**. Sin embargo, al ser nuestro circuito no demasiado amplio, el robot tenía problemas para tomar bien las curvas y acababa siempre saliéndose de la pista. Además, a esa velocidad le era imposible detectar el color de las estaciones, por lo que el movimiento era totalmente errático y no se detenía nunca.

La velocidad más lenta se corresponde con un valor de CMPA de $TBPRD \times 0.3$ en ambos motores porque valores por debajo de ese provocan que el motor se quede ocasionalmente "atascado" y que no tenga la suficiente fuerza para mover las ruedas (especialmente si las pilas no están lo suficientemente cargadas).

El movimiento descrito por las funciones izquierda y derecha son giros cerrados ideales tanto para las curvas de nuestro circuito como para el diseño de tres ruedas (siendo una de ellas una rueda loca) de nuestro robot.

Se podría incorporar un movimiento de marcha atrás poniendo los GPIO 6 y 15 a 1, pero con fines de simplificar este proyecto no se ha previsto el uso de este.

3.3 Sensores IR

La misión de los sensores de infrarrojos es detectar la línea negra que recorrerá el robot. Deben ser capaces de distinguirla claramente del fondo blanco.

3.3.1 Configuración

La configuración de estos sensores es bastante trivial, ya que solo se utilizan 2 GPIOs:

- **GPIO 7** como entrada digital, se corresponde con el **sensor IR derecho**. Cuando se detecte la línea negra se pondrá a nivel alto y en caso contrario, a nivel bajo.
- **GPIO 11** como entrada digital, se corresponde con el **sensor IR izquierdo**. Cuando se detecte la línea negra se pondrá a nivel alto y en caso contrario, a nivel bajo.

Enlazaremos estos GPIOs a las interrupciones externas XINT1 y XINT2. Si igualamos el registro POLARITY a 0, la interrupción se activará siempre que detecte un flanco de bajada, es decir, cuando los sensores detecten el fondo blanco.

3.3.2 Pruebas

Se ha comprobado que el alcance los sensores no es muy grande, con un funcionamiento óptimo a una distancia menor de 5mm del obstáculo. Originalmente, las líneas estaban pintadas con un marcador permanente, lo que provocaba un parpadeo de activación y desactivación ocasional en el sensor, sobre todo en los tramos de curva. Esto ocasionaba movimiento errático del robot, que no era capaz de mantenerse dentro de la línea al variar su movimiento continuamente debido al parpadeo.

Tras muchas pruebas, se ha descubierto que el sensor funciona mucho mejor con pintura mate, probablemente debido a que se distribuye de forma más uniforme y elimina gran parte de los reflejos que ocasionaba el marcador permanente.

3.4 Sensor RGB

El sensor RGB debe realizar mediciones de color de forma continua y con una frecuencia suficiente para no saltarse ninguna estación. Debido a la escasa información del TCS34725 en internet, unido a la complejidad del periférico I^2C por el que se comunica, la configuración de este sensor ha sido una de las tareas más difíciles de este proyecto.

3.4.1 Configuración

Antes que nada, el sensor precisa dos GPIOs:

- **GPIO 32** como SDA, el encargado de transmitir la información entre sensor y microcontrolador.
- **GPIO 33** como SCA, sincroniza los pulsos de reloj de ambos sistemas.

Los cuales configuramos como siempre en la función **`Gpio_select()`**.

En la página web del fabricante se puede encontrar un fichero header (.h) con las direcciones I^2C que utiliza el TCS34725. En nuestro código, se ha bautizado este fichero como *TCS34725.h*, facilitando mucho el manejo de direcciones.

La comunicación entre el microcontrolador y el sensor sigue el siguiente protocolo para la configuración:

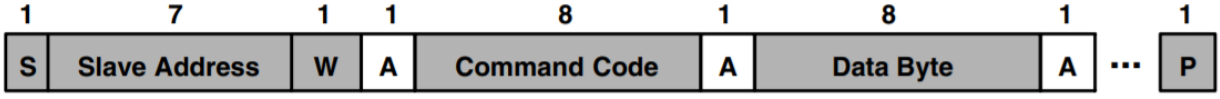

Figura 3-4. Protocolo de escritura para comunicación I^2C.

Este componente en concreto exige de al menos dos bytes para modificar alguno de sus registros.

El primero, *Command Code*, nos permite acceder a la dirección del registro que queremos configurar. Para ello, pondremos a 1 el bit de comando (que es el más significativo) y usaremos los 5 bits menos significativos para definir la dirección. Los 2 bits de en medio pueden utilizarse para variar el modo de lectura (repetida o incrementar registros para leer bytes sucesivos), por ahora usaremos el modo de lectura repetida, dejando ambos bits a 0.

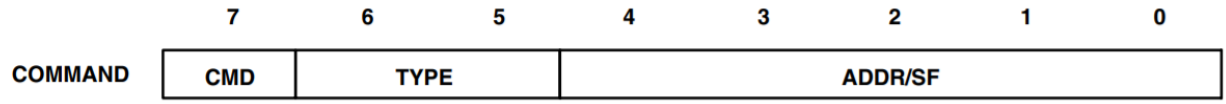

Figura 3-5. Estructura del registro *Command Code*.

El segundo byte será el que contenga la información que queremos escribir. El contenido de este byte variará en función del registro que queramos modificar.

La forma de proceder para configurar el TCS34725 es la siguiente:

1. Primero debemos establecer la dirección del módulo esclavo en el registro I2CSAR. En nuestro caso, la dirección del TCS34725 es 0x29, pero debido al define del fichero *TCS35725.h* podemos poner *TCS24725_SLAVE*.
2. Ponemos a 1 el bit ISR del registro CMDR para activar el periférico y mandar la dirección del TCS34725, "enlazando" ambos dispositivos.
3. Ahora, tras asegurarnos de que el bus de datos se ha liberado, modificamos el registro I2CCNT, que controla el número de bytes que se van a enviar o recibir. Fíjese que en la figura 3-4 son necesarios 2 bytes: el primero, que accede a la dirección del registro a configurar, y el segundo, que contiene la información necesaria para modificar dicho registro.
4. Lo siguiente será establecer en qué modo funcionará el periférico modificando el registro CMDR.

Vamos a detallarlo bit a bit:

- NACKMODE: Cuando el microcontrolador actúe como receptor, enviará una señal de reconocimiento cada vez que efectúe una lectura con éxito. Si ponemos a 1 este bit, se enviará una señal de "no reconocido" durante el siguiente ciclo de reconocimiento. **Lo dejamos a 0.**
- FREE: Este bit controla la acción que toma el módulo I^2C cuando aparece un *breakpoint* de depuración. Si lo ponemos a 1, la transmisión I^2C seguirá activa al encontrarse el punto de interrupción. **Lo ponemos a 1.**
- STT: Es el bit de condición START. **Lo ponemos a 1** para generar una señal de START, necesaria para comenzar la transmisión.
- El siguiente es un bit reservado que **siempre debe estar a 0.**
- STP: Es el bit de condición STOP. Si lo dejamos a 0 deberemos generar la señal de STOP manualmente, pero si lo ponemos a 1, ésta se generará sola cuando el registro contador, I2CCNT (que hemos fijado previamente a 2), llegue a 0. Esto significa que se hayan enviado o recibido el número de bytes definidos. **Lo ponemos a 1.**
- MST: Es el bit de modo maestro. **Lo ponemos a 1** para establecer al microcontrolador como dispositivo maestro y el que genera la señal de reloj en el pin SCL.
- TRX: Es el bit que define si el microcontrolador está en modo receptor o transmisor. Esta vez **lo pondremos a 1**, para que actúe como transmisor.
- XA: Con este bit podemos modificar el tamaño del direccionamiento entre 7 bits (direccionamiento normal) o 10 bits (direccionamiento expandido). **Lo dejamos a 0** para establecer registros de 7 bits.
- RM: Este bit activa el modo repetición, cuya función es la transmisión continua de bytes hasta que la señal STOP se active manualmente. Como nosotros vamos a usar el registro contador, I2CCNT, para determinar el número de bytes a transmitir, **lo dejaremos a 0.**
- DLB: Es el bit de *loopback*, que establece un número de ciclos antes de que la información enviada sea también recibida. **Lo dejamos a 0.**
- IRS: Este bit controla el reinicio del periférico. Si lo ponemos a 0 el periférico se reinicia/desactiva. Si lo ponemos a 1, se activa el periférico, enviado cualquier información que hubiese previamente registrada en el bus de datos. **Lo ponemos a 1.**
- STB: Modo de bit de START aislado. El periférico enviará un byte de START (0000 0001b) tras generar la señal START inicial. **Lo dejamos a 0.**
- FDF: Bit de formato libre. **Lo dejamos a 0** para mantener el formato de direccionamiento de 7 bits definido anteriormente.
- BC: Lo forman los últimos 3 bits. Es el bit encargado de definir cuantos bits se mandan en el byte de datos. **Lo dejamos a 0** para establecer 8 bits por byte de datos.

Por lo tanto, si escribimos todos los bits de forma consecutiva, nos queda 0110111000100000, que en hexadecimal puede escribirse como $0x6E20$. Este valor será el que escribamos en el registro CMDR.

5. Lo siguiente será decirle al periférico que registro suyo queremos modificar. Para ello, debemos utilizar el registro de *Command Code* tal y como se indica en la figura 3-5. Escribimos la información en el registro I2CDXR y esperamos a que se envíe el byte de datos (se puede comprobar verificando que el bit XRDY del registro I2CSTR esté a 0). Este es el primer byte que hemos enviado, nos queda otro para llegar a los 2 que establecimos en I2CCNT.
6. Escribimos de nuevo en el registro I2CDXR, pero esta vez la información para la configuración del registro ya seleccionado. Como este es el segundo byte enviado (cumpliéndose los 2 que se establecieron en I2CCNT), se generará una señal de STOP que pondrá fin a la transmisión.

7. Si queremos modificar otros registros, podemos reiniciar la señal de STOP poniendo el bit SCD del registro I2CSTR a 1. Esto permitirá la transmisión de 2 nuevos bytes, ya que el valor de I2CCNT no ha cambiado.

Ya podemos configurar los diferentes registros del sensor RGB TCS34725. En nuestro caso, vamos a modificar 3 de ellos:

Tabla 3-4. Registros modificados del TCS34725.

Registro	Dirección	Función	Transmisión	Modificación	Efecto
ATIME	`0x01`	Controla el tiempo de integración interno del ADC de los canales RGBC.	0xF6	Tiempo de integración a 24ms.	Mayor precisión y fiabilidad de datos a cambio de una respuesta más lenta.
CONTROL	`0x0F`	Controla la ganancia del canal RGBC.	0x01	Ganancia × 4	Mayor sensibilidad y resolución a cambio de menor estabilidad.
ENABLE	`0x00`	Encendido y apagado del sensor, activar diversas funciones, interrupciones…	0x03	-	Enciende el sensor y activa la detección RGBC.

Hecho esto, el sensor RGB quedaría configurado y listo para enviar la información al microcontrolador.

3.4.2 Programación

Al igual que para configurarlo, debemos usar comunicación I^2C para obtener la información que genere el sensor. Para la lectura de datos debemos seguir el siguiente protocolo:

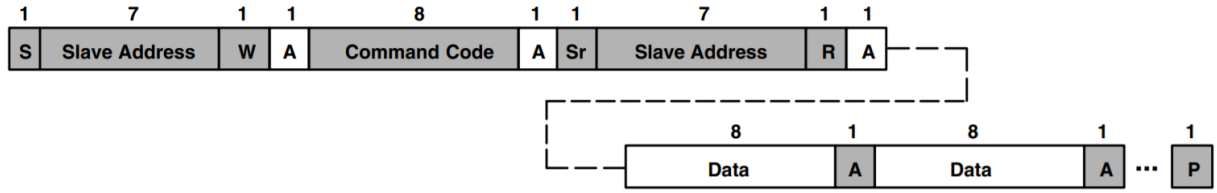

Figura 3-6. Protocolo combinado para lectura de I^2C.

La primera parte es similar al protocolo que utilizamos previamente para configurar los registros del sensor RGB. Primero se especifica la dirección del dispositivo I^2C esclavo (el sensor) y luego se accede al registro de comandos con la dirección del registro que se desea leer.

Los valores RGB, así como el de luz clara, se almacenan en 16 bits (o 2 bytes) en dos registros sucesivos. Por lo tanto, se debe configurar en el registro de comandos la lectura de palabras mediante los bits de tipo *TYPE* (figura 3-5). **Ponemos dichos bits a 01** para habilitar la lectura incremental de registros.

Tabla 3-5. Registros RGB y luz clara.

Registro	Dirección	Descripción
CDATA	0x14	Byte bajo de luz clara
CDATAH	0x15	Byte alto de luz clara
RDATA	0x16	Byte bajo de rojo
RDATAH	0x17	Byte alto de rojo
GDATA	0x18	Byte bajo de verde
GDATAH	0x19	Byte alto de verde
BDATA	0x1A	Byte bajo de azul
BDATAH	0x1B	Byte alto de verde

Ahora que ya hemos accedido a la dirección del registro del **byte de orden bajo** del valor que queramos leer (rojo, verde, azul o claro) podemos proceder a configurar el periférico para la recepción de datos.

Para ello, seguimos el siguiente procedimiento:

1. Ponemos el registro I2CCNT a 2 para indicar que queremos leer dos bytes de datos (byte de orden bajo y byte de orden alto).
2. Configuramos el periférico I^2C como receptor mediante el registro CMDR descrito anteriormente.
3. Esperamos hasta que el bus de datos reciba información (cuando el bit RRDY del registro I2CSTR se ponga a 1).
4. Copiamos en una variable creada por nosotros el valor del registro I2CDRR. Este es el valor del byte menos significativo o byte de orden bajo.
5. Repetimos el paso 3, esperando a que el bus de datos reciba nueva información. Esta vez, debido a la configuración que establecimos previamente en el registro de comandos, se transmitirá la información del registro del byte de orden alto.
6. Sumamos a la variable anterior el valor de este nuevo byte, que combinado con el byte de orden bajo nos proporciona el valor real.

Este proceso tenemos que repetirlo cada vez que queramos obtener **uno** de los 4 posibles valores que ofrece este sensor. Es decir, hay que repetirlo para obtener el valor de luz roja, luz verde, luz azul y luz clara. Por ello, nosotros hemos creado una función `I2C_leer_dos_bytes(int registro)` que tiene como argumento de entrada el registro a leer (del byte de orden bajo). La función devuelve el valor detectado por el sensor.

Sin embargo, para obtener un valor RGB estandarizado (escala de intensidad de 0 a 255) debemos realizar una serie de cálculos adicionales. Esto es tal que:

$$RGB_{(0,255)} = \frac{RGB_s}{CLARO_s} \times 255$$

Donde:

- $RGB_{(0,255)}$ es el valor RGB en escala de intensidad de 0 a 255.
- RGB_s es el valor RGB proporcionado directamente por el sensor.
- $CLARO_s$ es el valor de luz clara proporcionado directamente por el sensor.

Por lo que hemos creado otra función, **`obtenerRGB`**`(float *rojo, float *azul, float *verde)`, que nos permite hacerlo de forma más rápida. Los argumentos de entrada son punteros a variables del tipo *float* que representan la intensidad RGB en una escala de 0 a 255. El uso de punteros a variables declaradas en la función principal nos permite monitorizar su valor para ajustar y calibrar el sensor posteriormente.

En lenguajes de programación, de manera usual, la intensidad de cada una de las componentes se mide según una escala que va del 0 al 255 y cada color es definido por un conjunto de valores escritos entre paréntesis (correspondientes a valores "R", "G" y "B") y separados por comas.

Por ejemplo, el rojo se representa tal que (255, 0, 0), el verde, (0, 255, 0), y el azul, (0, 0, 255). El negro, que se define como la ausencia de color, se obtiene cuando las 3 componentes son 0 y el blanco se obtiene cuando los tres colores primarios tienen su máximo valor, (255, 255, 255).

Acotando estos valores se pueden definir las diferentes 8 estaciones de color que hemos dispuesto en nuestro circuito.

3.4.3 Pruebas

La calibración del sensor TCS34725 se ha hecho a base de prueba y error, comparando la tonalidad detectada (se ha utilizado un convertidor RGB online para generar el color detectado) y la real.

Las conclusiones son, que con la ganancia (× 1) y el tiempo de integración (2,4ms) por defecto, los resultados eran, mayoritariamente, erróneos. Las tonalidades se detectan mucho más oscuras que sus contrapartes reales, ya que el valor de las intensidades raramente sobrepasaba 150. Esto hace que la detección de algunos colores de tonalidad clara (incluido el blanco), sea imposible, ya que siempre se devuelve un color grisáceo.

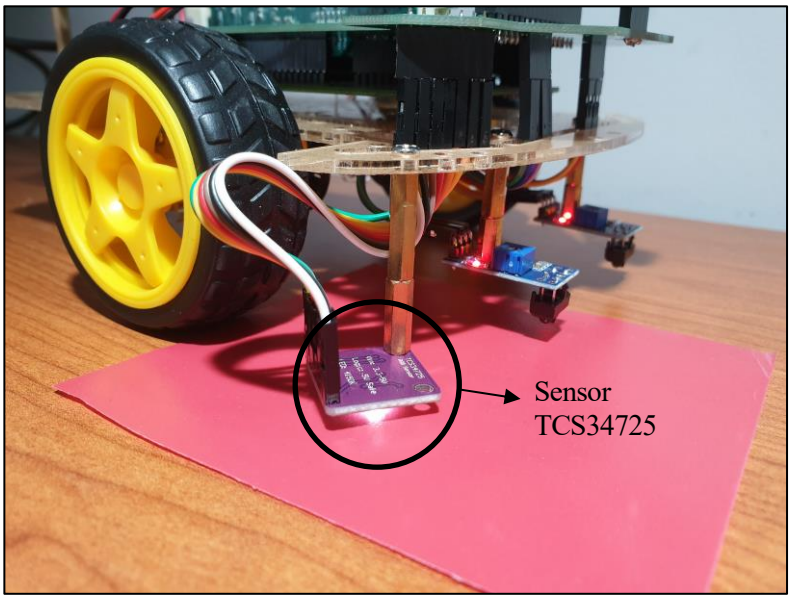

Figura 3-7. TCS34725 con LED neutro encendido.

Con esta configuración el sensor tenía problemas para diferenciar algunas de las estaciones y las confundía muy a menudo. Incluso detectaba a veces el fondo blanco como la estación de ALERGOLOGÍA, siendo esta de un color verde muy oscuro.

Posteriormente, se probó a incrementar la ganancia de forma drástica (hasta × 60), manteniendo el mismo tiempo de integración (2,4ms). Se comprobó que la tonalidad detectada era mucho más acorde con la real. Los colores claros se detectaban ahora casi a la perfección, muchas veces alcanzando la intensidad de 255 en aquellos muy intensos.

Sin embargo, los valores proporcionados variaban muy rápidamente, incluso estando el sensor fijo, provocando de nuevo múltiples errores de medición. El resultado es que, aunque se detecte una estación, los valores cambian tan rápido que, aunque el robot siguiera sobre dicha estación, la detección se producía intermitentemente y con poca precisión.

Este comportamiento inestable e incapaz de mantener unos valores fijos para un mismo color nos llevó a considerar un aumento en el tiempo de integración para mejorar la fiabilidad del sensor.

Por último, se hicieron pruebas con una ganancia × 4 y un tiempo de integración de 24ms. Aunque los colores detectados siguen siendo de una tonalidad más oscura que los reales, esta ganancia es suficiente para diferenciar las estaciones del circuito. Al tener una menor sensibilidad y un tiempo de integración más alto se ha conseguido también que los valores se mantengan fijos mientras el robot esté sobre la estación.

Expression	Type	Value	Address
rojo	float	124.999992	0x0000C08A@Data
verde	float	62.3076897	0x0000C08E@Data
azul	float	67.6923065	0x0000C08C@Data
Add new expression			

Figura 3-8. Valores RGB proporcionados por el sensor.

Nuestra preocupación a la hora de subir el tiempo de integración era si el robot sería capaz de detectar las estaciones a tiempo (sobre todo cuando va a la mayor velocidad), pero tras numerosas pruebas no se ha determinado ningún problema.

Si quisiéramos obtener la tonalidad real del color que está detectando el sensor, mi sugerencia sería establecer una ganancia × 60 y un tiempo de integración lo suficientemente grande como para que los valores no fluctúen en exceso. No obstante, debido a la relativa necesidad de una detección rápida y constante en este proyecto, se optará por la configuración previamente descrita.

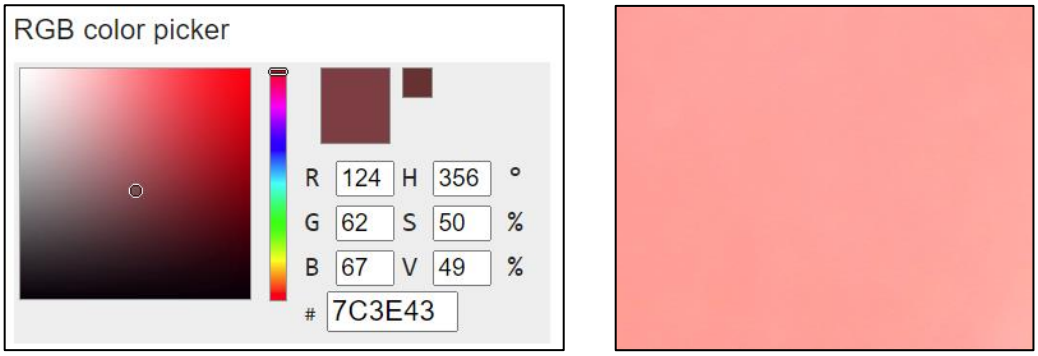

Figura 3-9. Tonalidad registrada por el sensor (izquierda) y tonalidad real (derecha).

3.5 Sensor de Proximidad

El sensor de proximidad o sensor de ultrasonidos funciona midiendo la distancia que hay entre una onda sónica que genera y el obstáculo que la hace rebotar. Aunque no necesita una configuración per se, su funcionamiento se basa en un proceso y una serie de cálculos que detallaremos a continuación.

3.5.1 Programación

El funcionamiento del HC-SR04 sigue el siguiente procedimiento:

1. Activar el pin de TRIGGER durante 10µs.
2. El sensor enviará 8 pulsos ultrasónicos de 40 kHz y activará el pin de ECHO.
3. El ECHO permanecerá activo hasta que llegue el rebote del ultrasonido.
4. Calculamos la distancia mediante la fórmula:

$$d(m) = t_{ECHO}(s) \times \frac{340\ m/s}{2}$$

   Donde:

   - d es la distancia al obstáculo en metros.
   - t_{ECHO} es el tiempo en segundos que el pin ECHO se mantiene activo.
   - $340\ m/s$ es la velocidad del sonido.
   - Se divide por dos por que el sonido recorre la distancia 2 veces (ida y vuelta).
5. Se recomienda esperar 60ms entre medición y medición para evitar el solapamiento de pulsos.

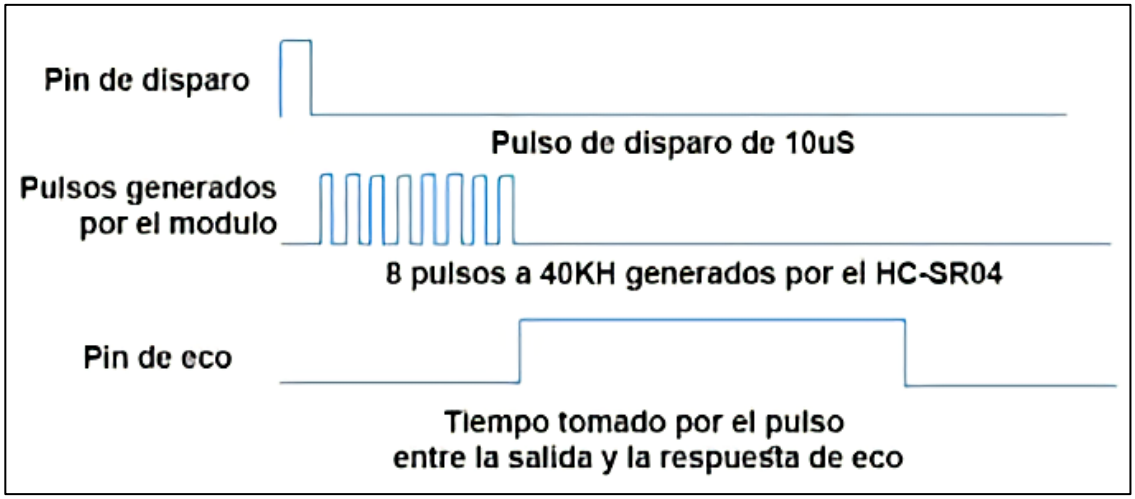

Figura 3-10. Funcionamiento del sensor HC-SR04.

Para facilitar el proceso, nosotros hemos creado la función **`midedistancia`**`(int *flag_echo)` cuyo argumento de entrada es un puntero a la variable *flag_echo* que debe tomar el valor 1 cuando se active el pin de ECHO. Sigue el siguiente procedimiento:

1. Nos aseguramos de que el GPIO 13, que corresponde al pin de TRIGGER, esté desactivado.
2. Configuramos el TIMER 2 de la CPU para una frecuencia de 150 MHz y un periodo de 4294967295 microsegundos (0xFFFFFFFF en hexadecimal), el máximo permitido para el timer.
3. Ponemos a 1 el GPIO 13, activando el TRIGGER. A continuación, con la función **`delay`**`(long int us)`, esperamos 10µs. Luego, ponemos al 0 el GPIO 13, desactivando el TRIGGER.

4. Ahora se activará el pin de ECHO, poniendo a 1 el GPIO 14 y activando la interrupción *obstaculo,* asociada a dicho pin. En esta interrupción, una condición comprobará que *flag_echo* no está a 1, por lo que se iniciará la cuenta atrás del TIMER 2, configurado previamente. También pondrá la variable bandera *flag_echo* a 1, simbolizando la activación del pin ECHO.
5. Se retrasa 10ms la función mientras se espera el retorno del ultrasonido.
6. Ahora pueden pasar dos cosas:
    - El pin de ECHO se desactiva, iniciando de nuevo la interrupción, que pondrá *flag_echo* a 0 y parará la cuenta atrás del TIMER 2.
    - Se acaban los 10ms de retardo y el pin de ECHO no se ha desactivado. En cuyo caso ponemos manualmente *flag_echo* a 0.
7. Obtenemos los microsegundos contabilizados por TIMER 2 restando al periodo máximo (0 xFFFFFFFF) el valor del registro TIM. Aplicamos los cálculos previamente descritos para sacar la distancia en **centímetros.**
8. La función devuelve este valor.

El paso 6 supone que la máxima distancia que detectará el sensor serán unos 160cm. Aumentando el valor de retardo se aumentaría también la máxima medición posible, pero en nuestro proyecto no contemplamos distancias tan grandes, por lo que ese valor es suficiente.

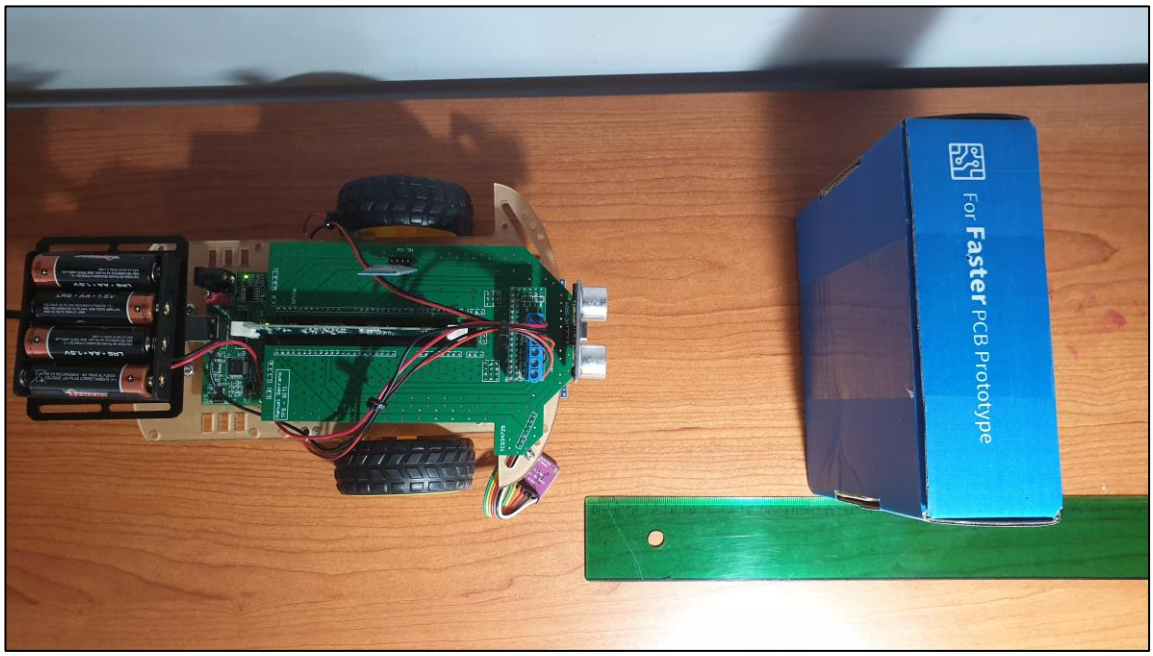

Figura 3-11. Distancia de 10cm entre sensor y obstáculo (vista cenital).

3.5.2 Pruebas

Las pruebas con el sensor HC-SR04 han sido bastante sencillas. Los principales errores se debían a un mal entendimiento de la naturaleza de los timers de la CPU en el microcontrolador TSM320F28335, que realizan una cuenta **regresiva.** Por lo tanto, el tiempo transcurrido se obtiene restando al periodo establecido el valor contabilizado por el timer.

Por lo demás, el sensor no presenta una precisión absoluta, pero cumple con creces su función de detectar obstáculos próximos. La precisión es mayor cuanto más cerca se encuentra el obstáculo. También se ha observado que cuanto más uniforme y perpendicular se encuentra el obstáculo al plano que lo une con el sensor, mayor precisión presenta éste en la medición.

Expression	Type	Value	Address
distancia	float	10.539999	0x0000C090@Data
Add new expression			

Figura 3-12. Valor proporcionado por el sensor HC-SR04.

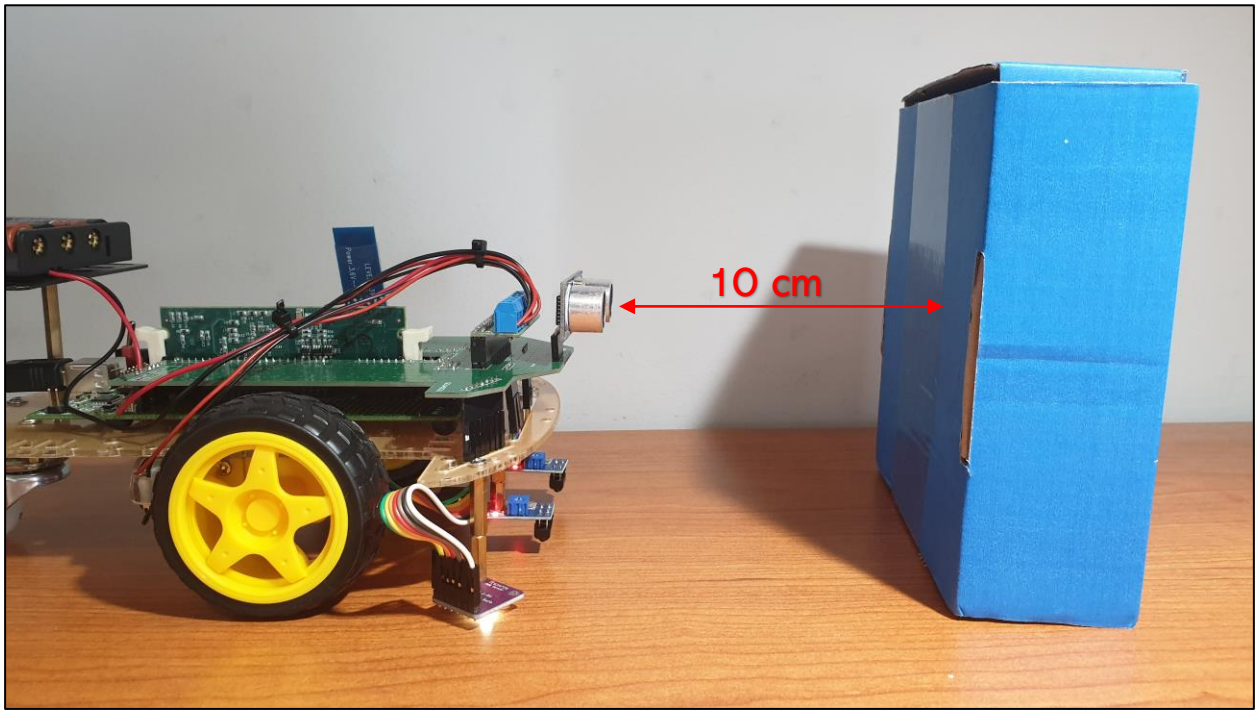

Figura 3-13. Distancia de 10cm entre sensor y obstáculo (vista lateral).

3.6 Módulo Bluetooth

El módulo bluetooth HC-06 se comunica con el microcontrolador mediante el periférico SCI. A parte de la configuración de este periférico, debemos elegir cautelosamente la información a transmitir al microcontrolador y como éste la procesará.

3.6.1 Configuración

Nuestra primera tarea es configurar el periférico SCI. Para ello, vamos a modificar los siguientes bits del registro SCICCR, que define el formato, protocolo y modo de comunicación del SCI:

- SCICHAR: Estos 3 bits controlan la longitud de la cadena de caracteres, desde 1 a 8 bits. Nosotros lo **ponemos a 7** (o 111 en binario) para fijar cadenas de 8 bits.
- ADDRIDLE_MODE: Este bit selecciona uno de los dos protocolos de multiprocesos. Lo **ponemos a 0** para activar el modo libre. Si lo pusiésemos a 1, tendríamos que reservar un bit para la dirección.
- PARITYENA: Es el bit que activa la función de paridad. **Lo dejamos a 0** porque no necesitamos bit de paridad.
- STOPBITS: Regula el número de bits de STOP. El receptor solo comprueba un bit de STOP, así que **lo dejamos a 0** (1 bit de STOP).

A continuación, ponemos el bit RXENA del registro SCICTL1 a 1 para habilitar la línea de recepción. Lo siguiente será configurar la velocidad de la comunicación. Se configuran el valor máximo y mínimo de velocidad en baudios (SCIHBAUD y SCILBAUD) para 9605 Hz, que es el valor más próximo alcanzable a los 9600 Hz a los que transmitirá el módulo HC-06.

Se habilitará la interrupción por recepción poniendo a 1 el bit RXBKINTENA del registro SCICTL2. Por último, se libera al SCI del reset poniendo a 1 el bit SWRESET del registro SCICTL1, quedando el periférico listo para usar.

3.6.2 Programación

La configuración previa hará que cada vez que nueva información sea enviada por el HC-06 al microcontrolador se genere una señal de interrupción que pausará el programa principal para interpretar los datos recibidos.

En dicha interrupción la información se almacenará en la variable char *buffer*, que tiene una capacidad de hasta 100 caracteres. Además, se pondrá a 1 la bandera *flag_rx*, que significa que hay información que procesar. El código usado para interpretar esta información se analizará en el capítulo posterior.

3.6.3 Pruebas

En general, la calidad de la comunicación entre el dispositivo móvil y el microcontrolador mediante este componente es muy buena. La velocidad de transmisión es lo suficientemente rápida como para que los cambios se produzcan instantes después de enviar la información.

La aplicación *Bluetooth Electronics*, que detallaremos en el siguiente capítulo, es muy personalizable y nos permite elegir meticulosamente que caracteres enviar al microcontrolador. Esto es de gran utilidad, pues podemos fijar determinados caracteres para que sirvan como terminadores de cadena, previa configuración en el código del programa.

El único inconveniente es la ocasional desconexión del módulo HC-06 con nuestro móvil, especialmente si lo bloqueamos o lo alejamos demasiado. Se ha notado también un gran consumo de batería, aunque podría ser debido a la aplicación móvil y no al módulo HC-06 en específico.

4 PROGRAMACIÓN DEL ROBOT

Con todos los componentes configurados y funcionando, estamos listos para diseñar el algoritmo que determine el funcionamiento de nuestro robot. Algunas funciones ya descritas en el capítulo anterior nos servirán ahora como base para crear nuestra función principal. Además, también detallaremos el funcionamiento de las interrupciones y como se integran con la función principal de nuestro robot.

Podemos distinguir tres fases claramente diferenciables:

1. **Tratamiento de la información:** Debemos decidir qué información se le transmite al robot y programarlo para que éste sea capaz de interpretarla. El robot debe actuar en consecuencia a dicha orden siempre que esta no interfiera con determinadas órdenes anteriores.
2. **Función principal:** Con la información que hemos enviado, el robot tendrá un objetivo que cumplir. Esta parte engloba que acciones debe tomar el robot para llevar a cabo con éxito las órdenes que le han sido transmitidas.
3. **Interrupciones:** Mientras el robot esté desempeñando su función principal pueden surgir interrupciones que le obliguen a detener lo que está haciendo para corregir o manejar algún imprevisto. Éstos pueden ser provocados (recepción de nuevas órdenes) o no (el robot se sale de la línea negra).

Todo el código de las funciones, interrupciones y configuración puede encontrarse íntegramente en los Anexos, siendo este capítulo un análisis detallado de dicho código.

4.1 Tratamiento de la Información

Lo que define el comportamiento del robot es en gran parte las órdenes que nosotros le demos. Es información que se transmitirá desde nuestro dispositivo móvil al módulo HC-06 y de ahí al microcontrolador. Para ello, vamos a ayudarnos de dos aplicaciones muy útiles diseñadas para este tipo de proyectos de electrónica: *BT Voice Control for Arduino* y *Bluetooth Electronics.*

Sin embargo, antes que nada, debemos definir las acciones que tomará nuestro robot en función de la información proporcionada.

4.1.1 Comandos y Órdenes

El comportamiento del robot es relativamente sencillo. Su función es desplazarse de una estación a otra y detenerse si se topa con algún obstáculo. Somos nosotros los que le diremos a qué estación queremos que vaya, **independientemente de donde se encuentre.**

Además, el robot debe ser capaz de desplazarse a dos velocidades y detenerse en cualquier punto del recorrido. Esto también lo determinaremos nosotros. Por lo tanto, al robot se le deben poder comunicar las siguientes acciones:

- Desplazarse hasta la estación fucsia: la **Recepción**.
- Desplazarse hasta la estación lima: **Medicina General.**
- Desplazarse hasta la estación marrón: **Oftalmología.**
- Desplazarse hasta la estación cyan: **Neurología.**
- Desplazarse hasta la estación amarilla: **Dermatología.**
- Desplazarse hasta la estación naranja: **Pediatría.**
- Desplazarse hasta la estación verde: **Alergología.**
- Desplazarse hasta la estación azul: **Urgencias.**
- **Aumentar la velocidad** en caso de que este no vaya ya a la velocidad máxima.
- **Disminuir la velocidad**, solo en caso de que vaya a la velocidad máxima.
- **Detenerse** por completo.

El único caso en el que el robot desobedecerá estas órdenes será cuando un obstáculo le bloquee el camino, lo que provocará su detención hasta que dicho obstáculo sea retirado. Mientras el robot esté obstaculizado no será posible enviarle nuevas órdenes.

Además, en el trayecto de una estación a la siguiente se podrá modificar la velocidad del robot, pero no cambiar su destino. Es decir, si se ha fijado la estación naranja como su destino y a mitad del camino le ordenamos que vaya a la estación azul, el robot continuará hasta la estación naranja.

Veamos ahora las aplicaciones que usaremos para enviarle los comandos al robot.

4.1.2 *BT Voice Control for Arduino*

Aunque, como su nombre indica, esta aplicación ha sido diseñada para *Arduino*, es perfectamente compatible con el módulo HC-06 y nuestro proyecto. Su finalidad es muy sencilla: transcribir nuestros comandos de voz y mandar la información al dispositivo bluetooth conectada.

Su interfaz es muy simple, pues consta de un botón circular que pulsaremos para empezar a hablar. Cuando hayamos terminado nuestro dictado, la aplicación reconocerá la voz automáticamente y la transcribirá a texto. Este texto se enviará al módulo HC-06 que a su vez enviará, carácter a carácter, al microcontrolador.

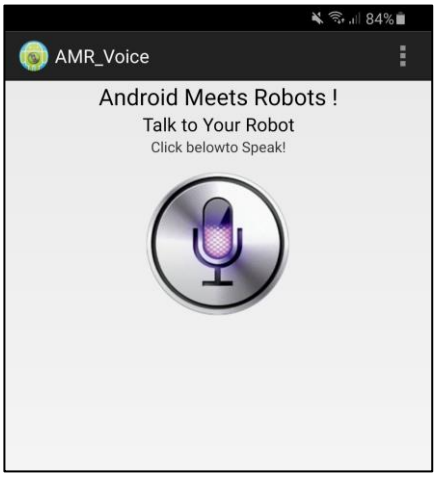

Figura 4-1. Interfaz de la aplicación *BT Voice Control for Arduino*.

Al terminar de enviar la cadena, la aplicación también envía un único carácter '**#**'. El código identifica dicho carácter como el final de la sentencia e interpreta la información proporcionada. Los comandos de voz deben seguir las siguientes reglas:

- Si queremos fijar un nuevo destino, se deberá decir el nombre de la especialidad médica (o recepción en su caso) claramente, pudiéndose acompañar de una frase sin problema. Por ejemplo: "*Quisiera ir **Oftalmología**, por favor".*
- Para aumentar la velocidad del robot, las palabras clave "deprisa" o "rápido" serán suficientes. Recordemos que la velocidad solo se puede aumentar si el robot se está desplazando a velocidad normal. Por ejemplo: "*Más **rápido***" o "***Deprisa**, por favor".*
- Para reducir la velocidad del robot, se usarán las palabras clave "lento" o "despacio". El robot solo reducirá la velocidad si va por encima de la velocidad normal. Por ejemplo: "*Más **lento***" o "***Despacio***".
- Para ordenar al robot que se detenga, se usarán las palabras clave "**detente**" o "**para**". Una vez parado, el robot podrá reanudar su marcha con las palabras "**vamos**" o "**sigue**".

Si preferimos usar una interfaz visual a los comandos de voz, podemos utilizar la siguiente aplicación.

4.1.3 *Bluetooth Electronics*

Esta aplicación ha sido específicamente desarrollada para su uso con los módulos bluetooth HC-05 y HC-06. Su contenido incluye una serie de paneles altamente personalizables que nos permiten seleccionar que caracteres se envían al microcontrolador.

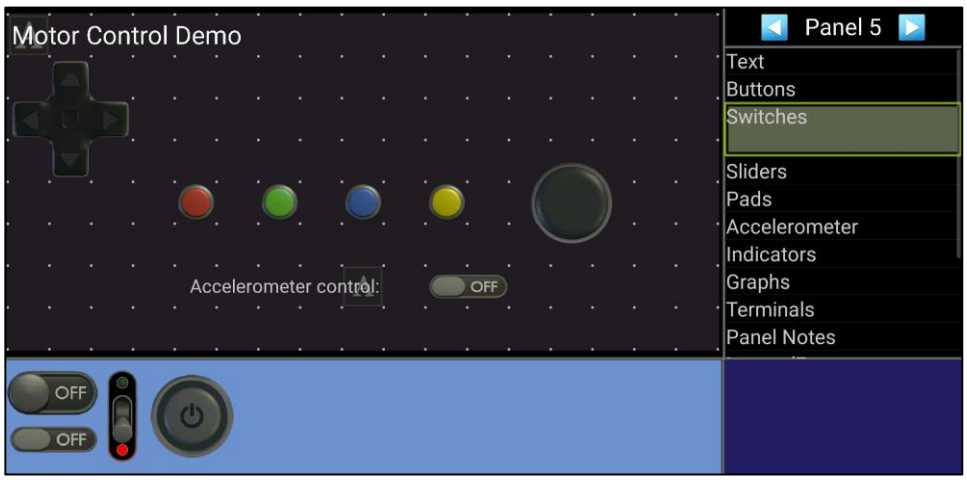

Figura 4-2. Ejemplo de panel configurable de la aplicación *Bluetooth Electronics*.

Entre los posibles ítems que podemos añadir al panel se encuentran botones, interruptores, acelerómetros, indicadores visuales y barras deslizantes. Cada uno se puede personalizar para determinar que caracteres se envían al pulsarlos (en el caso de los botones) o a que valores se pueden ajustas (barras).

Dado que el modo por defecto en nuestro robot será el de comunicación por voz (usando la app *BT Voice for Arduino*) debemos incorporar algún ítem que envíe una señal que habilite la interfaz y desactive los comandos por voz. Esto se hace para que no haya errores de lectura, ya que mientras que la aplicación por voz envía las cadenas de caracteres enteras, esta solo envía determinados caracteres, por lo que podrían llegar a mezclarse si ambas aplicaciones están activas al mismo tiempo.

Dicho esto, nosotros utilizaremos los siguientes elementos:

- 8 botones, cada uno para una de las 8 estaciones distintas. Se usarán para fijar el destino del robot.
- Una barra deslizante (o *slider*) que se usará para controlar la velocidad del robot. Tendrá tres niveles, representando los estados de detención, velocidad normal y velocidad superior.
- Un interruptor que, cuando esté activado, habilitará la interfaz táctil y desactivará los comandos. Para volver a usar los comandos de voz debe desactivarse el interruptor primero.

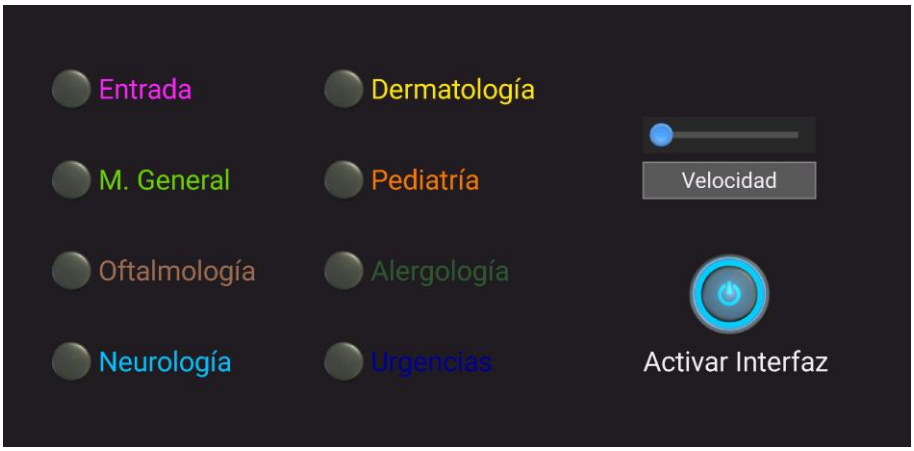

Figura 4-3. Interfaz para control del robot.

Cada elemento del panel envía las siguientes señales:

- El botón de **entrada** envía una "**e**" cada vez que es pulsado.
- El botón de **medicina general** envía una "**m**" cada vez que es pulsado.
- El botón de **oftalmología** envía una "**o**" cada vez que es pulsado.
- El botón de **neurología** envía una "**n**" cada vez que es pulsado.
- El botón de **dermatología** envía una "**d**" cada vez que es pulsado.
- El botón de **pediatría** envía una "**p**" cada vez que es pulsado.
- El botón de **alergología** envía una "**a**" cada vez que es pulsado.
- El botón de **urgencias** envía una "**u**" cada vez que es pulsado.
- La barra deslizante enviará los valores "**0**", "**1**" o "**2**" cuando se modifique su posición anterior.
- El interruptor enviará el símbolo "**+**" cuando se encienda y "**-**" cuando se apague.

Por supuesto, tanto esta aplicación como *BT Voice Control* necesitan que el móvil esté emparejado con el HC-06 para funcionar.

4.1.4 Lectura e interpretación

Como comentamos en el capítulo anterior, cada vez que detecte una nueva recepción de información tendrá lugar una interrupción que parará la función principal. Hemos nombrado a esta interrupción **`SCIA_RX_isr()`**. El procedimiento que sigue es el siguiente:

1. Pone a 1 la variable bandera *flag_rx*, que simboliza la entrada al microcontrolador de nueva información.
2. Se recogen en la variable de tipo vector de caracteres, *buffer*, el contenido del registro SCIRXBUF. Cada elemento del vector *buffer* es un carácter.
3. Se comprueba si el último carácter registrado es un '+' o un '-'.
    a. Si es un '+' se pone a 1 la variable *modo*, que es la que controla si se está usando los comandos de voz (cuando vale 0) o la interfaz táctil (cuando vale 1). Además, se reinicia el vector *buffer* con la función propia **`inicero(`**`buffer`**`)`**, que escribe el valor 0 en todos los elementos del vector.
    b. Si es un '-' se pone a 0 la variable *modo*.
4. Si *modo* tiene el valor 0 se comprueba si el último elemento es un '#', en cuyo caso significaría el final de una cadena de caracteres transcrita de un comando de voz. Se pone a 1 la variable *flag_rx0*, simbolizando que la nueva información procede de un comando de voz.
5. Se limpia el registro de la línea de interrupción INT9.

La tarea de la interrupción es confirmar la recepción de nuevos datos y, en caso afirmativo, discernir si provienen de comandos por voz o de la interfaz táctil. Activa una bandera que permite que se llame a la función **`lectura(`**`buffer,*dest,*flag_rx0,*modo,*vel`**`)`**.La función tiene los siguientes argumentos:

- *buffer* es el vector de caracteres que contiene la información transmitida por el HC-06.
- **dest* es un puntero a la variable entera *dest*, que nos indica el destino fijado para robot. Cuando haya un destino, el valor de la variable corresponderá a un entero entre 1 y 8, correspondiéndose cada uno con una de las 8 estaciones. Si la variable vale 0, significa que no hay ningún destino asignado y el robot deberá estar en reposo y a la espera de órdenes.
- **flag_rx0* es un puntero a la variable del mismo nombre ya mencionada antes. Indica que se ha recibido una cadena de caracteres que proviene de comandos de voz.
- **modo* es un puntero a la variable *modo*, que indica si está activado el modo de comandos de voz o de interfaz táctil.
- **vel* es un puntero a la variable *vel*, que indica la velocidad del robot. Tomará el valor 0, 1 o 2, siendo estos detención, velocidad normal y velocidad superior respectivamente.

Aunque esta función **`lectura`** se ejecuta en el bucle de la función principal, se explicará su contenido en este capítulo ya que es la encargada de interpretar la información proporcionada. Su procedimiento es tal que:

1. Se declaran las variables enteras *d* y *v* y se igualan a los valores de *dest* y *vel*, respectivamente. Esto se hace para evitar errores con el tratamiento de los punteros.
2. Ahora la función se divide en dos caminos diferentes dependiendo del modo de transmisión que esté activado. Para evitar confusiones usaremos la letra 'a' para designar al modo de control por voz y 'b' al modo de interfaz táctil. La división se realiza mediante el valor de la variable *modo*.
3. Si *dest* vale 0 (lo cual significa que no hay ningún destino asignado al robot), se procede de la siguiente manera:

   a. En el caso de control por voz se usarán condicionales con la función **`strstr(`**`buffer, "elementoabuscar")`, una función integrada que nos permite buscar una determinada configuración de caracteres en una cadena. Sus argumentos de entrada serán el vector *buffer* y la configuración a buscar entrecomillada. En nuestro caso buscamos el nombre de las especialidades médicas disponibles para asignarlas como destino. En caso de encontrarlas se le asignará a la variable *d* el valor correspondiente.

   b. Si en cambio estamos usando la interfaz táctil bastará con comprobar el último carácter del vector *buffer*, si este se corresponde a la letra inicial de cualquiera de las especialidades médicas, modificamos la variable *d* con el valor correspondiente.

4. Ahora se buscarán comandos que modifiquen la velocidad del robot. De la misma manera:

   a. Se buscan las palabras clave mencionadas en apartados anteriores ("rápido", "detente", etc.) usando de nuevo la función **`strstr.`** En caso de encontrarlas se modifica la variable *v* en consecuencia.

   b. Se comprueba si el último elemento del vector *buffer* son los valores 0, 1 o 2, en cuyo caso se modifica la variable *v* consecuentemente.

5. Se reinicia *buffer* mediante la función **`inicero`** y se copian las variables *d* y *v* en **dest* y **vel* respectivamente.

6. Se pone a 0 la variable *flag_rx*, la información ha sido leída e interpretada.

Hemos conseguido transmitir nuestras órdenes a las variables *dest* y *vel*, que son las que, en esencia, controlan el comportamiento de nuestro robot. Sigue un esquema para resumir el proceso entero:

4.2 Función Principal

Una vez proporcionada la información, el robot se pondrá en marcha para desempeñar la tarea que le ha sido encomendada. Su objetivo es llegar, con la velocidad establecida, hasta la estación que le haya sido asignada. Esto se realiza mediante 4 funciones que, a su vez, conforman el bucle iterativo de la función principal.

- ***midedistancia**:* Es la función encargada de comprobar que no hay obstáculos en los 10cm por delante del robot. Hace uso de las funcionalidades del sensor HC-SR04, así como de los Timers 0 y 2 de la CPU.
- ***obtenerRGB:*** Su tarea es la identificación de los valores RGB que esté viendo el sensor TCS34725 y clasificarlos en distintos colores. Nos permite distinguir las distintas estaciones y avisar al robot cuando esté atravesando alguna.
- ***recorrido**:* Es la función que define que camino seguirá el robot para llegar hasta su destino. Utiliza la tanto la información proporcionada por nosotros como por los sensores para decirle al robot que desvíos tomar y en que estación detenerse.
- ***lectura**:* Esta es la función que vimos en el capítulo anterior. Su función es interpretar información entrante del HC-06 y dar valor a ciertas variables en consecuencia. Solo se le invoca cuando se detecta la recepción de nuevos datos.

Empezaremos analizando la función *midedistancia*.

4.2.1 Medición de Proximidad

Es posible que en el camino el robot se encuentre con obstáculos que le impidan continuar su avance. Esta función se encarga de detectar dichos obstáculos dándole un valor a la distancia entre el robot y el objeto frontal más próximo.

Nosotros hemos definido 10cm como la mínima distancia que debe detectar el robot antes de detenerse. Es decir, cualquier objeto a menos de 10cm se considerará obstáculo y supondrá la detención del robot hasta que dicho obstáculo se retire.

La función `midedistancia(int *flag_echo)` ya fue descrita en el apartado 3.5.1 (*Pruebas del sensor HC-SR04*), por lo que no se repetirá el proceso que sigue aquí. La variable *distancia* es el resultado de dicha función y expresa, como su nombre indica, la distancia entre la parte frontal del robot y el objeto más próximo en cm.

Tras ejecutar la función, se ha añadido un bucle *while* que previene el avance del código siempre que *distancia* sea inferior a 10. En este bucle se ejecutará repetidamente la función *midedistancia*, hasta que el valor de *distancia* vuelva a ser mayor que 10, en cuyo caso se sale del bucle y se continua con la siguiente línea del código.

Hay que destacar que entre medición y medición se debe esperar unos 60ms, por lo que se usa la función `delay(int us)`, que usa nanosegundos como argumento para retrasar la siguiente línea de código.

4.2.2 Detección de Color

Durante el recorrido, y siempre que no se haya topado con ningún obstáculo, el sensor RGB del robot (colocado a la derecha de la parte frontal del mismo) efectuará medidas de color en cada iteración del bucle de la función principal.

La mayor parte del tiempo solo detectará el fondo blanco, pero, eventualmente, el robot alcanzará una de las 8 estaciones repartidas por el circuito y necesitará que la función *obtenerRGB* le proporcione esa información. La función devuelve la variable *estación*, que toma valores entre 0 y 8 dependiendo del color detectado.

La identificación de colores sigue la siguiente estructura:

1. Como vimos en el apartado 3.4.2 (*Pruebas del sensor RGB*), el TCS34725 y el microcontrolador se comunican por protocolo I^2C, haciendo uso de la función *I2C_leer_dos_bytes* que hemos creado nosotros mismos. Esta función se invoca 4 veces en *obtenerRGB*, para devolver los valores de luz roja, verde, azul y clara.

2. Con estos 4 valores, se obtienen las franjas de color RGB que miden la intensidad de cada uno de los tres componentes según una escala que va del 0 al 255.
3. Podemos clasificar estos valores para definir márgenes que representan los colores de las estaciones mediante condicionales. En caso de que se cumpla condición, asignamos a la variable *estacion* el color correspondiente (mediante identificadores numéricos). La siguiente tabla resumen como hemos clasificado los valores RGB:

Tabla 4-1. Clasificación de franjas RGB y colores.

Color	Estación	Identificador Numérico	R	G	B
Fucsia	Entrada	1	100 a 120	65 a 80	90 a 105
Lima	Medicina General	2	45 a 65	120 a 140	60 a 80
Marrón	Oftalmología	3	105 a 120	95 a 110	90 a 105
Cyan	Neurología	4	30 a 50	95 a 115	115 a 135
Amarillo	Dermatología	5	90 a 110	100 a 120	45 a 65
Naranja	Pediatría	6	140 a 160	70 a 90	50 a 70
Verde	Alergología	7	85 a 95	110 a 120	110 a 120
Azul	Urgencias	8	50 a 70	80 a 95	155 a 170
Blanco	-	0	-	-	-

4. Cuando se cumplen valores para los tres intervalos (R, G y B) de algún color se iguala la variable *estación* al identificador numérico que corresponde a dicho color. La variable adquiere el valor 0 cuando RGB no encaja en ninguna de las franjas descritas en la tabla, entendiéndose que se está detectando el fondo blanco.
5. Se devuelve la variable *estacion*.

Esta variable será fundamental para la siguiente función, que determinará el recorrido que sigue el robot por el circuito.

4.2.3 Recorrido del Robot

Ya tenemos el destino al que debe llegar el robot (*dest*), la velocidad a la que debe desplazarse (*vel*) y podemos monitorear regularmente su posición (*estación*). Estas tres variables nos permiten definir el comportamiento y recorrido que debe seguir el robot para llegar a su destino.

La función **`recorrido`** `(int dest, int vel, int estacion)` cumple este cometido. Sus argumentos de entrada son las tres variables anteriores y no devuelve ningún valor de forma directa. Es la función encargada de decidir qué acción debe tomar el robot cuando se alcanza una nueva estación.

- Seguir avanzando.
- Tomar el desvío de la derecha, si existe.
- Tomar el desvío de la izquierda, si existe.
- Detenerse.

Combinando estas cuatro acciones el robot es capaz de desplazarse desde cualquier estación del circuito a otra. La función sigue la siguiente estructura:

1. Se declaran las variables *flag_avance*, *flag_izquierda*, *flag_derecha* y *flag_llegada* y se igualan a 0. Son variables bandera que se activaran en determinadas condiciones.
2. Usaremos el mecanismo de control *switch case*, que permite que según el valor de una determinada variable o expresión cambie el flujo de control de la ejecución del programa. Dependiendo del valor de las variables *dest* y *estacion* se activarán diferentes banderas. Pongamos el ejemplo de que queremos llegar a la estación de *Medicina General* (color lima), que está asociada al identificador numérico 2:

Tabla 4-2. Acciones por estación cuando el destino es la estación 2.

Valor `dest`	Valor `estacion`	Nombre	Acción
2	1	Entrada	`flag_izquierda = 1`
	2	Medicina General	`flag_llegada = 1`
	3	Oftalmología	`flag_avance = 1`
	4	Neurología	`flag_izquierda = 1`
	5	Dermatología	`flag_avance = 1`
	6	Pediatría	`flag_derecha = 1`
	7	Alergología	`flag_avance = 1`
	8	Urgencias	`flag_avance = 1`
	-	-	`flag_avance = 1`

3. De todas estas acciones, solo ocurrirá una cada vez que se invoque a la función (que es en cada iteración del bucle), por lo que solo una bandera se activará. Según que bandera se active se ejecuta una de las siguientes acciones:
    a. *flag_llegada*: Significa que el robot ha llegado a la estación de destino, por lo que se detiene. Quedará en reposo hasta que se le asigne un nuevo destino. Se invoca a la función **`reposo`**.
    b. *flag_avance:* El robot sigue avanzando hacia delante, siguiendo el camino marcado por la línea negra, ya sea porque es el camino que tomar o porque no hay bifurcaciones. Es la acción que se realizará cuando no se detecte ninguna estación (*dest* es 0). Se invoca a la función **`avance1`** o **`avance2`**, dependiendo del valor de *vel.*
    c. *flag_derecha:* El robot, que se encuentra ante una bifurcación, toma el desvío de la derecha. Se invoca a la función **`derecha1`** o **`derecha2`**, dependiendo del valor de *vel*.
    d. *flag_izquierda:* El robot, que se encuentra ante una bifurcación, toma el desvío de la izquierda. Se invoca a la función **`izquierda1`** o **`izquierda2`**, dependiendo del valor de *C*
4. Cuando el robot llegue a su destino (caso 3a) se pondrá la variable *dest* a 0.

Esta función solo se invocará cuando la variable *dest* no sea 0, es decir, que el robot tenga asignado un destino y no se encuentre sobre él.

4.3 Interrupciones

Aunque parece que ya tenemos definido el comportamiento de nuestro robot, nos falta lo más importante. El robot se desplaza por el circuito siguiendo la línea negra que lo recorre. Esto es posible porque el robot corrige su movimiento en el momento que detecta que se ha salido de la línea, pausando todas las demás tareas hasta que vuelve a detectarla.

Este comportamiento de ejecuta en la **rutina de interrupción** del microcontrolador, un programa invocado por una solicitud de interrupción de un dispositivo hardware. Nuestra función principal cuenta con 4 interrupciones:

- *SCIA_RX_isr* es la rutina de interrupción asociada a la recepción de datos por el dispositivo HC-06. Fue detallada en el apartado 4.1.4, por lo que no nos detendremos de nuevo a explicarla.
- *obstaculo* está asociada al sensor de proximidad y es la encargada de activar el temporizador que mide el tiempo que tarda el eco en volver al sensor. Su funcionamiento se explicó junto al del sensor HC-SR04 en el apartado 3.5.2.
- *cpu_timer0_isr* es la interrupción del Timer0 de la CPU. Se activa cuando la cuenta atrás del Timer0, previamente configurado, llega a 0. La usamos para la función *delay* que retrasa en microsegundos la siguiente línea de código.
- *rectificar* es probablemente la rutina de interrupción más importante y la que controla que el robot siga la línea negra en todo momento. Está asociada a los GPIOs 7 y 11, que están conectados a los sensores IR.

Los registros de los GPIOs asociados a los sensores IR adquieren el valor 1 cuando estos detectan la línea negra y 0 cuando detectan el fondo blanco. Por lo tanto, a nosotros nos interesa que se genere una señal de interrupción cuando ocurran flancos de bajada en los valores de estos registros.

Para ello podemos dar el valor 0 al registro *XINTxCR.bit.POLARITY*, que controla cuando se generan las interrupciones. Esto se realiza tanto para la *XINT1* como para la *XINT2* en la función de configuración de los GPIOs.

Aunque la señal de interrupción se puede generar en cualquier momento, *rectificar* no se ejecutará cuando el robot esté tomando desvíos o giros provocados por la función *recorrido.* Estos giros son necesarios para seguir el camino más corto hasta el destino y están programados para durar apenas 10ms. Digamos que son movimientos forzados y que, en gran parte de los casos, causarán que alguno de los sensores IR salga de la línea negra.

La rutina de interrupción *rectificar* sigue el siguiente esquema:

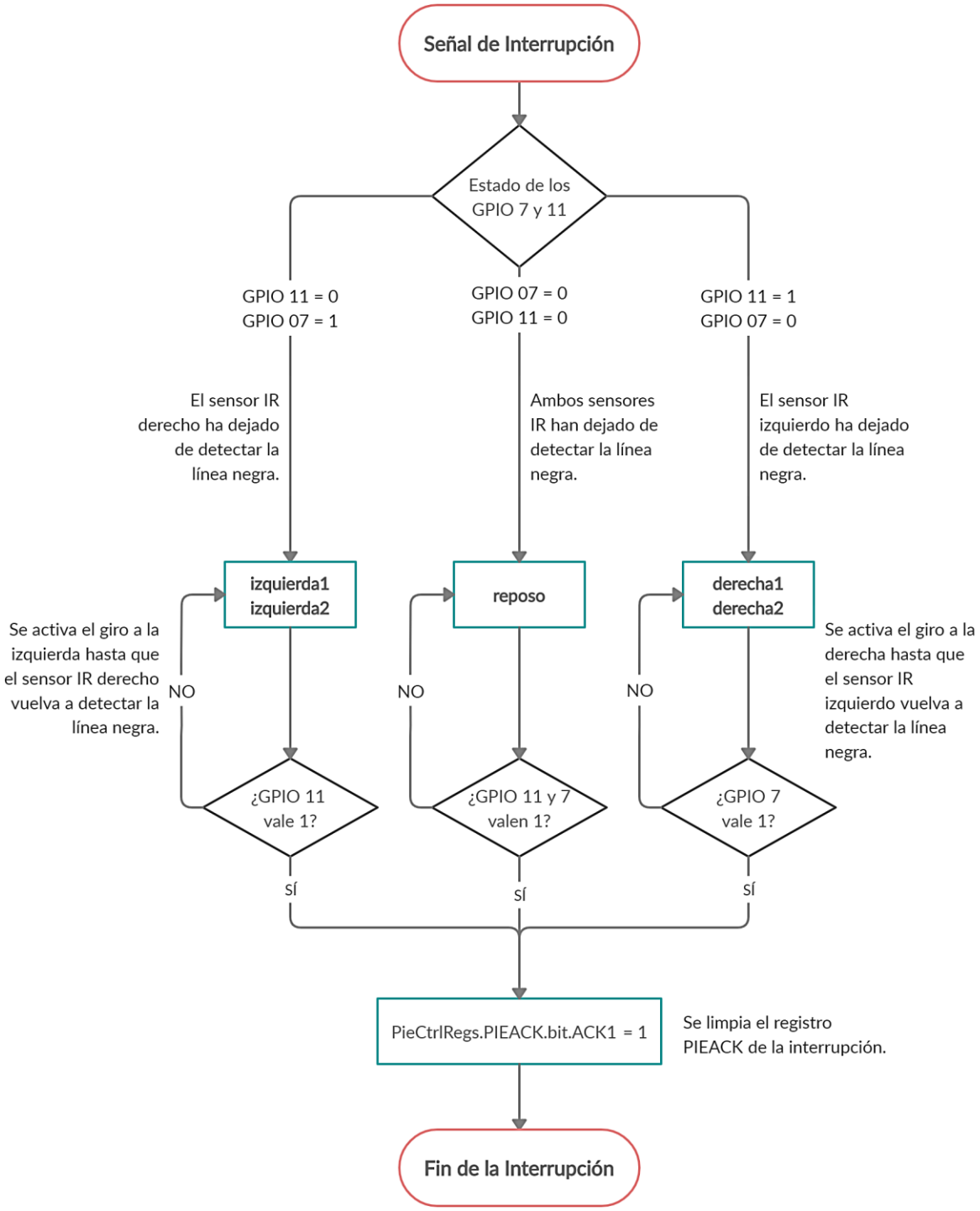

Figura 4-4. Esquema de funcionamiento de la interrupción *rectificar*.

La interrupción analiza el estado de los pines GPIO para actuar en consecuencia. Una vez que el robot vuelve a estar centrado en la línea finaliza la interrupción. Si por algún motivo, ambos sensores IR dejaran de detectar la línea negra el robot quedaría en reposo y tendríamos que colocarlo manualmente de nuevo sobre la línea.

5 RESULTADOS Y CONCLUSIONES

Nuestro robot está montado y programado para cumplir con su función. En este capítulo analizaremos los resultados obtenido y sus posibles márgenes de mejora, así como ampliaciones adicionales que podrían ser de utilidad.

5.1 Pruebas del Robot

El error más común que comete nuestro robot es confundir los colores de las estaciones. En concreto, la estación 8 *Alergología*, que corresponde al color verde oscuro. El sensor no es capaz de distinguir claramente entre ese color y el blanco del fondo. Como comentamos en el capítulo 3.4.3, la tonalidad de los colores registrados es más oscura que la real.

Esta confusión causa paradas repentinas a lo largo del circuito cuando el destino es la estación de *Alergología*, ya que el robot cree que está sobre la estación cuando realmente es el fondo. En un principio se intentó ajustar los valores RGB para acotar aún más la franja válida para el verde, pero el error persistía. Ajustar la ganancia del sensor para aumentar la sensibilidad estaba fuera de la cuestión, pues provoca lecturas erráticas y una alta variación en los valores.

La solución ha sido sustituir el color verde oscuro por el rojo. Ahora el sensor parece capaz de distinguirlo perfectamente de los demás y, más importante, del fondo.

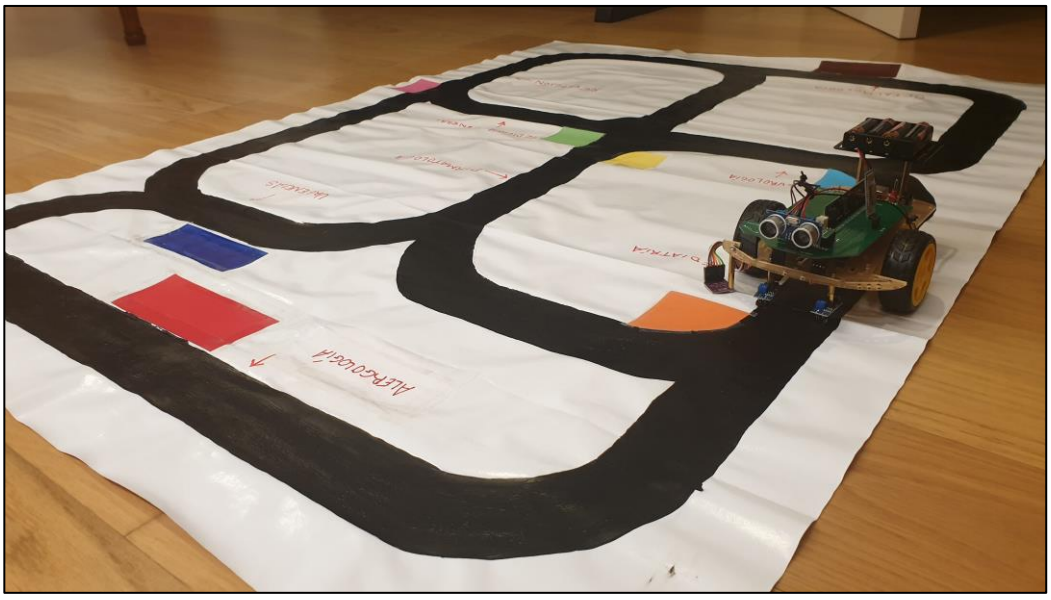

Figura 5-1. Circuito con estación de *Alergología* roja.

El color de la estación 2 *Medicina General*, lima, también tendía a confundir al sensor. A menudo el robot pasaba de largo sin ni siquiera detectar dicho color. El problema se ha solucionado ajustando los valores RGB.

En el ámbito del seguimiento de líneas, el funcionamiento es prácticamente perfecto, tanto a la velocidad normal como a la aumentada. Los sensores IR responden de inmediato, permitiendo al robot corregir su posición rápidamente en el caso de que se salga de la línea.

Es posible que aumentando la anchura de los carriles del circuito se mejorara un poco la maniobrabilidad del robot, ya que a veces presenta giros o movimientos bruscos al estar ambos sensores continuamente al borde del carril. Sin embargo, esto no presenta un problema grave y el robot sigue desplazándose por el circuito perfectamente.

En un principio se planteó la posibilidad de añadir una velocidad adicional. Las pruebas concluyeron que el robot era todavía capaz de seguir la línea adecuadamente, pero el sensor RGB no era lo suficientemente rápido como para detectar las estaciones, por lo que fue descartada. Se pueden encontrar restos en el código de esta tercera velocidad, que no fue totalmente eliminada en caso de encontrar una solución.

En cuanto al sensor de proximidad, su funcionamiento es también excelente. La detección de obstáculos se produce de manera eficaz y casi siempre a la distancia especificada (10 cm). A la velocidad superior el sensor detecta el obstáculo con un poco menos de margen, pero el robot se detiene sin llegar a chocar con él. No se hicieron pruebas a la tercera velocidad, pero creemos que la detección de los obstáculos hubiera seguido siendo posible, aunque quizás de manera un poco justa.

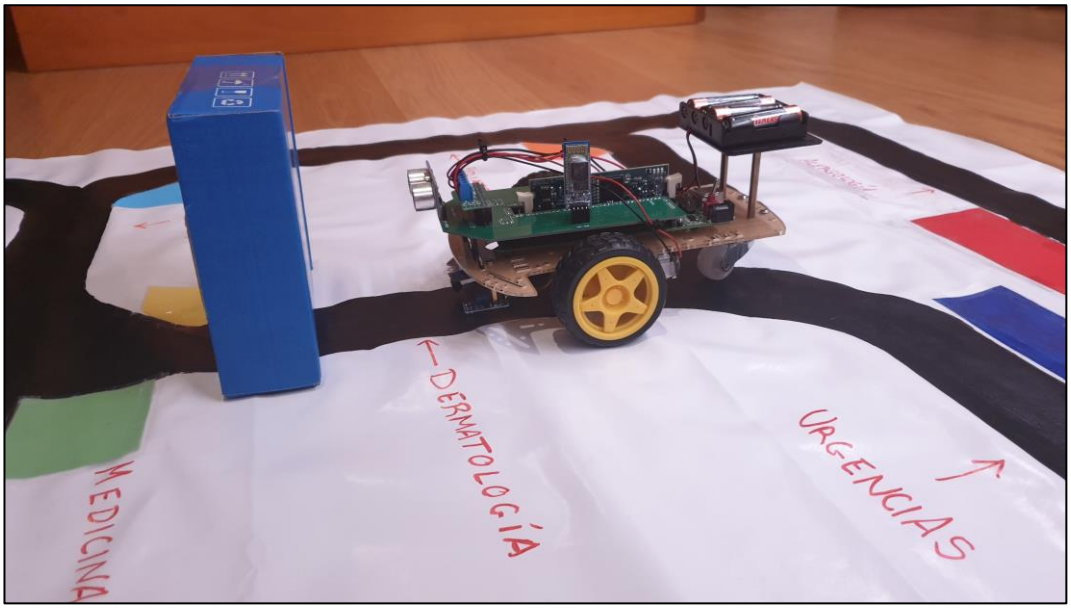

Figura 5-2. El robot obstaculizado.

Se ha detectado que, en muy escasas ocasiones, cuando el robot va a velocidad normal, los motores pueden quedarse pillados al no tener la suficiente fuerza como para mover la rueda. Esto por lo general empieza a ser más frecuente cuando las pilas pierden energía. En caso de ocurra, un simple toque con el dedo resuelve el problema y el motor vuelve a funcionar con normalidad.

Afortunadamente, parece que la programación en *flash* no ha afectado al rendimiento ni potencia del robot. En otros proyectos nos hemos encontrado con problemas a la hora de cargar el problema en la memoria *flash*, ya sea por problemas de memoria o de potencia. Éste no es el caso en nuestro proyecto.

5.2 Conclusiones

Parece que uno de los grandes limitadores de nuestro proyecto es el sensor RGB TCS34725. Su funcionamiento, aunque adecuado, dista mucho de ser perfecto. Se ha intentado configurarlo de diferentes maneras y ninguna nos ha convencido del todo, siendo la detección de los colores muy difusa y, a menudo, errónea. Aunque es suficiente como para distinguir algunas tonalidades de otras, dudo de su capacidad a la hora de integrarlo en proyectos donde identificar una mayor gama de colores sea fundamental.

Por otra parte, el sensor de proximidad HC-SR04 nos ha sorprendido con su precisión y rapidez a la hora de medir distancias. Los sensores IR también funcionan perfectamente y hacen del guiado de un robot seguidor de líneas una tarea sencilla y cómoda.

En cuanto a los modos de comunicación, se ha llegado a la conclusión de que la interfaz táctil de la app *Bluetooth Electronics* presenta un control más intuitivo e inmediato. Sin embargo, el control por voz también funciona muy bien y, aunque el robot tarde unos instantes más en responder, podría ser el preferido por algunos usuarios. De cualquier manera, ambos se integran perfectamente en el resultado final.

Aunque el robot pueda parecer sencillo y su comportamiento diste de ser complejo, el proyecto ha supuesto una cantidad de trabajo considerable, ya que se ha tenido que investigar el funcionamiento de todos los componentes (información que no abunda en la red), probar sus distintas configuraciones y calibrarlos. Además, se ha logrado un diseño bastante compacto y ligero, en parte gracias al PCB que hemos diseñado nosotros mismos.

5.3 Posibles Ampliaciones

Las posibilidades de ampliación para nuestro robot son prácticamente infinitas. Podríamos mejorar aún más su funcionamiento o añadirle piezas de hardware que habiliten nuevas funcionalidades.

Para empezar, el trabajo de los motores no es perfecto. Un mayor ajuste del periférico ePWM, unido a unos motores de mayor calidad, permitirían un movimiento más suave y preciso del robot. Cuando se produce un giro, una de las ruedas se para en seco, provocando deslizamiento. Esto podría arreglarse haciéndola girar levemente en sentido contrario, pero nuestros motores carecen de esa capacidad a velocidades tan pequeñas.

Se podrían añadir un par de ruedas más para mejorar la maniobrabilidad. Requeriríamos de un nuevo *driver* para los dos motores adicionales y un nuevo ajuste del PWM. Podríamos incluso sustituir los motores de corriente continua por servomotores, que permitirían giros mucho más fluidos.

Figura 5-3. Servomotor.

Otra forma para detectar mejor las bifurcaciones y obtener movimientos más fluidos sería añadiendo múltiples sensores IR, colocados de forma que sean capaces de detectar los cambios del robot de manera más suave y precisa.

Esta configuración mejoraría el giro del robot en las curvas de nuestro circuito, que son todas muy cerradas y de 90º. Mientras se detecte la línea funcionarán los tres sensores centrales, pero si hay una bifurcación brusca los sensores externos se activarán, y el control podrá efectuar un movimiento para adaptarse al tipo de pista.

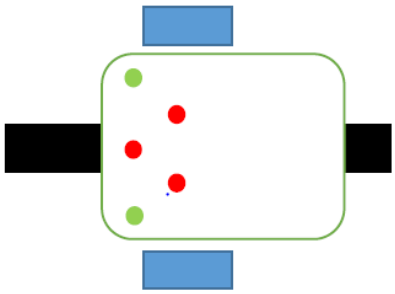

Figura 5-4. Configuración de robot seguidor de líneas con 5 sensores IR.

Otra funcionalidad que se tenía en mente inicialmente era un sistema para que el robot monitorease a la persona que estuviera siendo guiada. Es decir, el robot comprobaba periódicamente que la persona supuestamente le sigue hasta el destino no se quedará atrás. Al principio, se planteó el uso del sensor de proximidad HC-SR04, pero finalmente el mismo se recicló para la detección de obstáculos, ya que la medida de la distancia no es demasiado precisa si no se realiza con superficies planas.

En un uso real, el robot, de tamaño mayor, podría tener una especie de barra a la que agarrarse, asegurando la presencia de la persona en todo momento. Esto podría realizarse mediante un sensor táctil, o simplemente un sensor de temperatura.

Se llegó a adquirir un componente que habilitaría esta función: el sensor táctil capacitivo TTP223B. El funcionamiento sería sencillo, se conecta este sensor a la parte trasera del robot y, al ser tocado, se avisa de que la persona está presente.

Figura 5-5. Sensor táctil capacitivo TTP223B.

Sin embargo, la idea fue desechada por el inconveniente de mantener dicho sensor activado con la reducida escala de nuestro robot.

Por último, se podría implementar un chip de sonido/altavoz que permita al robot avisar cuando llegue a su destino o si ocurre algún improvisto. Para ello, podría usarse un módulo de tarjeta SD (para guardar los archivos de audio) y un minialtavoz integrado en el robot. Mediante el uso del PWM se conseguiría la reproducción de sonido. La idea encaja con la propuesta de nuestro robot, ya que encontrarse con usuarios que agradecieran señales auditivas no sería extraña en una zona de hospitales.

REFERENCIAS

[1] Wikipedia, *Automated Vehicle Robtos,*
Disponible en: https://en.wikipedia.org/wiki/Automated_guided_vehicle.

[2] Revista de Robots, *Vehículos AGV y AIV, los vehículos de guiado automático inteligentes,*
Disponible en: https://revistaderobots.com/robots-y-robotica/

[3] Lodamaster, *AUTOMATED GUIDED VEHICLES,*
Disponible en: https://www.lodamaster.com/en/solutions/automated-guided-vehicles

[4] Wikipedia, *Guidance System,*
Disponible en: https://en.wikipedia.org/wiki/Guidance_system

[5] Atria Innovation, *AGVs, ¿Vehículos de Guiado Automático todavía no sabes lo que son?,*
Disponible en: https://www.atriainnovation.com/

[6] Wikipedia, *Robot seguidor de líneas,*
Disponible en: https://es.wikipedia.org/wiki/Robot_seguidor_de_l%C3%ADnea

[7] Instructables, *How to Make Line Follower Robot Using Arduino,*
Disponible en: https://www.instructables.com/id/Line-Follower-Robot-Using-Arduino-2/

[8] Wikipedia, *Texas Instruments,*
Disponible en: https://en.wikipedia.org/wiki/Texas_Instruments

[9] Robot Platform, *Wheeled Robots,*
Disponible en: http://www.robotplatform.com/knowledge/Classification_of_Robots

[10] Sven Böttcher, *Principles of robot locomotion*,
Disponible en: http://www2.cs.siu.edu/~hexmoor/classes/CS404-S09/RobotLocomotion.pdf

[11] Electric Bricks, *Rueda loca,*
Disponible en: http://blog.electricbricks.com/2010/05/caster-wheel/

[12] Wikipedia, *Puente H (electrónica),*
Disponible en: https://es.wikipedia.org/wiki/Puente_H_(electr%C3%B3nica)

[13] Wikipedia, *Sensor infrarrojo,*
Disponible en: https://es.wikipedia.org/wiki/Sensor_infrarrojo

[14] Prometec, *Los Sensores Infrarrojos,*
Disponible en: https://www.prometec.net/siguelineas-ir/

[15] DYOR: Do Your Own Robot, *Módulo TCRT5000 (seguilíneas),*
Disponible en: http://dyor.roboticafacil.es/seguilineas/

[16] Luis Llamas, *DETECTOR DE LÍNEAS CON ARDUINO Y SENSOR TCRT5000L,*
Disponible en: https://www.luisllamas.es/arduino-detector-lineas-tcrt5000l/

[17] Adafruit, *RGB Color Sensor with IR filter and White LED - TCS34725,*
Disponible en: https://www.adafruit.com/product/1334

[18] Luis Llamas, *MEDIR VALORES RGB CON ARDUINO Y SENSOR DE COLOR TCS34725,*
Disponible en: https://www.luisllamas.es/arduino-sensor-color-rgb-tcs34725/

[19] Hardwarelibre, *HC-SR04: todo sobre el sensor de ultrasonidos,*
Disponible en: https://www.hwlibre.com/hc-sr04/

[20] Luis Llamas, *CONECTAR ARDUINO POR BLUETOOTH CON LOS MÓDULOS HC-05 Ó HC-06,*
Disponible en: https://www.luisllamas.es/conectar-arduino-por-bluetooth-con-los-modulos-hc-05-o-hc-06/

[21] Prometec, *Módulo Bluetooth HC-06,*
Disponible en: https://www.prometec.net/bt-hc06/

[22] CDMX Electrónica, *Módulo Bluetooth HC-06,*
Disponible en: https://cdmxelectronica.com/producto/modulo-bluetooth-hc-06/

[23] Altium Designer, *Cómo crear un esquemático en Altium Designer: La jornada de mil PCBs,*
Disponible en: https://resources.altium.com/es/p/schematic-tutorial-altium-designer-journey-thousand-pcbs

[24] Altium Designer, *Interactive Routing,*
Disponible en: https://www.altium.com/documentation/altium-designer/interactive-routing-ad?version=18.1

[25] Manuel J. Bellido Díaz, *Diseño y Fabricación de PCBs,*
Disponible en: https://www.dte.us.es/docencia/etsii/gii-ic/laboratorio-de-desarrollo-hardware/temas/Tema3DPCB

[26] Manuel J. Bellido Díaz, *Normas Básicas y Recomendaciones en el Diseño de PCBs,*
Disponible en: https://www.dte.us.es/docencia/etsii/gii-ic/laboratorio-de-desarrollo-hardware/temas/Tema5NormasPCB

[27] TAOS Texas Advanced Optoelectronic Solutions, *TCS3472 Color light-to-digital converter with IR filter.*
Disponible en: https://cdn-shop.adafruit.com/datasheets/TCS34725.pdf

[28] Sparkfun, *Ultrasonic Ranging Module HC - SR04,*
Disponible en: https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf

[29] Pololu Robotics & Electronics, *DRV8835 Dual Motor Driver Kit for Raspberry Pi*
Disponible en: https://www.pololu.com/product/2753

[30] JLCPCB, *PCB Prototype and PCB Fabrication*
Disponible en: https://jlcpcb.com/

[31] Texas Instruments, *TMS320x2833x, 2823x System Control and Interrupts,*
Disponible en: https://www.ti.com/product/TMS320F28335

[32] Texas Instruments, *TMS320F2833x, TMS320F2823x Digital Signal Controllers (DSCs)*
Disponible en: https://www.ti.com/product/TMS320F28335

[33] Texas Instruments, TMS320x2833x, 2823x Inter-Integrated Circuit (I2C)
Disponible en: https://www.ti.com/product/TMS320F28335

ANEXOS

Los anexos de este proyecto recogen el código íntegro utilizado para programar el robot, con comentarios explicativos de algunas líneas.

I. Funciones de Configuración

a. PWM

```
void Setup_ePWM(void)
{
    /********** CONGIFURACIÓN PWM **********/

    // Configuración del módulo Time-Base
    EPwm2Regs.TBCTL.bit.CLKDIV = 0;         // Pre-escalador CLKDIV = 1
    EPwm2Regs.TBCTL.bit.HSPCLKDIV = 1;      // Pre-escalador HSPCLKDIV = 2
    EPwm2Regs.TBCTL.bit.CTRMODE = 2;        // Modo up - down
    EPwm2Regs.TBCTL.bit.PRDLD = 0;          // Shadow activado
    EPwm2Regs.TBCTL.bit.SYNCOSEL = 0;       // Sincronización deshabilitada
    EPwm2Regs.TBPRD = 37500;                // Periodo para 1KHz de fPWM

    EPwm6Regs.TBCTL.bit.CLKDIV = 0;         // Pre-escalador CLKDIV = 1
    EPwm6Regs.TBCTL.bit.HSPCLKDIV = 1;      // Pre-escalador HSPCLKDIV = 2
    EPwm6Regs.TBCTL.bit.CTRMODE = 2;        // Modo up - down
    EPwm6Regs.TBCTL.bit.PRDLD = 0;          // Shadow activado
    EPwm6Regs.TBCTL.bit.SYNCOSEL = 0;       // Sincronización deshabilitada
    EPwm6Regs.TBPRD = 37500;                // Periodo para 1KHz de fPWM

    // Configuración del módulo Compare-Counter
    EPwm2Regs.CMPCTL.bit.SHDWAMODE = 0;     // shahow activado para CMPA
    EPwm2Regs.CMPCTL.bit.SHDWBMODE = 0;     // shahow activado para CMPB
    EPwm2Regs.CMPCTL.bit.LOADAMODE = 0;     // shadow de CMPA cargado en TBCTR = 0
    EPwm2Regs.CMPCTL.bit.LOADBMODE = 0;     // shadow de CMPB cargado en TBCTR = 0

    EPwm6Regs.CMPCTL.bit.SHDWAMODE = 0;     // shahow activado para CMPA
    EPwm6Regs.CMPCTL.bit.SHDWBMODE = 0;     // shahow activado para CMPB
    EPwm6Regs.CMPCTL.bit.LOADAMODE = 0;     // shadow de CMPA cargado en TBCTR = 0
    EPwm6Regs.CMPCTL.bit.LOADBMODE = 0;     // shadow de CMPB cargado en TBCTR = 0

    // Configuración del módulo Action Qualifier
    EPwm2Regs.AQCTLA.all = 0;
    EPwm2Regs.AQCTLA.bit.CAD = 2;           // EPWM1A a low en CMPA down
    EPwm2Regs.AQCTLA.bit.CAU = 1;           // EPWM1A a high en CMPA up
    EPwm2Regs.AQCTLB.all = 0;
    EPwm2Regs.AQCTLB.bit.CBD = 2;           // EPWM1B a low en CMPB down
    EPwm2Regs.AQCTLB.bit.CBU = 1;           // EPWM1B a high CMPB up

    EPwm6Regs.AQCTLA.all = 0;
    EPwm6Regs.AQCTLA.bit.CAD = 2;           // EPWM2A a low en CMPA down
    EPwm6Regs.AQCTLA.bit.CAU = 1;           // EPWM2A a high en CMPA up
    EPwm6Regs.AQCTLB.all = 0;
    EPwm6Regs.AQCTLB.bit.CBD = 2;           // EPWM2B a low en CMPB down
    EPwm6Regs.AQCTLB.bit.CBU = 1;           // EPWM2B a high CMPB up

    // Configuración del módulo Dead Band
    EPwm2Regs.DBCTL.bit.OUT_MODE = 0;       // Módulo activado
    EPwm6Regs.DBCTL.bit.OUT_MODE = 0;       // Módulo activado

    // Configuración del módulo PWM-Chopper
    EPwm2Regs.PCCTL.bit.CHPEN = 0;          // Módulo descativado
```

```
    EPwm6Regs.PCCTL.bit.CHPEN = 0;          // Módulo descativado

    // Configuarción del módulo Trip Zone
    EPwm2Regs.TZCTL.bit.TZA = 3;            // No hacer nada
    EPwm2Regs.TZCTL.bit.TZB = 3;            // No hacer nada
    EPwm2Regs.TZEINT.all = 0;               // interrupción deshabilitada

    EPwm6Regs.TZCTL.bit.TZA = 3;            // No hacer nada
    EPwm6Regs.TZCTL.bit.TZB = 3;            // No hacer nada
    EPwm6Regs.TZEINT.all = 0;               // interrupción deshabilitada

    // Configuración del módulo Event Trigger
    EPwm2Regs.ETSEL.bit.SOCAEN = 0;         // Generación de SOCA deshabilitado
    EPwm2Regs.ETSEL.bit.SOCBEN = 0;         // Generación de SOCB deshabilitado
    EPwm2Regs.ETSEL.bit.INTEN = 0;          // Señal de interrupción deshabilitada

    EPwm6Regs.ETSEL.bit.SOCAEN = 0;         // Generación de SOCA deshabilitado
    EPwm6Regs.ETSEL.bit.SOCBEN = 0;         // Generación de SOCB deshabilitado
    EPwm6Regs.ETSEL.bit.INTEN = 0;          // Señal de interrupción deshabilitada
}
```

b. GPIO

```
void Gpio_select(void)
{
    /********** CONFIGURACIÓN GPIOs **********/

    EALLOW;

    // GPIOs para POLOLU DUAL MOTOR DRIVER KIT
    GpioCtrlRegs.GPAMUX1.bit.GPIO2 = 1;     // GPIO2 como EPWM2A
    GpioCtrlRegs.GPAMUX1.bit.GPIO10 = 1;    // GPIO10 como EPWM6A

    GpioCtrlRegs.GPAPUD.bit.GPIO6 = 0;
    GpioCtrlRegs.GPAMUX1.bit.GPIO6 = 0;     // GPIO6 como E/S
    GpioCtrlRegs.GPADIR.bit.GPIO6 = 1;      // GPIO6 como salida

    GpioCtrlRegs.GPAPUD.bit.GPIO15 = 0;
    GpioCtrlRegs.GPAMUX1.bit.GPIO15 = 0;    // GPIO15 como E/S
    GpioCtrlRegs.GPADIR.bit.GPIO15 = 1;     // GPIO15 como salida


    // GPIOs para TCRT5000 (SENSORES IR)
    GpioCtrlRegs.GPAPUD.bit.GPIO11 = 0;         // GPIO
    GpioCtrlRegs.GPAMUX1.bit.GPIO11 = 0;        // GPIO11 como E/S
    GpioCtrlRegs.GPADIR.bit.GPIO11 = 0;         // GPIO11 como entrada
    GpioIntRegs.GPIOXINT1SEL.bit.GPIOSEL = 11;  // Activamos interrupción para GPIO11

    GpioCtrlRegs.GPAPUD.bit.GPIO7 = 0;          // GPIO
    GpioCtrlRegs.GPAMUX1.bit.GPIO7 = 0;         // GPIO7 como E/S
    GpioCtrlRegs.GPADIR.bit.GPIO7 = 0;          // GPIO7 como entrada
    GpioIntRegs.GPIOXINT2SEL.bit.GPIOSEL = 7;   // Activamos interrupción para GPIO7


    // GPIOs para HC-SR04 (SENSOR DE PROXIMIDAD)
    GpioCtrlRegs.GPAMUX1.bit.GPIO13 = 0;        // GPIO13 como E/S
    GpioCtrlRegs.GPADIR.bit.GPIO13= 1;          // GPIO13 como salida (TRIGGER)

    GpioCtrlRegs.GPAMUX1.bit.GPIO14 = 0;        // GPIO14 como E/S
    GpioCtrlRegs.GPADIR.bit.GPIO14= 0;          // GPIO14 como entrada (ECHO)
    GpioIntRegs.GPIOXNMISEL.bit.GPIOSEL = 14;   // Activamos interrupción para GPIO14

    // GPIOs para HC-05 (MÓDULO BLUETOOTH)
    GpioCtrlRegs.GPAMUX2.bit.GPIO28 = 1;   // GPIO28 como SCIRXDA
```

```
    // GPIOs para TCS34725 (SENSOR RGB)
    GpioCtrlRegs.GPBPUD.bit.GPIO32 = 0;   // Activamos pull-up en GPIO32
    GpioCtrlRegs.GPBPUD.bit.GPIO33 = 0;   // Activamos pull-up en GPIO33
    GpioCtrlRegs.GPBQSEL1.bit.GPIO32 = 3; // asynch input
    GpioCtrlRegs.GPBQSEL1.bit.GPIO32 = 3; // asynch input
    GpioCtrlRegs.GPBMUX1.bit.GPIO32 = 1;  // GPIO32 como SDAA (Protocolo I2C)
    GpioCtrlRegs.GPBMUX1.bit.GPIO33 = 1;  // GPIO33 como SCLA (Protocolo I2C)

    EDIS;

    // INTERRUPCIONES
    XIntruptRegs.XINT1CR.bit.ENABLE = 1;
    XIntruptRegs.XINT1CR.bit.POLARITY = 0;
    XIntruptRegs.XINT2CR.bit.ENABLE = 1;
    XIntruptRegs.XINT2CR.bit.POLARITY = 0;
    XIntruptRegs.XNMICR.bit.SELECT = 1;
    XIntruptRegs.XNMICR.bit.ENABLE = 1;
    XIntruptRegs.XNMICR.bit.POLARITY = 3;
}
```

c. SCI

```
void SCIA_init()
{
    /********** CONFIGURACIÓN COMUNICACIÓN SCI **********/

    SciaRegs.SCICCR.all = 0;
    SciaRegs.SCICCR.bit.SCICHAR = 7;        // 8 bits de datos
    SciaRegs.SCICCR.bit.ADDRIDLE_MODE = 0;  // Sin bit de dirección
    SciaRegs.SCICCR.bit.PARITYENA = 0;      // Sin bit de paridad
    SciaRegs.SCICCR.bit.STOPBITS = 0;       // 1 bit de stop


    SciaRegs.SCICTL1.all = 0;               // Se pone a cero para hacer el RESET del SCIA
    SciaRegs.SCICTL1.bit.RXENA = 1;         // Se habilita la línea de recepción

    // Configuración de velocidad de comunicación a 9600 baudios
    // SYSCLOCKOUT = 150MHz; LSPCLK = 37.5 MHz
    // BRR = (LSPCLK / (9600 x 8)) -1
    // BRR = 487 gives 9605 Baud
    SciaRegs.SCIHBAUD = 487 >> 8;           // Highbyte
    SciaRegs.SCILBAUD = 487 & 0x00FF;       // Lowbyte

    SciaRegs.SCICTL2.bit.RXBKINTENA = 1;    // Se habilita la interrupción por recepción

    SciaRegs.SCICTL1.bit.SWRESET = 1;       // Se libera el SCI del RESET para poder operar
}
```

d. I²C

```
void I2CA_Init(void)
{
    /********** CONFIGURACIÓN PROTOCOLO I2C ***********/

    I2caRegs.I2CMDR.all = 0x0000;           // Reiniciamos el módulo I2C
    I2caRegs.I2CSAR = TCS24725_SLAVE;       // Dirección del registro del módulo esclavo
    I2caRegs.I2CPSC.all = 14;               // Reloj interno del módulo I2C = SYSCLK/(PSC +1)
                                            // = 10 MHz

    I2caRegs.I2CCLKL = 95;                  // Tmaster = (PSC +1)[ICCL + 5 + ICCH + 5] /
//150MHz
    I2caRegs.I2CCLKH = 95;                  // Tmaster =  10 [ICCL + ICCH + 10] / 150 MHz

    I2caRegs.I2CMDR.bit.IRS = 1;            // Activamos el módulo I2C

    while (I2caRegs.I2CSTR.bit.BB == 1);    // Aseguramos que el bus de datos esté libre
    I2caRegs.I2CCNT = 2;                    // Envía 2 bytes (dirección + datos)
```

```
    I2caRegs.I2CMDR.all = 0x6E20;           // Modo en el que actúa el I2C
    /*
                Bit15 = 0;  sin NACK en modo receptor
                Bit14 = 1;  FREE  sigue operando al encontrar un punto de interrupción
                Bit13 = 1;  STT   genera señal de START
                Bit12 = 0;  reservado
                Bit11 = 1;  STP   genera señal de STOP
                Bit10 = 1;  MST   modo maestro
                Bit9  = 1;  TRX   maestro como transmisor
                Bit8  = 0;  XA    registro de 7 bits
                Bit7  = 0;  RM    sin repetición, I2CCNT determina el número de bytes
                Bit6  = 0;  DLB   sin loopback
                Bit5  = 1;  IRS   módulo I2C activado
                Bit4  = 0;  STB   sin byte de START aislado
                Bit3  = 0;  FDF   sin formato de datos libre
                Bit2-0: 0;  BC    8 bits por byte de datos
    */

    I2caRegs.I2CDXR = TCS34725_COMMAND_BIT | TCS34725_ATIME;    // Tiempo de integración
    while(I2caRegs.I2CSTR.bit.XRDY == 0);                       // Sale el primer byte
    I2caRegs.I2CDXR =0xF6;                                      // Fijamos el tiempo de
//integración a 2.4ms (DATASHEET)
    while(I2caRegs.I2CSTR.bit.SCD == 0);                        // Esperamos la señal de STOP
    I2caRegs.I2CSTR.bit.SCD = 1;                                // Reiniciamos la señal STOP

    delay(100);
    I2caRegs.I2CMDR.all = 0x6E20;
    while(I2caRegs.I2CSTR.bit.XRDY == 0);
    I2caRegs.I2CDXR = TCS34725_COMMAND_BIT | TCS34725_CONTROL;  // Control de ganancia
    while(I2caRegs.I2CSTR.bit.XRDY == 0);                       // Sale el primer byte
    I2caRegs.I2CDXR =0x01;                                      // Establecemos la ganancia a
//x60 (DATASHEET)
    while(I2caRegs.I2CSTR.bit.SCD == 0);                        // Esperamos la señal de STOP
    I2caRegs.I2CSTR.bit.SCD = 1;                                // Reiniciamos la señal STOP

    delay(100);
    I2caRegs.I2CMDR.all = 0x6E20;
    while(I2caRegs.I2CSTR.bit.XRDY == 0);
    I2caRegs.I2CDXR = TCS34725_COMMAND_BIT | TCS34725_ENABLE;   // Activación del TCS34725
    while(I2caRegs.I2CSTR.bit.XRDY == 0);                       // Sale el primer byte
    I2caRegs.I2CDXR =TCS34725_ENABLE_PON | TCS34725_ENABLE_AEN; // Encendemos el TCS34725 y
//activamos el modo RGBC (DATASHEET)
    while(I2caRegs.I2CSTR.bit.SCD == 0);                        // Esperamos la señal de STOP
    I2caRegs.I2CSTR.bit.SCD = 1;                                // Reiniciamos la señal STOP

}
```

II. Función Principal

```
#include "coche.h"

// Prototipado de funciones externas
extern void InitSysCtrl(void);
extern void InitPieCtrl(void);
extern void InitPieVectTable(void);
extern void InitCpuTimers(void);
extern void ConfigCpuTimer(struct CPUTIMER_VARS *, float, float);

// Prototipado de funciones de interrupción
interrupt void SCIA_RX_isr(void);
interrupt void rectificar(void);
interrupt void obstaculo(void);
interrupt void cpu_timer0_isr(void);

// Declaración de variables globales

char buffer[100];

int flag_rx0=0;
int flag_rx=0;
int i=0;
int dest=0;
int estacion;
int vel=1;
int modo=0;
float rojo, azul, verde; // Parámetros RGB
int flag_echo=0;
float distancia = 0.0;
int nointerrumpir=0;

void main(void){

    InitSysCtrl();      // Inidicalización del sistema (DSP2833x_SysCtrl.c):
    Gpio_select();      // Configuración GPIOs
    Setup_ePWM();       // Configuración módulo ePWM
    SCIA_init();        // Inicialización módulo SCIA
    InitPieCtrl();      // Inicialización de la PIE (DSP2833x_PieCtrl.c)
    InitPieVectTable(); // Inicialización de la tabla PIE (DSP2833x_PieVect.c)

    // Modificación de la PIE para direccionar las rutinas de interrupción

    EALLOW;
    PieVectTable.SCIRXINTA = &SCIA_RX_isr;  // Lectura Bluetooth
    PieVectTable.XINT1 = &rectificar;       // Sensor IR derecho
    PieVectTable.XINT2 = &rectificar;       // Sensor IR izquierdo
    PieVectTable.XINT13 = &obstaculo;       // Sensor de proximidad
    PieVectTable.TINT0 = &cpu_timer0_isr;   // Timer0 de la CPU (función delay(ns));
    EDIS;

    InitCpuTimers();  // Inicializamos los Timer0, 1 y 2

    PieCtrlRegs.PIEIER9.bit.INTx1 = 1; // Interrupción en la PIE, línea INT9.1 (SCIRXINTA)
    PieCtrlRegs.PIEIER1.bit.INTx4 = 1; // Interrupción en la PIE, línea INT1.4 (XINT1)
    PieCtrlRegs.PIEIER1.bit.INTx5 = 1; // Interrupción en la PIE, línea INT1.5 (XINT2)
    PieCtrlRegs.PIEIER1.bit.INTx7 = 1; // Interrupción en la PIE, línea INT1.7 (TINT0)

    IER |= 0x0001;
    IER |= 0x0100;
    IER |= 0x1000;
    IER |= 0x0800;

    EINT;  // Se habilitan todas las líneas de interrupción
    ERTM;
```

```
    I2CA_Init(); // Inicialización módulo I2C (era necesaria la interrupción del Timer0)

    inicero(buffer,100);
    reposo();

    while(1){ // Bucle de función principal

        distancia=midedistancia(&flag_echo);
        delay(60000);
        while (distancia<10.0 && distancia>1.0){
            reposo();
            distancia=midedistancia(&flag_echo);
            delay(60000);
        }

        estacion=obtenerRGB(&rojo, &azul, &verde);

        if (dest!=0)recorrido(dest, vel, estacion, &nointerrumpir);
        if (dest==estacion)dest=0;

        if (flag_rx!=0){
            lectura(buffer, &dest, &flag_rx0, &modo, &i, &vel);
            flag_rx=0;
        }

     }
}
```

III. Funciones Auxiliares

a. Movimiento

```
/*********************************************************************/
/************************** MOVIMIENTO ***************************/
/*********************************************************************/

void reposo(void){

    GpioDataRegs.GPACLEAR.bit.GPIO15 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO6 = 1;

    EPwm2Regs.CMPA.half.CMPA = EPwm2Regs.TBPRD*0.0;       //iz
    EPwm6Regs.CMPA.half.CMPA = EPwm6Regs.TBPRD*0.0;       //der

}
void avance1(void){

    GpioDataRegs.GPACLEAR.bit.GPIO15 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO6 = 1;

    EPwm2Regs.CMPA.half.CMPA = 37500*0.3;       //del iz
    EPwm6Regs.CMPA.half.CMPA = 37500*0.3;       //tras iz
}

void avance2(void){

    GpioDataRegs.GPACLEAR.bit.GPIO15 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO6 = 1;

    EPwm2Regs.CMPA.half.CMPA = 37500*0.5;       //del iz
    EPwm6Regs.CMPA.half.CMPA = 37500*0.5;       //tras iz
}

void izquierda1(void){


    GpioDataRegs.GPACLEAR.bit.GPIO15 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO6 = 1;

    EPwm2Regs.CMPA.half.CMPA = 37500*0.4;       //del der
    EPwm6Regs.CMPA.half.CMPA = 37500*0.0;       // tras iz
}

void derecha1(void){

    GpioDataRegs.GPACLEAR.bit.GPIO15 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO6 = 1;

    EPwm2Regs.CMPA.half.CMPA = 37500*0.0;      //del iz
    EPwm6Regs.CMPA.half.CMPA = 37500*0.4;       // tras iz
}

void izquierda2(void){


    GpioDataRegs.GPACLEAR.bit.GPIO15 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO6 = 1;

    EPwm2Regs.CMPA.half.CMPA = 37500*0.6;       //del der
    EPwm6Regs.CMPA.half.CMPA = 37500*0.0;       // tras iz
}

void derecha2(void){
```

```
    GpioDataRegs.GPACLEAR.bit.GPIO15 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO6 = 1;

    EPwm2Regs.CMPA.half.CMPA = 37500*0.0;      //del iz
    EPwm6Regs.CMPA.half.CMPA = 37500*0.6;       // tras iz
}

void recorrido (int dest, int vel, int estacion, int *nointerrumpir){

   int flag_izquierda=0;
   int flag_derecha=0;
   int flag_avance=0;
   int flag_llegada=0;

   switch (dest){

       case 0:
           reposo();

       case 1:
           switch(estacion){
                        case 1: flag_llegada=1;   break;
                        case 2: flag_izquierda=1; break;
                        case 3: flag_avance=1;    break;
                        case 4: flag_izquierda=1; break;
                        case 5: flag_avance=1;    break;
                        case 6: flag_derecha=1;   break;
                        case 7: flag_avance=1;    break;
                        case 8: flag_avance=1;    break;
                        case 0: flag_avance=1;    break;
                    }
        break;

        case 2:
            switch(estacion){
                        case 1: flag_izquierda=1; break;
                        case 2: flag_llegada=1;   break;
                        case 3: flag_avance=1;    break;
                        case 4: flag_izquierda=1; break;
                        case 5: flag_avance=1;    break;
                        case 6: flag_derecha=1;   break;
                        case 7: flag_avance=1;    break;
                        case 8: flag_avance=1;    break;
                        case 0: flag_avance=1;    break;
                     }
        break;

        case 3:
            switch(estacion){
                        case 1: flag_izquierda=1; break;
                        case 2: flag_avance=1;    break;
                        case 3: flag_llegada=1;   break;
                        case 4: flag_izquierda=1; break;
                        case 5: flag_derecha=1;   break;
                        case 6: flag_derecha=1;   break;
                        case 7: flag_avance=1;    break;
                        case 8: flag_avance=1;    break;
                        case 0: flag_avance=1;    break;
                     }
        break;

        case 4:
            switch(estacion){
                        case 1: flag_izquierda=1; break;
                        case 2: flag_avance=1;    break;
                        case 3: flag_avance=1;    break;
```

```
                        case 4: flag_llegada=1;   break;
                        case 5: flag_derecha=1;   break;
                        case 6: flag_derecha=1;   break;
                        case 7: flag_avance=1;    break;
                        case 8: flag_avance=1;    break;
                        case 0: flag_avance=1;    break;
                    }
        break;

        case 5:
            switch(estacion){
                        case 1: flag_avance=1;    break;
                        case 2: flag_avance=1;    break;
                        case 3: flag_avance=1;    break;
                        case 4: flag_derecha=1;   break;
                        case 5: flag_llegada=1;   break;
                        case 6: flag_derecha=1;   break;
                        case 7: flag_avance=1;    break;
                        case 8: flag_avance=1;    break;
                        case 0: flag_avance=1;    break;
                    }
        break;

        case 6:
            switch(estacion){
                        case 1: flag_izquierda=1; break;
                        case 2: flag_avance=1;    break;
                        case 3: flag_avance=1;    break;
                        case 4: flag_derecha=1;   break;
                        case 5: flag_derecha=1;   break;
                        case 6: flag_llegada=1;   break;
                        case 7: flag_avance=1;    break;
                        case 8: flag_avance=1;    break;
                        case 0: flag_avance=1;    break;
                    }
             break;

        case 7:
            switch(estacion){
                        case 1: flag_izquierda=1; break;
                        case 2: flag_avance=1;    break;
                        case 3: flag_avance=1;    break;
                        case 4: flag_derecha=1;   break;
                        case 5: flag_derecha=1;   break;
                        case 6: flag_avance=1;    break;
                        case 7: flag_llegada=1;   break;
                        case 8: flag_avance=1;    break;
                        case 0: flag_avance=1;    break;
                    }
            break;

        case 8
        :
            switch(estacion){
                        case 1: flag_avance=1;    break;
                        case 2: flag_izquierda=1; break;
                        case 3: flag_avance=1;    break;
                        case 4: flag_derecha=1;   break;
                        case 5: flag_avance=1;    break;
                        case 6: flag_avance=1;    break;
                        case 7: flag_avance=1;    break;
                        case 8: flag_llegada=1;   break;
                        case 0: flag_avance=1;    break;
                    }
            break;
}
```

```
    if (flag_llegada==1){
        reposo();
    }

    else if (flag_izquierda==1){
        *nointerrumpir=1;
        switch(vel){
                case 0: reposo();     break;
                case 1: izquierda1(); delay(10000); break;
                case 2: izquierda2(); delay(10000); break;
                case 3: izquierda3(); delay(10000); break;
        }
        *nointerrumpir=0;
     }

    else if (flag_derecha==1){
        *nointerrumpir=1;
        switch(vel){
                case 0: reposo();   break;
                case 1: derecha1(); delay(10000); break;
                case 2: derecha2(); delay(10000); break;
                case 3: derecha3(); delay(10000); break;
        }
        *nointerrumpir=0;
     }

    else if (flag_avance==1){
        switch(vel){
                case 0: reposo();  break;
                case 1: avance1(); break;
                case 2: avance2(); break;
                case 3: avance3(); break;
        }
     }

}
```

b. Sensor de Color

```
/*****************************************************************/
/************************** SENSOR RGB ***************************/
/*****************************************************************/


int obtenerRGB(float *rojo, float *azul, float *verde){

    // Función que nos proporciona los valores RGB detectados por el TCS34725
    int b=I2C_leer_dos_bytes(TCS34725_BDATAL);
    int g=I2C_leer_dos_bytes(TCS34725_GDATAL);
    int r=I2C_leer_dos_bytes(TCS34725_RDATAL);
    int c=I2C_leer_dos_bytes(TCS34725_CDATAL);

    *rojo = (float)r / c * 255.0;
    *verde = (float)g / c * 255.0;
    *azul = (float)b / c * 255.0;

    int estacion=0;

    if(*rojo>30 && *rojo<50 && *verde>95 && *verde<115 && *azul>115 && *azul<135){
        estacion=4; //Celeste
    }

    else if(*rojo>140 && *rojo<160 && *verde>70 && *verde<90 && *azul>50 && *azul<70){
        estacion=6; //Naranja
    }
```

```
    else if(*rojo>45 && *rojo<65 && *verde>120 && *verde<140 && *azul>60 && *azul<80){
        estacion=2; //Lima
    }

    else if(*rojo>90 && *rojo<110 && *verde>100 && *verde<120 && *azul>45 && *azul<65){
        estacion=5; //Amarillo
    }

    else if(*rojo>100 && *rojo<120 && *verde>65 && *verde<80 && *azul>90 && *azul<105){
        estacion=1; //Fucsia
    }

    else if(*rojo>50 && *rojo<70 && *verde>80 && *verde<95 && *azul>155 && *azul<170){
        estacion=8; //Azul
    }

    else if(*rojo>85 && *rojo<95 && *verde>110 && *verde<120 && *azul>110 && *azul<120){
        estacion=7; //Verde
    }

    else if(*rojo>105 && *rojo<120 && *verde>95 && *verde<110 && *azul>90 && *azul<105){
        estacion=3; //Morado
    }

    else{
        estacion=0; //No se registra ningún color
    }

    return estacion;

}

Uint16 I2C_leer_dos_bytes(Uint16 registro){

    // Función que permite leer dos bytes del módulo I2C esclavo
    int valor;

    while (I2caRegs.I2CSTR.bit.BB == 1);        // Aseguramos que el bus de datos esté libre
    I2caRegs.I2CSTR.bit.SCD = 1;                // Esperamos a la señal de STOP
    while(I2caRegs.I2CMDR.bit.STP == 1);        // Reiniciamos la señal de STOP

    //Primero tenemos que indicar que registro queremos leer

    I2caRegs.I2CCNT = 1;                        // Eviamos un byte
    I2caRegs.I2CMDR.all = 0x6620;               // Configuramos el I2C como transmisor
    while(I2caRegs.I2CSTR.bit.XRDY == 0);       // Aseguramos que el bus de datos esté vacío
    I2caRegs.I2CDXR = 0xA0 | registro;          // Accedemos al registro que queremos leer
    while(!I2caRegs.I2CSTR.bit.ARDY);           // Aseguramos que el registro está disponible

    I2caRegs.I2CCNT = 2;                        // Vamos a leer dos bytes
    I2caRegs.I2CMDR.all = 0x6C20;           // Configuramos el I2C como receptor
    while(I2caRegs.I2CSTR.bit.RRDY == 0);   // Esperamos a que el bus reciba información
    valor=I2caRegs.I2CDRR;                  // Copiamos los datos a I2CDRR (LSB)
    while(I2caRegs.I2CSTR.bit.RRDY == 0);   // Esperamos a que el bus reciba información
    valor+=I2caRegs.I2CDRR << 8;            // Copiamos los datos a I2CDRR (MSB)
    return(valor);
}
```

c. Sensor de Proximidad

```
/*****************************************************************************/
/************************** SENSOR DE PROXIMIDAD *****************************/
/*****************************************************************************/

float midedistancia(int *flag_echo){
```

```
    float tus=0;
    float ts=0;
    float distancia=0.0;

    GpioDataRegs.GPACLEAR.bit.GPIO13 = 1;        // Aseguramos que TRIGGER está desactivado
    delay(5);

    Timer2ECHO (&CpuTimer2);    // Configuramos el Timer2 (lo usamos en la función delay)

    GpioDataRegs.GPASET.bit.GPIO13 = 1;          // Activamos el TRIGGER
    delay(10);                                   // Lo mantenemos 10 us
    GpioDataRegs.GPACLEAR.bit.GPIO13 = 1;        // Desactivamos el TRIGGER

    delay(10000);
    *flag_echo=0;

    tus = (0xffffffff - CpuTimer2Regs.TIM.all)/150;  // Duración del flanco en us
    ts = tus / 1000000;                              // Duración del flanco en s
    distancia = 100*ts*340/2 ;                       // Distancia a obstáculo en cm
    return distancia;
}

void Timer2ECHO (struct CPUTIMER_VARS *Timer) // Configuración del timer para medida del ECHO
{
    Uint32  temp;

    Timer->CPUFreqInMHz = 150;
    Timer->PeriodInUSec = 0xffffffff; // Período máximo que podemos aplicarle al TIMER
    temp = (long) (150*0xffffffff);
    Timer->RegsAddr->PRD.all = temp;

    // Set pre-scale counter to divide by 1 (SYSCLKOUT)

    Timer->RegsAddr->TPR.all  = 0;
    Timer->RegsAddr->TPRH.all  = 0;

    // Initialize timer control register

    // 1 = Stop timer, 0 = Start/Restart Timer

    Timer->RegsAddr->TCR.bit.TSS = 1;

    Timer->RegsAddr->TCR.bit.TRB = 1;    // 1 = reload timer
    Timer->RegsAddr->TCR.bit.SOFT = 1;
    Timer->RegsAddr->TCR.bit.FREE = 1;   // Timer Free Run

    // 0 = Disable/ 1 = Enable Timer Interrupt

    Timer->RegsAddr->TCR.bit.TIE = 0; // Desactivamos la interrupción, lo pararemos
//manualmente

    // Reset interrupt counter

    Timer->InterruptCount = 0;
}
```

d. Lectura HC-06

```
/*****************************************************************************/
/********************************* BLUETOOTH *********************************/
/*****************************************************************************/

void lectura(char buffer[], int *dest, int *flag_rx0, int *modo, int *i, int *vel){
```

```
    int d=0;
    int v=0;

    d=*dest;
    v=*vel;

    if(*flag_rx0!=0){

        if (d==0){

           if (strstr(buffer,"entrada") != NULL || strstr(buffer,"recep") != NULL)d=1;
           else if (strstr(buffer,"gener") != NULL)d=2;
           else if (strstr(buffer,"oftal") != NULL)d=3;
           else if (strstr(buffer,"neur") != NULL)d=4;
           else if (strstr(buffer,"derma") != NULL)d=5;
           else if (strstr(buffer,"pedia") != NULL)d=6;
           else if (strstr(buffer,"alerg") != NULL)d=7;
           else if (strstr(buffer,"urgen") != NULL)d=8;
        }

        if (strstr(buffer,"para") != NULL || strstr(buffer,"detente") != NULL)v=0;
        else if (strstr(buffer,"sigue" ) != NULL || strstr(buffer,"vamos") != NULL)v=1;
        else if (strstr(buffer,"lento") != NULL || strstr(buffer,"despacio") != NULL){
           if (v>1)v=v-1;
        }
        else if (strstr(buffer,"rápido") != NULL || strstr(buffer,"deprisa") != NULL){
           if (v<3)v=v+1;
        }
        inicero(buffer,100);
        *flag_rx0=0;
        }

    if(*modo==1){

       if (*dest==0){

           if (buffer[*i-1] == 'e')d=1;
           else if (buffer[*i-1] == 'g')d=2;
           else if (buffer[*i-1] == 'o')d=3;
           else if (buffer[*i-1] == 'n')d=4;
           else if (buffer[*i-1] == 'd')d=5;
           else if (buffer[*i-1] == 'p')d=6;
           else if (buffer[*i-1] == 'a')d=7;
           else if (buffer[*i-1] == 'u')d=8;
       }

       if (buffer[*i-1] == '0')v=0;
       else if (buffer[*i-1] == '1')v=1;
       else if (buffer[*i-1] == '2')v=2;
       else if (buffer[*i-1] == '3')v=3;
       inicero(buffer,100);
       *i=0;
    }

    *dest=d;
    *vel=v;
}
```

e. Otros

```
/*****************************************************************************/
/********************************** OTROS ************************************/
/*****************************************************************************/

// Función que rellena de 0s el buffer de datos de la comunicación SCI
```

```
void inicero(char buffer[],int size){
    int j;
    for(j=0;j<size;j++){
        buffer[j]=0;
    }
}

// Función que retrasa la ejecución de la siguiente línea en microsegundos

void delay(long int us){
    ConfigCpuTimer(&CpuTimer0,150,us);
    CpuTimer0Regs.TCR.bit.TSS = 0;
    while (CpuTimer0Regs.TCR.bit.TSS == 0){}
}
```

IV. Interrupciones

a. *SCIA_RX_isr*

```
interrupt void SCIA_RX_isr(void)
{
    if (SciaRegs.SCIRXST.bit.BRKDT == 1)
    {
        SciaRegs.SCICTL1.bit.SWRESET=0;
        SciaRegs.SCICTL1.bit.SWRESET=1; //Hold current state
    }
    else
    {
        flag_rx=1;
        buffer[i++] = SciaRegs.SCIRXBUF.bit.RXDT;

        if (buffer[i-1]=='+'){
            modo=1;
            inicero(buffer,100);
            i=0;
        }
        if (buffer[i-1]=='-'){
            modo=0;
        }

        if (modo==0)
        {
            if(buffer[i-1] == '#')
            {
                flag_rx0=1;
                i=0;
            }
        }
    }
    PieCtrlRegs.PIEACK.bit.ACK9 = 1;       // Clear al registro PIEACK para la línea INT9
}
```

b. *Rectificar*

```
interrupt void rectificar(void){

    if (dest!=0 && nointerrumpir==0){

    if(GpioDataRegs.GPADAT.bit.GPIO11==1 && GpioDataRegs.GPADAT.bit.GPIO7==0){ // Desvío hacia
// la izquierxa
        switch(vel){

        case 0: reposo();   break;
        case 1: derecha1(); break;
        case 2: derecha2(); break;
        case 3: derecha3(); break;

        }

        while(GpioDataRegs.GPADAT.bit.GPIO7==0){
        }

        switch(vel){

        case 0: reposo();  break;
        case 1: avance1(); break;
        case 2: avance2(); break;
        case 3: avance3(); break;
        }
    }
```

```
    else if(GpioDataRegs.GPADAT.bit.GPIO11==0 && GpioDataRegs.GPADAT.bit.GPIO7==1){ // Desvío
// hacia la derecha
        switch(vel){

        case 0: reposo();     break;
        case 1: izquierda1(); break;
        case 2: izquierda2(); break;
        case 3: izquierda3(); break;

        }
        while(GpioDataRegs.GPADAT.bit.GPIO11==0){
        }

        switch(vel){

        case 0: reposo();  break;
        case 1: avance1(); break;
        case 2: avance2(); break;
        case 3: avance3(); break;

        }
    }
    else if(GpioDataRegs.GPADAT.bit.GPIO11==0 && GpioDataRegs.GPADAT.bit.GPIO7==0){ // Se sale
// completamente
        reposo();
        while(GpioDataRegs.GPADAT.bit.GPIO11==0 && GpioDataRegs.GPADAT.bit.GPIO7==0){
        }
    }
    }
    PieCtrlRegs.PIEACK.bit.ACK1 = 1;
}
```

c. *Obstaculo*

```
interrupt void obstaculo(void){
    EALLOW;

        if (flag_echo == 0)      //Flanco de subida detectado
        {
            CpuTimer2Regs.TCR.bit.TSS = 0;  // Se inicia la cuenta atrás del Timer2
            flag_echo = 1;
        }
        else                                //Flanco de bajada detectado
        {
            CpuTimer2Regs.TCR.bit.TSS = 1;  // Se detiene la cuenta atrás del Timer2
            flag_echo = 0;                  //Bajamos la bandera
        }
        EDIS;
}
```

d. *Cpu_timer0_isr*

```
interrupt void cpu_timer0_isr (void)
{
    CpuTimer0Regs.TCR.bit.TSS = 1;  // Se detiene la cuenta atrás del Timer0

    PieCtrlRegs.PIEACK.bit.ACK1 = 1;
}
```

V. Fichero header "*coche.h*"

```
#ifndef COCHE_H_
#define COCHE_H_

#include "DSP2833x_Device.h"
#include <string.h>
#include <stdlib.h>
#include "TCS24725.h"

// Configuración e Inicialización de módulos

void Setup_ePWM(void);  // Configuración de módulo ePWM
void Gpio_select(void); // Configuración de GPIOs
void I2CA_Init(void);   // Inicialización I2C
void SCIA_init(void);   // Inicialización SCI

// Funciones propias

void inicero(char buffer[],int size);         // Pone a 0 el buffer de datos
void delay(long int us);                      // Mantiene a la espera durante n microsegundos

void lectura (char buffer[], int *dest, int *flag_rx0, int *modo, int *i, int *vel);

Uint16 I2C_leer_dos_bytes(Uint16 registro);   // Lee dos bytes de memoria por comunicación I2C
int obtenerRGB(float *rojo, float *azul, float *verde);  // Obtiene los valores RGB del sensor
//TCS24725 y devuelve el color

float midedistancia(int *flag_echoFi);      // Mide la distancia entre el sensor de proximidad
//y el obstáculo (si lo hay)
void Timer2ECHO (struct CPUTIMER_VARS *);   // Configura el Timer2 para medir el ECHO

void recorrido(int dest, int vel, int estacion, int *nointerrumpir);
void reposo (void);
void avance1(void);
void derecha1(void);
void izquierda1(void);
void avance2(void);
void derecha2(void);
void izquierda2(void);

#endif /* COCHE_H_ */
```

VI. Fichero header "*TCS34725.h*"

```
#ifndef TCS24725_H_
#define TCS24725_H_

#define TCS24725_SLAVE          0x29    // slave address TCS24725

#define TCS34725_COMMAND_BIT      0x80

#define TCS34725_ENABLE           0x00
#define TCS34725_ENABLE_AIEN      0x10    /* RGBC Interrupt Enable */
#define TCS34725_ENABLE_WEN       0x08    /*ç/ Wait enable - Writing 1 activates the wait
timer
#define TCS34725_ENABLE_AEN       0x02    /* RGBC Enable - Writing 1 actives the ADC, 0
disables it */
#define TCS34725_ENABLE_PON       0x01    /* Power on - Writing 1 activates the internal
oscillator, 0 disables it */
#define TCS34725_ATIME            0x01    /* Integration time */
#define TCS34725_WTIME            0x03    /* Wait time (if TCS34725_ENABLE_WEN is asserted) */
#define TCS34725_WTIME_2_4MS      0xFF    /* WLONG0 = 2.4ms   WLONG1 = 0.029s */
#define TCS34725_WTIME_204MS      0xAB    /* WLONG0 = 204ms   WLONG1 = 2.45s  */
#define TCS34725_WTIME_614MS      0x00    /* WLONG0 = 614ms   WLONG1 = 7.4s   */
#define TCS34725_AILTL            0x04    /* Clear channel lower interrupt threshold */
#define TCS34725_AILTH            0x05
#define TCS34725_AIHTL            0x06    /* Clear channel upper interrupt threshold */
#define TCS34725_AIHTH            0x07
#define TCS34725_PERS             0x0C    /* Persistence register - basic SW filtering
mechanism for interrupts */
#define TCS34725_PERS_NONE        0b0000  /* Every RGBC cycle generates an interrupt
*/
#define TCS34725_PERS_1_CYCLE     0b0001  /* 1 clean channel value outside threshold range
generates an interrupt   */
#define TCS34725_PERS_2_CYCLE     0b0010  /* 2 clean channel values outside threshold range
generates an interrupt  */
#define TCS34725_PERS_3_CYCLE     0b0011  /* 3 clean channel values outside threshold range
generates an interrupt  */
#define TCS34725_PERS_5_CYCLE     0b0100  /* 5 clean channel values outside threshold range
generates an interrupt  */
#define TCS34725_PERS_10_CYCLE    0b0101  /* 10 clean channel values outside threshold range
generates an interrupt */
#define TCS34725_PERS_15_CYCLE    0b0110  /* 15 clean channel values outside threshold range
generates an interrupt */
#define TCS34725_PERS_20_CYCLE    0b0111  /* 20 clean channel values outside threshold range
generates an interrupt */
#define TCS34725_PERS_25_CYCLE    0b1000  /* 25 clean channel values outside threshold range
generates an interrupt */
#define TCS34725_PERS_30_CYCLE    0b1001  /* 30 clean channel values outside threshold range
generates an interrupt */
#define TCS34725_PERS_35_CYCLE    0b1010  /* 35 clean channel values outside threshold range
generates an interrupt */
#define TCS34725_PERS_40_CYCLE    0b1011  /* 40 clean channel values outside threshold range
generates an interrupt */
#define TCS34725_PERS_45_CYCLE    0b1100  /* 45 clean channel values outside threshold range
generates an interrupt */
#define TCS34725_PERS_50_CYCLE    0b1101  /* 50 clean channel values outside threshold range
generates an interrupt */
#define TCS34725_PERS_55_CYCLE    0b1110  /* 55 clean channel values outside threshold range
generates an interrupt */
#define TCS34725_PERS_60_CYCLE    0b1111  /* 60 clean channel values outside threshold range
generates an interrupt */
#define TCS34725_CONFIG           0x0D
#define TCS34725_CONFIG_WLONG     0x02    /* Choose between short and long (12x) wait times
via TCS34725_WTIME */
#define TCS34725_CONTROL          0x0F    /* Set the gain level for the sensor */
#define TCS34725_ID               0x12    /* 0x44 = TCS34721/TCS34725, 0x4D =
TCS34723/TCS34727 */
```

```
#define TCS34725_STATUS           0x13
#define TCS34725_STATUS_AINT      0x10    /* RGBC Clean channel interrupt */
#define TCS34725_STATUS_AVALID    0x01    /* Indicates that the RGBC channels have completed
an integration cycle */
#define TCS34725_CDATAL           0x14    /* Clear channel data */
#define TCS34725_CDATAH           0x15
#define TCS34725_RDATAL           0x16    /* Red channel data */
#define TCS34725_RDATAH           0x17
#define TCS34725_GDATAL           0x18    /* Green channel data */
#define TCS34725_GDATAH           0x19
#define TCS34725_BDATAL           0x1A    /* Blue channel data */
#define TCS34725_BDATAH           0x1B

#endif
```