Trabajo Fin de Grado
Grado de Ingeniería en Tecnologías Industriales

Mapeado de Superficie con Robot Inteligente basado en DSP TMS320F28335 de Texas Instruments

Autor: Álvaro Rodríguez Gómez
Tutor: Federico José Barrero García

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

Sevilla, 2019

Trabajo Fin de Grado
Grado de Ingeniería en Tecnologías Industriales

Mapeado de Superficie con Robot Inteligente basado en DSP TMS320F28335 de Texas Instruments

Autor:
Álvaro Rodríguez Gómez

Tutor:
Federico José Barrero García
Catedrático de Universidad

Dpto. de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2019

Proyecto Fin de Carrera: Mapeado de Superficie con Robot Inteligente basado en DSP TMS320F28335 de Texas Instruments

Autor: Álvaro Rodríguez Gómez

Tutor: Federico José Barrero García

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

Presidente:

Vocales:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

Me gustaría dedicar unas palabras de agradecimiento a todas aquellas personas que me han acompañado durante toda mi vida, y especialmente durante estos últimos 4 años mientras cursaba el Grado de Ingeniería en Tecnologías Industriales.

Agradezco a los docentes del Grado de Ingeniería en Tecnologías Industriales de la Escuela Técnica Superior de Ingeniería de la Universidad de Sevilla su tiempo y dedicación durante estos 4 años de formación. Y en especial al tutor de este trabajo de fin de grado, el Catedrático D. Federico José Barrero García por su entusiamo impartiendo la asignatura de Electrónica Industrial y por su disponibilidad y apoyo durante la elaboración del trabajo.

A mis amigos de toda la vida y a los que he conocido en la ETSI, que siempre han estado a mi lado en las buenas y en las malas.

Finalmente agradecer a mis padres y abuelos que son mi motor y mayor inspiración, que a través de su amor, paciencia, consejo y buenos valores me ayudan a trazar mi camino.

Álvaro Rodríguez Gómez

Sevilla, 2019

Resumen

El objetivo principal de este trabajo de fin de grado consiste en la realización y programación de un robot autónomo, dotado de los componentes necesarios para poder recorrer una habitación reconociendo los obstáculos y evitándolos a la vez que realiza un mapa de la superficie, sin necesidad de ser guiado ni corregido por acciones externas.

Abstract

The main objective of this end-of-degree work consists in the creation and programming of an autonomous robot, equipped with the necessary components to be able to cross a room recognizing the obstacles and avoiding them at the same time as it makes a map of the surface, without the need to be guided or corrected by external actions.

Índice

Agradecimientos **vii**

Resumen **ix**

Abstract **xi**

Índice **xii**

Índice de Tablas **xiv**

Índice de Figuras **xv**

Notación **xvii**

1 Introducción **1**

2 Mapeo **3**
2.1 Algoritmos y Representación de Mapas *3*

3 Componentes del Robot **7**
3.1 Principales Componentes *12*
3.1.1 Microprocesador DSP TMS320F28335 y Kit de Experimentación 12
3.1.2 Sensor de Ultrasonidos HC-SR04 14
3.1.3 Ruedas y Motores 16
3.1.4 Giroscopio / Acelerómetro BMI088 19
3.2 Montaje Final del Robot *21*

4 Programación del Robot **23**
4.1 Programación de la Función Principal *23*
4.1.1 Mapa de Rejilla de Ocupación 24
4.2 Programación del Sensor de Ultrasonidos *24*
4.3 Programación del Giroscopio *26*
4.4 Programación del Módulo PWM *27*

5 Pruebas del Robot **31**
5.1 Pruebas PWM *31*
5.2 Pruebas Sensor Ultrasonidos *32*
5.3 Pruebas Giroscopio *33*
5.4 Pruebas Integrales *33*

6 Posibles Ampliaciones **35**

7 Conclusiones **37**

Referencias **38**

Glosario **40**

Anexo I: Código Función Principal **41**

Anexo II: Código Módulo PWM **46**

Anexo III: Código Sensor Ultrasonidos **51**

Anexo IV: Código Giroscopio **56**

ÍNDICE DE TABLAS

Tabla 3–1 Comparativa de algoritmos de mapeo 4
Tabla 4–1 Conexionado del sensor de ultrasonidos 15
Tabla 4–2 Conexionado de los drivers de los motores 19
Tabla 4–3 Conexionado del giroscopio 20
Tabla 5–1 Duty cycle módulo PWM 28
Tabla 6–1 Pruebas PWM 32

ÍNDICE DE FIGURAS

Figura 2-1. Mapa de rejilla de ocupación 5
Figura 2-2. Mapa topológico 5
Figura 2-3. Mapa vector percepción general 6
Figura 3-1. Modelo diferencial 8
Figura 3-2. Modelo síncrono 9
Figura 3-3. Modelo triciclo 9
Figura 3-4. Modelo Ackerman 9
Figura 3-5. Puente en H 10
Figura 3-6. Diagrama de pines del DSP TMS320F8335 12
Figura 3-7. Diagrama de Bloques Funcional 13
Figura 3-8. Kit experimentación Delfino F28335 14
Figura 3-9. Sensor ultrasónico HC-SR04 15
Figura 3-10. Esquemático circuito adaptador de 3.3 V a 5 V DC 16
Figura 3-11. Elevador de tensión de 0.9 a 5 V con salida 5 V DC y 480 mA 16
Figura 3-12. Ruedas 17
Figura 3-13. Motor con caja reductora 17
Figura 3-14. Driver Dual Pololu DRV8835 18
Figura 3-15. Pines del driver DRV8835 18
Figura 3-16. Placa base BMI088 Bosch 19
Figura 3-17. Montaje final del robot 21
Figura 4-1. Captura de Pantalla de Code Composer Studio (CCS) 23
Figura 4-2. Máquina de estados del sensor de ultrasonidos 25
Figura 4-3. Cálculo de la distancia al obstáculo 25
Figura 4-4. Máquina de estados del giroscopio 26
Figura 4-5. Cálculo de la velocidad angular 27
Figura 4-6. Modulación PWM 27
Figura 4-7. Programación PWM giro a la derecha 28
Figura 4-8. Primera máquina de estados PWM 29
Figura 4-9. Segunda máquina de estados PWM 30
Figura 5-1. Función “delay_loop” 31
Figura 5-2. Pruebas de los motores 32
Figura 5-3. Distancia medida a 0.5 m 33
Figura 5-4. Distancia medida a 1 m 33
Figura 5-5. Distancia medida a 1.5 m 33
Figura 5-6. Ángulo de giro calculado y velocidad angular medida 33

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

Actualmente, el mapeo de superficies con la utilización de robots móviles es un aspecto fundamental en numerosos ámbitos de investigación, como por ejemplo la astronomía, siendo robots los encargados de reconocer la superficie de los planetas cercanos como Marte; la geología, gracias a la posibilidad de acceder a cuevas o a lugares de difícil acceso para su exploración; y en general para el análisis de cualquier entorno, incluso aquellos inaccesibles por el hombre. Es por ello por lo que, a lo largo de estos últimos años, el estudio de nuevas técnicas de mapeo de superficies se ha convertido en indispensable, con el fin de posibilitar la autonomía de los robots y el conocimiento de su posición y del entorno que les rodea.

El proceso de creación del mapa de una superficie con un robot móvil requiere de dos pasos, los cuales hacen del mismo una labor compleja:

- Construcción y programación del robot: lo cual se realizará en este proyecto, añadiendo los sensores y actuadores necesarios para captar el entorno por el que circula el autómata, así como programando los algoritmos para el movimiento de éste cubriendo en la medida de lo posible la totalidad de la superficie.
- Análisis de los datos obtenidos: para lo cual es necesario exportar los datos que obtiene el robot de sus alrededores a un programa que permita elaborar el mapa de la superficie, lo cual no se realiza en el proyecto, ya que no se dispone de los medios necesarios para ello.

Este primer capítulo, en el cual se encuentra una breve introducción y se presentan los objetivos del proyecto, tras lo cual se expondrán los siguientes capítulos:

- **Capítulo 2: Mapeo**

  En este capítulo se describen los distintos métodos de mapeo de superficies y sus clasificaciones.

- **Capítulo 3: Componentes del Robot**

  En este capítulo se describen los componentes de un robot autónomo, así como los componentes específicos utilizados.

- **Capítulo 4: Programación del Robot**

  En este capítulo se describe la programación de los componentes del robot.

- **Capítulo 5: Pruebas del Robot**

  En este capítulo se describen las pruebas realizadas a lo largo de la ejecución del proyecto hasta lograr el correcto funcionamiento del robot.

- **Capítulo 6: Posibles Ampliaciones**

  En este capítulo se enumeran las distintas ampliaciones aplicables al proyecto.

- **Capítulo 7: Conclusiones**

  En este capítulo se resumen las principales conclusiones extraídas, así como las lecciones aprendidas durante la ejecución del trabajo de fin de grado.

El objetivo principal de este proyecto consiste en el montaje y programación de un robot autónomo, el cual gracias a los componentes que lo integran pueda realizar un mapa de un recinto y desplazarse por el mismo sin necesidad de ningún tipo de control externo.

Con el fin de lograr dicho objetivo será necesario el uso de sensores y actuadores, gracias a los cuales se podrá realizar el mapeo analizando las medidas obtenidas. Para ello será necesario realizar un previo estudio de los sensores existentes de forma que, teniendo en cuenta su funcionamiento, se puedan elegir aquellos que mejor se adecuen a nuestras pretensiones a la vez que se demuestran los conocimientos adquiridos a lo largo del grado.

Es por ello que para lograr el objetivo principal se podría decir que se requieren diversos objetivos secundarios, entre los que destacan la adquisición de conocimientos acerca de los dispositivos disponibles en el mercado, su

análisis y elección, así como el aprendizaje del programa con el cual se programarán los componentes seleccionados.

Adicionalmente, hay que destacar un último objetivo secundario, y no por ello el menos importante, el de seguir aprendiendo y adquiriendo los conocimientos que permitirán abarcar el mundo laboral con una mínima solidez y confianza.

2 MAPEO

Los seres humanos, así como los animales, tenemos la capacidad de interactuar con nuestro entorno, examinándolo constantemente para poder desarrollar cualquier tipo de actividad. Sin embargo, únicamente algunos animales, incluyendo las personas, poseen la capacidad de elaborar un mapa de sus alrededores y almacenarlo en su memoria, así como el de los recorridos y lugares transitados a lo largo de su vida, y posteriormente interpretarlo.

Los estudios acerca de las técnicas de mapeo y los avances tecnológicos en este ámbito se enfocan a implementar esta misma capacidad en los robots, ya que contribuiría notablemente al posicionamiento y movimiento del sistema robótico a lo largo de la superficie reconocida.

2.1 Algoritmos y Representación de Mapas

Hoy en día, los sistemas robóticos dedicados al reconocimiento de superficies son capaces de tomar datos, como distancias y orientación (los cuales permiten su localización), mediante la exploración del terreno y posteriormente almacenarlos en su memoria. Con la recopilación de varias series de datos se forman segmentos que pueden guardar relaciones entre sí, obteniendo la lista de información con la que más adelante se podrá generar el mapa.

Los distintos métodos de representación de mapas se pueden clasificar de la siguiente manera:

- **Métodos Geométricos**

  Los métodos de mapeo de superficies incluidos en este grupo se basan en las relaciones geométricas entre los obstáculos que el robot encuentra a lo largo de su recorrido. Entre ellos se encuentran:

  - Espacio de Configuración.
  - Mapas Difusos.
  - Mapas de Rejilla de Ocupación

- **Métodos Topológicos**

  Los métodos que se encuentran en este grupo se basan en la representación de las localizaciones más importantes mediante nodos y su unión con arcos de grafo formando todo ello un mapa en forma de grafo. Entre ellos se encuentran:

  - Mapas Topológicos.
  - Mapas Basados en Características.
  - Vector de Percepción General.
  - Esquema de Seguimiento de Camino.

  Así mismo podemos encontrar otro tipo de algoritmos o métodos los cuales mezclan características de ambos grupos y se pueden adaptar según las necesidades, siendo estos los mapas híbridos.

- **Comparación de los diversos algoritmos**

  La tabla 3-1 [10], recoge las principales características de los algoritmos antes citados, así como sus principales ventajas e inconvenientes.

Sistemas de Representación	*Resumen Características*	*Ventajas*	*Inconvenientes*
Rejilla de Ocupación	Matriz de celdas que contienen la probabilidad de estar en el estado ocupado o vacío.	Facilidad de uso, construcción y actualización.	Ocupa mucho espacio en memoria y consume mayor tiempo.
Basados en Características	Representación del entorno por características paramétricas como líneas, puntos, arcos…	Más información métrica, idóneo para entornos estáticos. Menor espacio en memoria.	Difícil concretar obstáculos con exactitud. Difícil uso y actualización. No usable para entornos desestructurados.
Topológico	Grafo de nodos representando lugares del espacio, conectados por arcos si hay un camino entre ellos.	Menor espacio en memoria, compactado. Bueno para planificación de rutas.	Menor información métrico-espacial. Ambigüedad de reconocimiento de lugares.
Difusos	Conjunto difuso compuesto por la unión de conjunto vacío y ocupado de puntos del espacio.	Ventajas de Rejilla de Ocupación. Mejora de estimación de ocupación.	Desventajas de Rejilla de Ocupación. Cálculos de intersección de conjuntos.
Vector de Percepción General	Vector con información de los alrededores cercanos que se actualiza en cada movimiento fuera de este. Mapa parcial.	Poco espacio en memoria. Decisión de camino más rápida. Generación más rápida.	Mayor incertidumbre. No tiene memoria. Recalculado para zonas ya visitadas.
Esquema de Seguimiento de Camino	Genera hitos de referencia entre dos puntos. Almacena tuplas de posición relativa entre hitos.	Fácilmente escalable, bueno para planificación de rutas. Poco espacio en memoria.	Necesario definir hitos en el sendero. No información global.
Híbridos	Combinación de métodos. Típicamente entre uno topológico y otro métrico.	Se compensan las desventajas de los métodos.	Doble computación y mantenimiento. Dificultad de relación entre representaciones.
Espacio de Configuración	Lista de pares vértice/borde que definen el contorno del obstáculo	Estructurado. Solo representa el espacio ocupado. Alta precisión.	Dificultad para trazar rutas, computación más costosa.

Tabla 2–1 Comparativa de algoritmos de mapeo

Los métodos de mapeo de superficies más utilizados en la actualidad son los siguientes:

- **Mapas de Rejilla de Ocupación:**

  Este método consiste en la representación de una zona cuadrada de la superficie mediante una matriz de NxM celdas, cada una de las cuales almacena un valor entre 1 y 0 que indica la probabilidad de que en ella exista o no un obstáculo.

  Todas las celdas deben comenzar conteniendo un valor intermedio indicando la incertidumbre de su estado, y a lo largo de su recorrido de reconocimiento por la superficie el robot será capaz, gracias a sus sensores, de modificar este valor hasta llegar hasta su valor definitivo, siendo 1 la certeza de que la casilla se encuentra ocupada con un obstáculo, y 0 la seguridad de que la casilla está vacía.

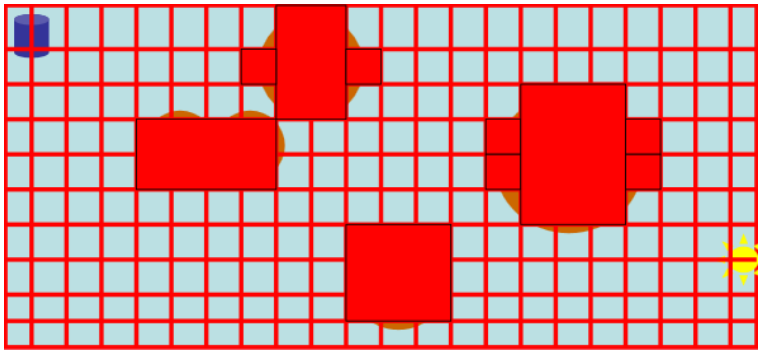


Figura 2-1. Mapa de rejilla de ocupación

- **Mapas Topológicos:**

  Los mapas topológicos son el ejemplo más representativo de los algoritmos pertenecientes al grupo de métodos topológicos, ya que consisten en la localización de lugares clave transitables y el almacenamiento de su posición, de forma que si es posible la circulación entre dos nodos se conectan mediante una serie de arcos. Este método es óptimo cuando se trata de encontrar el mejor camino, ya que traza las rutas posibles entre dos puntos diferentes.

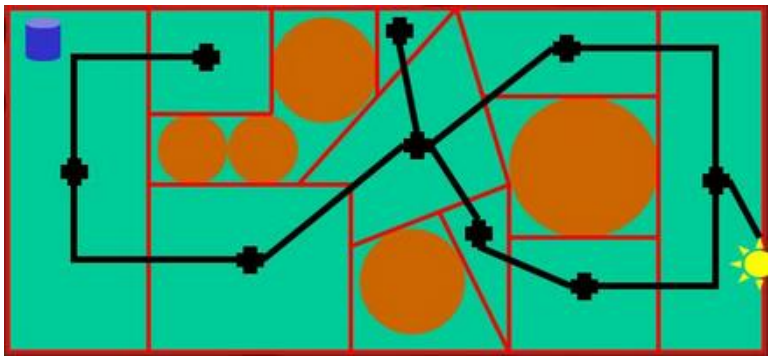


Figura 2-2. Mapa topológico

- **Vector de Percepción General**

  Este método consiste en el almacenamiento de todos los datos recogidos por el robot en una posición determinada en un único vector, logrando así una mayor velocidad a la hora de analizarlos, pero con menor precisión.

  Una vez extraídos estos datos, se obtiene una probabilidad de encontrar un obstáculo a una distancia determinada del autómata, con lo que éste se desplazará a una nueva posición segura donde realizará

una nueva recogida de datos para continuar con su progreso.

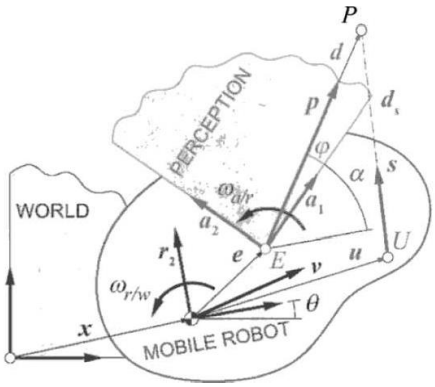


Figura 2-3. Mapa vector percepción general

En el caso que nos ocupa el método utilizado para el mapeo de la superficie es el de mapas de rejilla de ocupación.

3 COMPONENTES DEL ROBOT

Un robot autónomo requiere varios dispositivos para poder funcionar correctamente. Cada dispositivo tiene unas características que deben ser tenidas en cuenta a la hora de realizar el montaje.

Estos componentes, en general, se pueden agrupar de la siguiente forma:

- **Sensores.**
- **Motricidad.**
- **Comunicaciones.**
- **Control.**
- **Alimentación.**

A continuación, se hace una breve explicación genérica de cada elemento, incluyendo los distintos tipos de dispositivos que componen cada agrupación.

- **Sensores:**

  Un sensor es un dispositivo que está capacitado para detectar acciones o estímulos externos y responder en consecuencia. Es decir, nos permiten captar la información del medio físico que nos rodea. Se encargan de medir las magnitudes físicas y transformarlas en señales eléctricas capaces de ser entendidas por un microcontrolador.

  Los sensores más populares son:

  - Sensores de luz, distancia y proximidad: la función de estos sensores es medir o detectar la luz, posición o el movimiento de un determinado objeto en el espacio.
  - Sensores acústicos y piezo-eléctricos: los sensores acústicos son micrófonos pequeños que detectan o bien la presión de la onda de sonido (omnidireccional) o bien la velocidad de la onda de sonido (direccional). Estos sensores son sensibles a las presiones que emiten las ondas acústicas, y las transforma en pulsos eléctricos. Por su parte, los sensores piezoeléctricos se utilizan a menudo para el tacto, la vibración y las medidas de choque.
  - Sensores de presión, flexión y capacitivos: estos sensores son los más simples, ya que son interruptores que se activan o desactivan si se encuentran en contacto con un objeto.
  - Sensores de temperatura: los sensores de temperatura son dispositivos que transforman los cambios de temperatura en cambios en señales eléctricas que son procesados por equipo eléctrico o electrónico.
  - Sensores de desplazamiento e inclinación: los sensores rotatorios o deslizantes, también conocidos como potenciómetros, son resistores eléctricos con un valor de resistencia variable. Por su parte los sensores de inclinación son dispositivos que, en función de la inclinación, permiten o no el paso de la corriente funcionando como si fueran un interruptor.
  - Sensores de Aceleración: este tipo de sensores son muy importantes, ya que la información de la aceleración sufrida por un objeto es de vital importancia, ya que, si se produce una aceleración en un objeto, este experimenta una fuerza que tiende a hacer poner el objeto en movimiento.

  En el caso que nos ocupa, el robot equipa los siguientes sensores: sensor de ultrasonidos (para detectar obstáculos y determinar la distancia a los mismos) y un giroscopio/acelerómetro (para controlar el giro del robot)

- **Motricidad:**

  Motricidad o locomoción hace referencia al movimiento que realiza un animal, un microorganismo, un aparato o máquina para moverse de un lugar a otro, para trasladarse en el espacio. La locomoción varía en términos de forma, estructura, velocidad y otros elementos de acuerdo con el tipo de sujeto al que

hagamos referencia.

- Componentes de tracción:
  - Locomoción mediante patas: cuando la tarea a la que se destina el robot requiere de movilidad, los creadores de éstos han intentado imitar las distintas formas de desplazamiento de la que la naturaleza ha dotado a los animales, incluidos los humanos.

    Al dotar de movimiento con patas a un robot, debemos tener en cuenta su posición y velocidad, pero también debemos asegurar que el robot permanezca en equilibrio y no se caiga, usando solamente el movimiento en las articulaciones mediante motores. En robots bípedos, el desplazamiento requiere necesariamente mantener el equilibro en una de las patas mientras la otra se mueve, lo que conlleva una inestabilidad en cada paso. Los principales tipos de robots con patas son:

    - Robots bípedos.
    - Robots cuadrúpedos.
    - Robots hexápodos.
  - Locomoción mediante ruedas:
    - Motrices o De Tracción: Se componen de 2 ruedas en un eje común, cada rueda se controla independientemente, puede realizar movimientos en línea recta, en arco y sobre su propio eje de contacto de rodamiento, requiere de una o dos ruedas adicionales para balance o estabilidad. Sencillo mecánicamente, puede presentar problemas de estabilidad y su cinemática es sencilla (La cinemática de un robot se refiere a la manera en que se mueve), para lograr el movimiento en línea recta requiere que las dos ruedas de tracción giren a la misma velocidad.
    - Ruedas omnidireccionales: es una extensión del caso de dirección diferencial. Se tienen varias ruedas, normalmente paralelas, estos se configuran coordinadamente de tal forma que se muevan en la dirección que uno desee.

    En robots con ruedas, la cinemática estará ligada a la ubicación y numero de estas. Las composiciones más conocidas son:

    - Diferencial: siendo el más sencillo, se compone de dos ruedas de tracción unidas en un eje común y otras dos que le dan estabilidad. La traslación y rotación viene determinada por las ruedas de tracción, que se controlan de forma independiente.

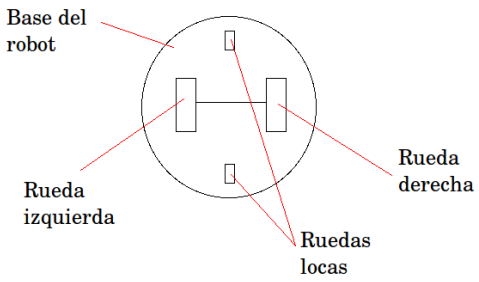


Figura 3-1. Modelo diferencial

    - Síncrono: las ruedas se mueven al mismo instante, es un caso particular del diferencial. Las ruedas están ligadas de tal manera que siempre apuntan en la misma dirección y para girar, lo hacen sobre el eje vertical, dejando la estructura fija.

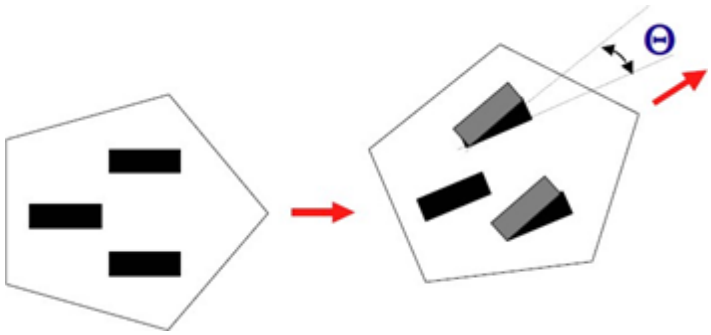


Figura 3-2. Modelo síncrono

- Triciclo: compuesto, normalmente, por dos ruedas de tracción y una tercera de dirección. Se caracterizan por disponer de buena estabilidad y simplicidad mecánica, pero su cinemática es más compleja.

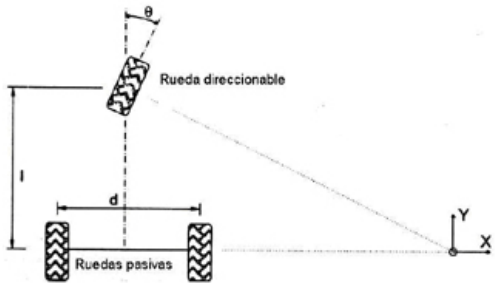


Figura 3-3. Modelo triciclo

También se puede usar la otra configuración posible, donde la única rueda de tracción es la delantera. Su cinemática resultará más simple, determinada por el número de pulsos de la rueda de tracción y la dirección de esta. Esta configuración presenta un problema, ya que cuando el robot se encuentra en subidas, su centro de gravedad tiende a alejarse de la rueda de motriz y provoca una pérdida de tracción.

- Ackerman: similares al triciclo, pero el modelo cuenta con dos ruedas de tracción y dos de dirección. El modelo Ackerman se utiliza casi exclusivamente en la industria del automóvil (ruedas traseras de tracción y dos ruedas de dirección delanteras), permitiendo que la rueda interior tenga un ángulo ligeramente más agudo que la exterior, evitando posibles derrapes.

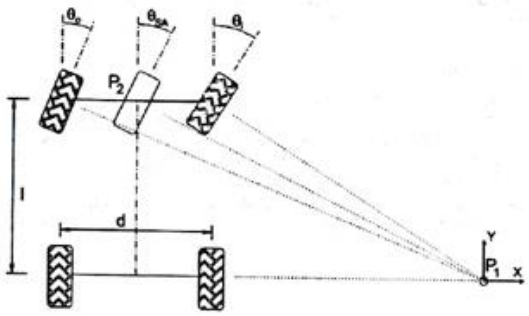


Figura 3-4. Modelo Ackerman

Una vez vistos los diferentes modelos que se pueden implementar, y limitados por los componentes que se poseían para la realización del proyecto, se optó por una composición de tipo diferencial con 4 ruedas motrices no direccionales.

- Motores:

  Son los componentes encargados de dotar de movimiento al robot. Se pueden clasificar según su forma de funcionamiento, siendo los más utilizados en robótica los siguientes:

  - Motores DC: este tipo de motor ofrece un movimiento lineal y rotatorio al aplicarle una diferencia de potencial en sus extremos.
  - Motores servo eléctricos: este motor es similar al motor DC, con la variación de que no ofrece un movimiento rotatorio, en su lugar tiene la capacidad de ubicarse en cualquier posición dentro de su rango de operación, y mantenerse estable en dicha posición.
  - Motores paso a paso: este motor funciona a través de los impulsos eléctricos que recibe, realizando desplazamientos angulares discretos, lo que significa que es capaz de avanzar una serie de grados dependiendo de sus entradas de control.

  Una vez vistos los diferentes modelos que se pueden implementar, y limitados por los componentes que se poseían para la realización del proyecto, se optó por 4 motores DC cada uno controlando una rueda diferente.

- Control de los motores:

  El consumo de los motores es significativamente mayor que el del resto de componentes electrónicos del robot. Por ello, es necesario disponer de una alimentación y control independientes del resto, los cuáles se pueden integrar mediante un puente en H. El puente H es un circuito electrónico que permite a un motor eléctrico DC girar en ambos sentidos, avanzar y retroceder.

  Los puentes H ya vienen hechos en algunos circuitos integrados, pero también se pueden construir a partir de componentes eléctricos y/o electrónicos.

  Un puente H se construye con 4 interruptores (mecánicos o mediante transistores). Cuando los interruptores S1 y S4 están cerrados (S2 y S3 abiertos) se aplica una tensión haciendo girar el motor en un sentido. Abriendo los interruptores S1 y S4 (cerrando S2 y S3), el voltaje se invierte, permitiendo el giro en sentido inverso del motor.

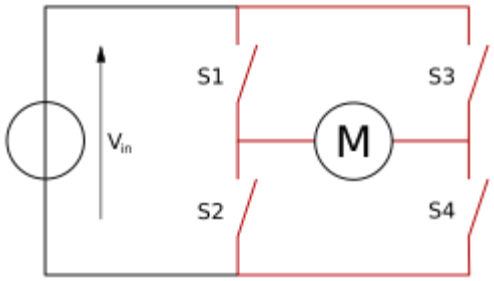


Figura 3-5. Puente en H

  Un puente H no solo se usa para invertir el giro de un motor, también se puede usar para frenarlo de manera brusca, al hacer un corto entre los bornes del motor, o incluso puede usarse para permitir que el motor frene bajo su propia inercia, cuando desconectamos el motor de la fuente que lo alimenta.

  En el caso de nuestro robot, el puente en H está integrado en el driver del propio motor.

- **Comunicación:**

  En caso de que el robot necesitase comunicarse con el exterior, ya sea para enviar datos o para ser controlado, se pueden utilizar distintas interfaces de comunicación:

  - Por cable:
    - Ethernet.
    - Serie.
    - Óptica.

  - Inalámbrica:
    - Wifi.
    - Bluetooth.
    - Radio Frecuencia.
    - Infrarrojos.

  Dado que nuestro robot es autónomo y que no es necesario extraer el mapa de la superficie una vez reconocida, no se necesita ningún tipo de interfaz de comunicación externo.

- **Control:**

  Para que el robot sea autónomo todos los componentes deben estar controlados por un microcontrolador, el cual se encarga de realizar las diferentes tareas en función de los datos recibidos por los sensores.

  En nuestro caso vamos a utilizar un DSP.

  Un DSP "Digital Signal Processor" [7] es un microprocesador con las siguientes características:

  - Capacidad de procesamiento de señales digitales en tiempo real. Los DSPs normalmente tienen que procesar datos en tiempo real. La exactitud de la operación depende en gran medida del tiempo en el que el procesamiento se completa.
  - Alta tasa de transferencia "high-throughput". Los DSPs soportan retransmisión de datos a alta velocidad como por ejemplo audio y multimedia.
  - Operación determinista. El tiempo de ejecución de los programas del DSP puede ser calculado con precisión, garantizando la repetibilidad del funcionamiento deseado.
  - Re-programabilidad vía software. El comportamiento del sistema puede variar según el código que se esté ejecutando en el DSP, sin necesidad de modificaciones en el hardware.

  El campo de uso y aplicación del DSP es enorme. Se emplean actualmente en todo tipo de dispositivos en aplicaciones de comunicaciones (banda ancha e inalámbricas), consumo (seguridad, entretenimiento, juguetes), industria (medicina, puntos de venta, automatismos industriales, sistemas de visión) y militar y aeroespacial (guiado, aviónica, radio digital, detección de blancos)

- **Alimentación:**

  Para que el robot funcione correctamente cada componente debe estar conectado a una fuente de alimentación que puede ser a través de cable (conexión a la red eléctrica) o mediante baterías.

  Por cable: la principal ventaja es que el robot no está limitado en el tiempo de trabajo. Sin embargo, se limita el alcance a la longitud del cable.

  Por baterías: la principal ventaja es que el alcance del robot es ilimitado. Sin embargo, su tiempo de funcionamiento depende de la capacidad de las baterías y del consumo del robot.

A continuación, se describen los componentes requeridos para el correcto funcionamiento de nuestro robot y alcanzar el objetivo de crear un sistema autónomo capaz de recorrer una habitación reconociendo los obstáculos y evitándolos, a la vez que se guarda internamente un mapa.

Para la construcción del robot se utilizan los siguientes componentes:

- Microprocesador DSP (incluyendo placa experimental)
- Sensor de ultrasonidos.
- Ruedas y motores (incluyendo puente en H y drivers)
- Giróscopo/Acelerómetro.

3.1 Principales Componentes

3.1.1 Microprocesador DSP TMS320F28335 y Kit de Experimentación

El microprocesador utilizado es el DSP TMS320F28335 de Texas Instruments [11].

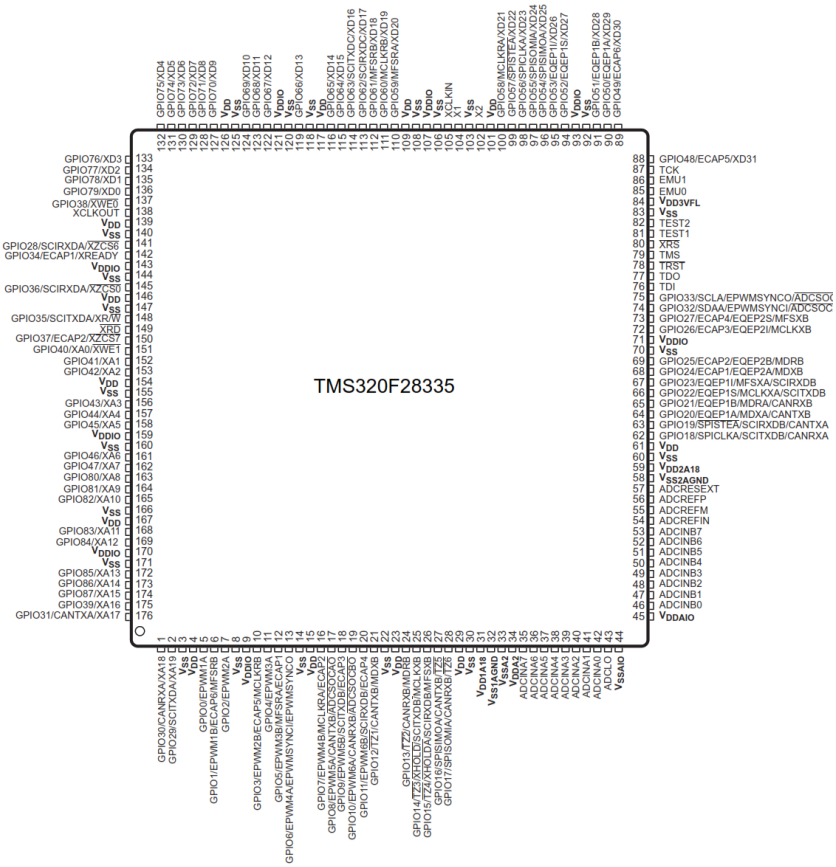


Figura 3-6. Diagrama de pines del DSP TMS320F8335

Sus principales características son las siguientes:

- Velocidad de operación 150 MHz
- CPU de 32 bits.
- Controlador DMA de seis canales.
- Interfaz externa de 16 o 32 bits.
- Memoria 256K x 16 Flash, 34K x 16 SARAM.
- 1K x 16 OTP ROM.
- BOOT ROM (8K x 16).
- Control del sistema y relojes.
- Pines I/O (GPIO0 a GPIO63).
- Expansión interrupciones de periféricos.
- Código de seguridad de 128 bits.
- 18 salidas PWM.
- 6 salidas HRPWM con 150ps de resolución MEP.

- 3 temporizadores de la CPU de 32 bits.
- 3 puertos periféricos.
- ADC de 12 bits con 16 canales.

Este DSP dispone de un amplio rango de periféricos que permiten su uso en numerosas aplicaciones.

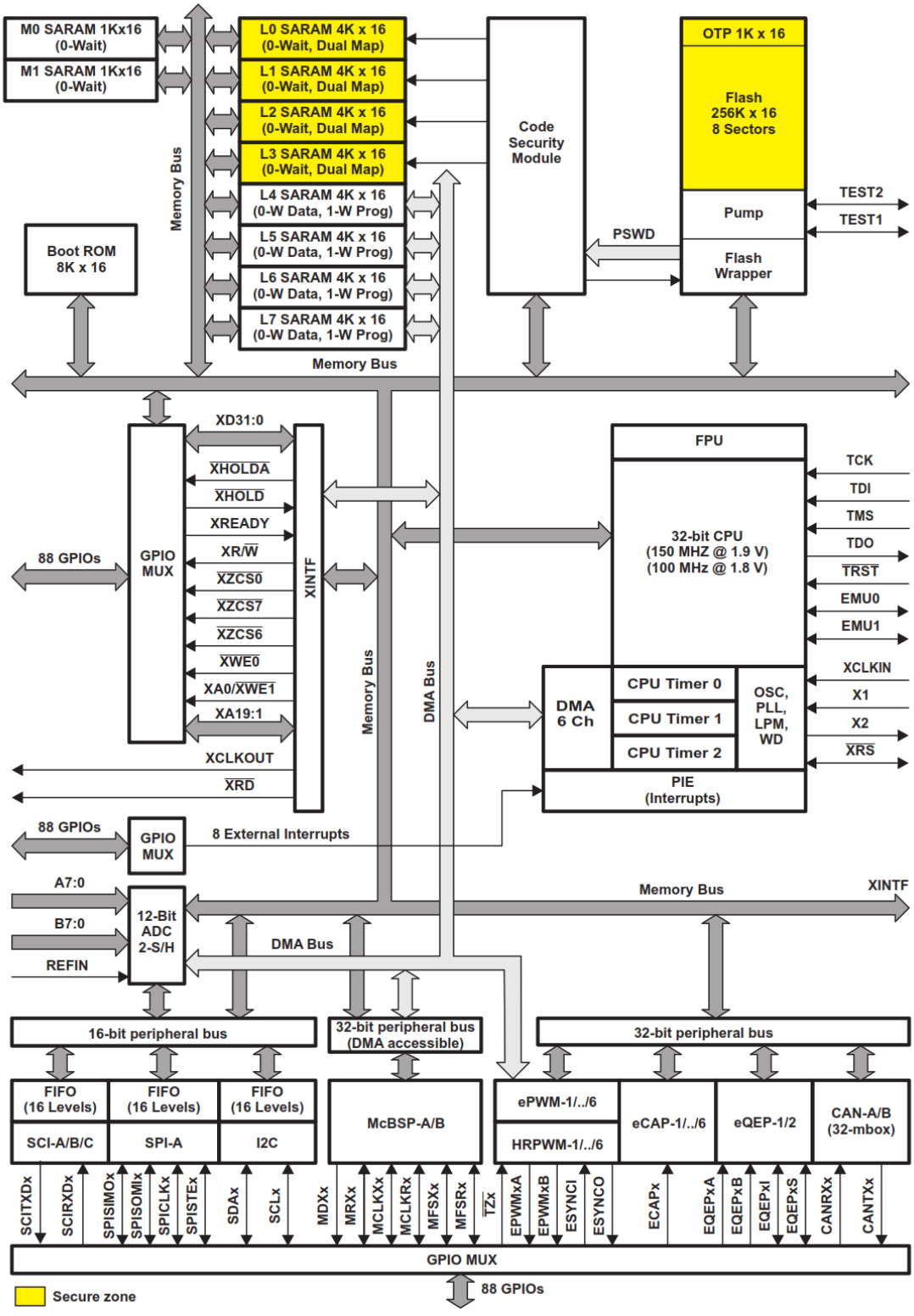


Figura 3-7. Diagrama de Bloques Funcional

El kit de experimentación utilizado es el TMDSDOCK28335 (Delfino F28335) [12].

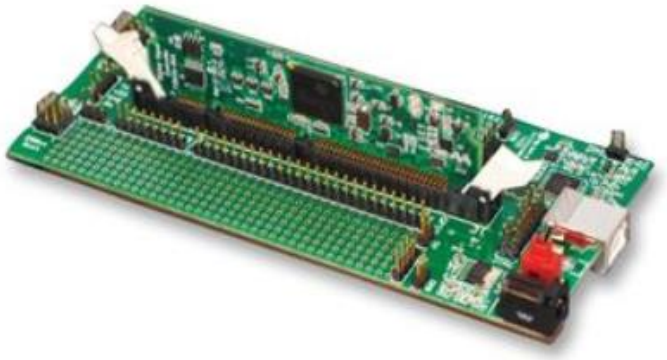


Figura 3-8. Kit experimentación Delfino F28335

El TMDSDOCK28335 es un kit de experimentación C2000 de Texas Instruments, un producto ideal para OEM destinado a la comprobación y exploración inicial de dispositivo. El kit de experimentación 28335 tiene una estación de conexión con acceso a todas las señales de controlCARD, áreas de la placa de pruebas, conector RS232, JTAG y emulación USB JTAG integrada. Cada kit contiene un controlCARD 28335. El controlCARD es un módulo completo a nivel de placa que utiliza un factor de forma DIMM estándar para proporcionar una solución de controlador de placa simple de bajo perfil.

- Módulo controlCARD basado en MCU TMS320F28335.
- Estación de conexión con emulación USB JTAG integrada.
- Soporta IDE Code Composer Studio v3.3 C28x gratuito de 32 Kbytes.
- Software de aplicaciones y código de ejemplo disponibles para descarga.
- Emulación USB integrada o capacidad de utilizar un emulador JTAG externo.
- Permite utilizar la fuente de alimentación USB de 5V o una fuente externa para alimentar la placa.
- Comunicaciones UART mediante puente de USB a UART integrado en placa.
- Puentes de arranque para todos los modos cubiertos por los dispositivos F280xx y F2833x.
- Suministro de 5,0V para área de prototipado.

Este DSP se programa con la herramienta que proporciona el propio fabricante (Texas Instruments) denominada Code Composer Studio (CCS), la cual incluye los “linkers” necesarios para trabajar con los distinstos tipos de micros de esta marca.

3.1.2 Sensor de Ultrasonidos HC-SR04

El sensor HC-SR04 es un sensor de distancia de bajo costo que utiliza ultrasonido para determinar la distancia de un objeto en un rango de 2 a 450 cm. Destaca por su pequeño tamaño, bajo consumo energético, buena precisión y excelente precio. El sensor HC-SR04 es el más utilizado dentro de los sensores de tipo ultrasonido, principalmente por la cantidad de información y proyectos disponibles en la web. De igual forma es el más empleado en proyectos de robótica como robots laberinto o sumo, y en proyectos de automatización como sistemas de medición de nivel o distancia [13].

Figura 3-9. Sensor ultrasónico HC-SR04

El sensor HC-SR04 posee dos transductores: un emisor y un receptor piezoeléctricos, además de la electrónica necesaria para su operación. El funcionamiento del sensor es el siguiente: el emisor piezoeléctrico emite 8 pulsos de ultrasonido(40KHz) luego de recibir la orden en el pin TRIG, las ondas de sonido viajan en el aire y rebotan al encontrar un objeto, el sonido de rebote es detectado por el receptor piezoeléctrico, luego el pin ECHO cambia a Alto (5V) por un tiempo igual al que demoró la onda desde que fue emitida hasta que fue detectada, el tiempo del pulso ECO es medido por el microcontrolador y así se puede calcular la distancia al objeto. El funcionamiento del sensor no se ve afectado por la luz solar o material de color negro (aunque los materiales blandos acústicamente como tela o lana pueden llegar a ser difíciles de detectar).

La distancia se puede calcular utilizando la siguiente formula:

*Distancia(m) = {(Tiempo del pulso ECO) * (Velocidad del sonido=340m/s)} / 2*

Sus principales características son las siguientes:

- Voltaje de Operación: 5V DC.
- Corriente de reposo: < 2mA.
- Corriente de trabajo: 15mA.
- Rango de medición: 2cm a 450cm.
- Precisión: +- 3mm.
- Ángulo de apertura: 15°.
- Frecuencia de ultrasonido: 40KHz.
- Duración mínima del pulso de disparo TRIG (nivel TTL): 10 µS.
- Duración del pulso ECO de salida (nivel TTL): 100-25000 µS.
- Dimensiones: 45mm x 20mm x 15mm.
- Tiempo mínimo de espera entre una medida y el inicio de otra 20ms (recomendable 50ms)

A continuación, indicamos el conexionado del sensor de ultrasonidos HC-SR04 al DSP TMS320F28335.

Pin DSP TMS320F28335	**Pin HC-SR04**
5 V DC	Vcc
GPIO4	Trig
GPIO58	Echo
GND	GND

Tabla 3–1 Conexionado del sensor de ultrasonidos

Debido a un error en el kit de experimentación TMDSDOCK28335 (Delfino F28335), no es posible alimentar a 5 V DC el sensor desde la propia placa del micro. Por este motivo, es necesario utilizar un circuito adaptador

de tensión, que proporcione adaptada a 5 V DC la entrada lógica del sensor (“trigger”), y la salida lógica (“echo”) a 3.3 V DC para adecuarla al nivel lógico del microprocesador.

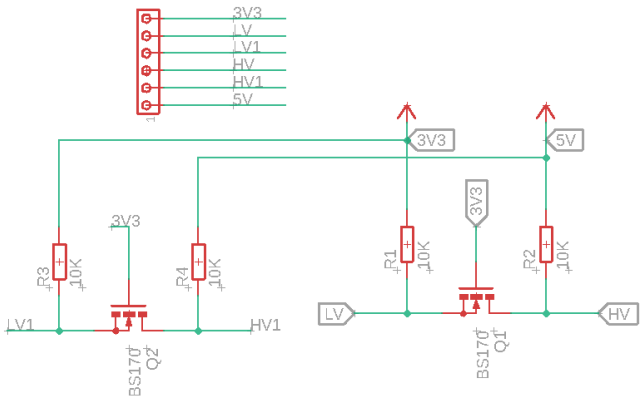


Figura 3-10. Esquemático circuito adaptador de 3.3 V a 5 V DC

El diseño del circuito se realizó con la ayuda del programa de diseño EAGLE [17], empleando transistores MOSFET BS170.

Además, fue necesario adquirir un elevador de tensión de rango 0.9 a 5V con salida de 5V y 480mA. Este elevador se conectó a la salida de 3.3 V DC del DSP y a la tierra lógica, y con la salida de 5 V DC se alimentó el sensor de ultrasonidos HC-SR04, además de dotar de la referencia de 5 V DC al circuito adaptador antes citado.

Figura 3-11. Elevador de tensión de 0.9 a 5 V con salida 5 V DC y 480 mA

3.1.3 Ruedas y Motores

A continuación, se describen los componentes usados para el desplazamiento del robot.

- **Ruedas:**

  Se utilizan 4 ruedas con un diámetro de 6.5 cm, que corresponden a 40.84 cm recorridos por cada vuelta completa.

Figura 3-12. Ruedas

- **Motores DC con Reductora:**

  Los motores utilizados en nuestro robot son "TT Geared Motor with Back Shaft (160rpm 6v L Shape)", de la marca DFRobot.

  Estos motores de pequeño tamaño incluyen una caja reductora que permite ejercer un mayor par a costa de una menor velocidad lineal, y sus características son las siguientes:

  - Rango de Voltaje de Operación: 3 ~ 7.5V.
  - Tensión nominal: 6V.
  - Max. Corriente en vacío (3V): 140 mA.
  - Max. Corriente en vacío (6V): 170 mA.
  - Velocidad en vacío (3V): 90 rpm.
  - Velocidad en vacío (6V): 160 rpm.
  - Max. Par de salida: 0.8 kgf/cm.
  - Max. Corriente de bloqueo: 2.8 A.

  Para controlar la velocidad de giro de los motores utilizaremos una señal PWM generada por el DSP y cuya frecuencia variará en función de la velocidad de giro deseada en cada uno de los motores.

Figura 3-13. Motor con caja reductora

- **Driver de los Motores:**

  Para el control y la alimentación de los motores utilizamos el Pololu DRV8835 Dual Motor Driver Kit for Raspberry Pi [14].

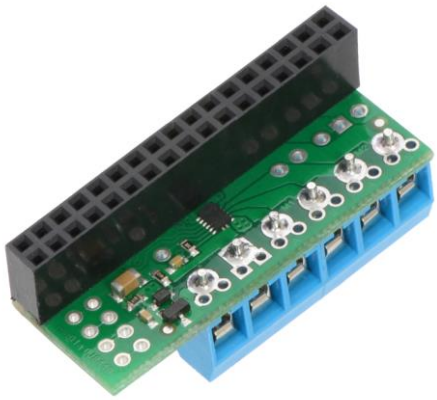


Figura 3-14. Driver Dual Pololu DRV8835

Las principales características del DRV8835 son:

- Controlador de motor de doble puente H: puede controlar dos motores DC o un motor paso a paso bipolar.
- Tensión de alimentación del motor: 1,5 V a 11 V.
- Tensión de alimentación lógica de 2 V a 7 V.
- Corriente de salida: 1,2 A continuos (1,5 A pico) por motor.
- Las salidas del motor pueden ser paralelas para entregar 2.4 A continuos (3 A pico) a un solo motor.
- Funcionamiento del PWM hasta 250 kHz (las frecuencias ultrasónicas permiten un funcionamiento más silencioso del motor)
- Dos modos de interfaz posibles: PHASE/ENABLE (por defecto - un pin para la dirección, otro para la velocidad) o IN/IN (salidas en su mayoría entradas espejo)
- La placa puede opcionalmente alimentar la base de frambuesa Pi directamente a través de un regulador adicional (no incluido).
- La librería Python facilita el uso de esta placa como placa de expansión de controladores de motor.
- Los mapeos de pines de GPIO pueden ser personalizados si los mapeos por defecto no son convenientes.
- Protección de tensión inversa en la alimentación del motor.
- Bloqueo de baja tensión y protección contra sobrecorriente y sobretemperatura.

Utilizamos 2 drivers para el control de los 4 motores. Cada driver controla 2 motores (uno los dos delanteros y otro los traseros)

A continuación, indicamos el conexionado del sensor de ultrasonidos HC-SR04 al DSP TMS320F28335 y a la batería auxiliar de 7.5 V DC.

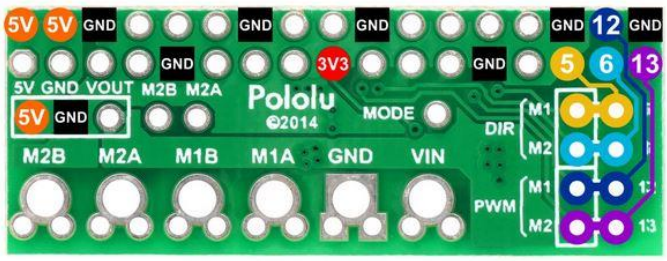


Figura 3-15. Pines del driver DRV8835

Pin DSP TMS320F28335	Pin DRV8835 (1)	Pin DRV8835 (2)	Batería 7.5 V DC	Motores Delanteros	Motores Traseros
3.3 V DC	3.3 V DC	3.3 V DC	---	---	---
GPIO0	12	---	---	---	---
GPIO1	13	---	---	---	---
GPIO2	---	12	---	---	---
GPIO3	---	13	---	---	---
GPIO10	5 y 6	5 y 6	---	---	---
GND	GND (conector negro)	GND (conector negro)	---	---	---
---	VIN	VIN	Positivo (+)	---	---
---	GND (conector azul)	GND (conector azul)	Negativo (-)	---	---
---	M1A y M1B	---	---	Izquierdo	---
---	M2A y M2B	---	---	Derecho	---
---	---	M1A y M1B	---	---	Izquierdo
---	---	M2A y M2B	---	---	Derecho

Tabla 3–2 Conexionado de los drivers de los motores

3.1.4 Giroscopio / Acelerómetro BMI088

El giroscopio / acelerómetro que utiliza nuestro robot es el BMI088 de Bosh [15]. BMI088 es un sensor inercial de 6 ejes de alto rendimiento que consta de un acelerómetro digital triaxial de 16 bits y un giroscopio digital triaxial de 16 bits y ±2000°/s. BMI088 permite medir con gran precisión la orientación y la detección de movimiento a lo largo de tres ejes ortogonales.

Con una alta resistencia a las vibraciones y un tamaño reducido de 3 x 4,5 x 0,95 mm³, se utiliza en entornos difíciles, como los de los aviones no tripulados y las aplicaciones robóticas. BMI088 está específicamente diseñado para suprimir eficazmente las vibraciones que puedan producirse debido a las resonancias en la PCB o en la estructura del sistema completo. Además de la alta resistencia a las vibraciones, la excelente estabilidad de temperatura del BMI088 ayuda a reducir el esfuerzo de diseño y los costes a nivel de sistema.

Figura 3-16. Placa base BMI088 Bosch

A continuación, se describen sus principales características:

- Resolución digital:
  - Acelerómetro (A): 16 bits
  - Giroscopio: 16 bits
- Resolución:
  - Acelerómetro: 0.09 mg
  - Giroscopio (G): 0.004°/s
- Rangos de medición programables.
- Temperatura de funcionamiento de entre -40º C y +85º C.
- Sensibilidad de calibrado.
- Anchos de banda programables de 5 Hz y 523 Hz.
- E/S digitales: SPI, I²C, 4x interrupciones digitales.
- Menos de 10 μA de consumo de corriente en modo de baja potencia.
- Paquete LGA de 16 pines, 3 mm x 4.5 mm x 0.95 mm.
- Voltaje de alimentación de 2.4 VDD a 3.6 VDD.
- Voltaje de alimentación de E/S de 1.2 VDD a 3.6 VDD.

A continuación, indicamos el conexionado del giroscopio BMI088 al DSP TMS320F28335.

Pin DSP TMS320F28335	**Pin BMI088**
3.3 V DC	VDD
3.3 V DC	VDDIO
GPIO32	SDA
GPIO33	SCL
GND	GND

Tabla 3–3 Conexionado del giroscopio

Uno de los criterios utilizados en la selección de este giroscopio fue que su rango de alimentación permitiese alimentar el sensor directamente desde la placa del micro a 3.3 V DC.

3.2 Montaje Final del Robot

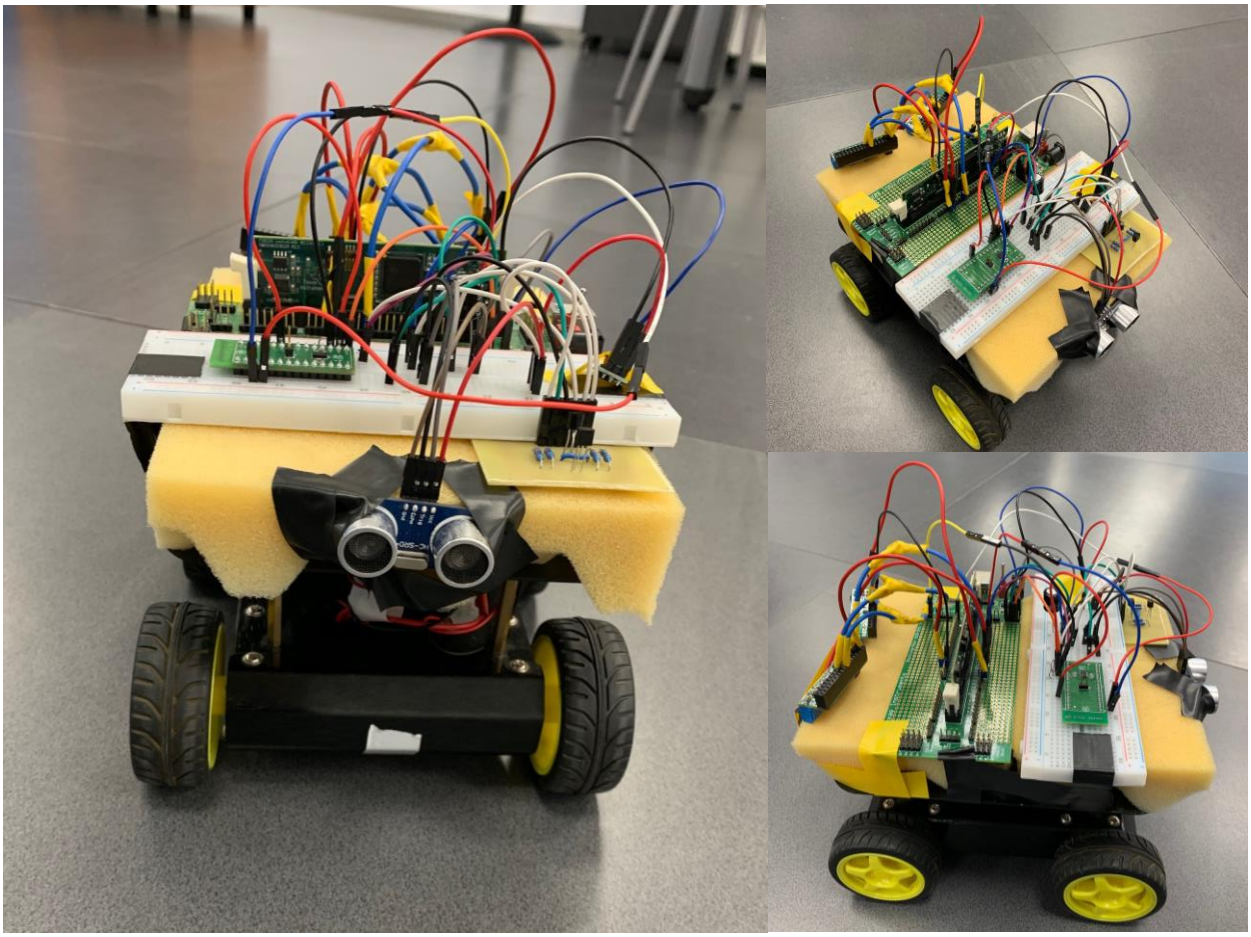


Figura 3-17. Montaje final del robot

El robot presenta una composición modular de 3 niveles.

- En la parte inferior están las 4 ruedas conectadas a sus respectivos motores DC. De los motores salen los cables que van a los drivers de los motores.
- En la parte central encontramos únicamente las baterías de alimentación.
- En la parte superior encontramos todos los componentes electrónicos incluida la “protoboard” que nos servirá para interconectar los siguientes componentes:
  - Giroscopio / acelerómetro.
  - Sensor de ultrasonidos.
  - Drivers de los motores.
  - Kit de experimentación del DSP.

4 PROGRAMACIÓN DEL ROBOT

Para que el microprocesador sea capaz de realizar las acciones deseadas (es decir, accionar los motores a la velocidad y dirección adecuadas, activar el sensor de ultrasonidos y calcular con éste la distancia al obstáculo más cercano en la dirección del movimiento, y leer las medidas del giroscopio para realizar los giros adecuados para maniobrar al acercarse a dicho obstáculo), debemos programar primeramente en nuestro ordenador las líneas de código necesarias.

En nuestro caso usaremos el programa Code Composer Studio (CCS) [16] de Texas Instruments para la programación del robot, el cual es un entorno de desarrollo integrado para desarrollar aplicaciones para procesadores embebidos de la misma marca. Los procesadores embebidos de Texas Instruments incluyen DSPs TMS320, sistemas en un chip OMAP, sistemas en un chip DaVinci, procesadores de aplicaciones Sitara, microcontroladores Hercules, microcontroladores Simplelink (MSP432 y microcontroladores de conectividad inalámbrica), MSP430 y microcontroladores Tiva/Stellaris. También permite la depuración en varios subsistemas como Ducati, IVA Accelerator y PRU-ICSS.

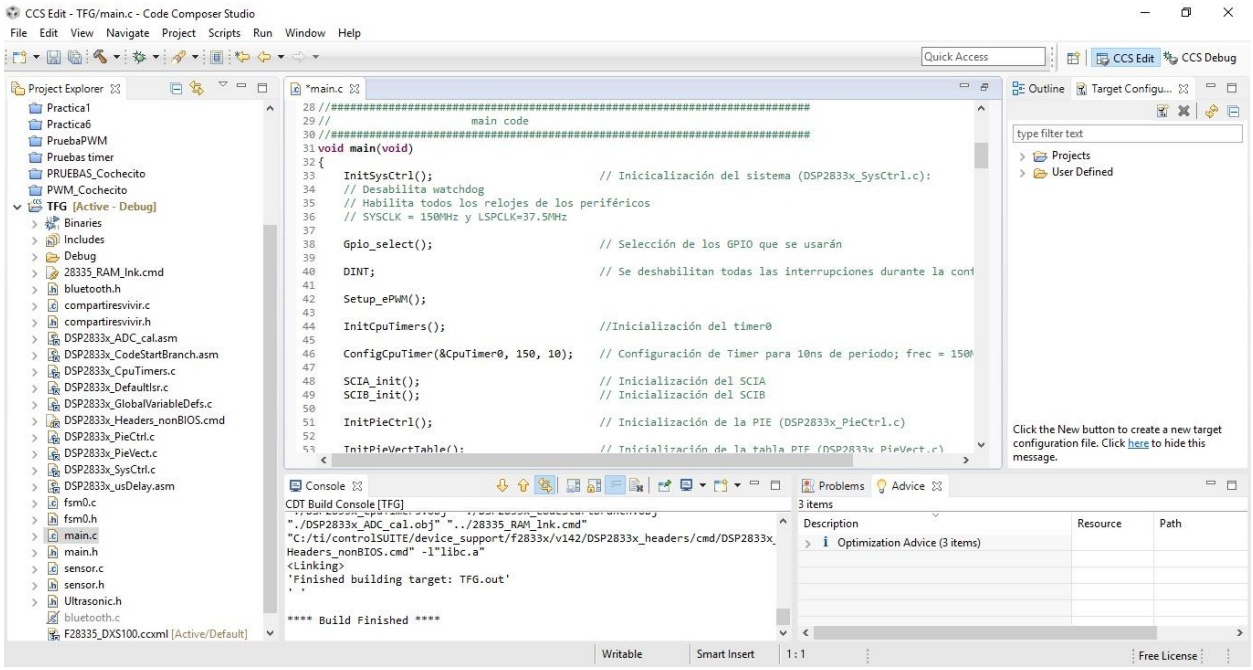


Figura 4-1. Captura de Pantalla de Code Composer Studio (CCS)

Existen numerosas versiones de este programa, pero usaremos la versión 6.0.0, la cual incluye por primera vez el soporte para los sistemas operativos Windows XP, Windows 7 y Windows 8.

Code Composer Studio presenta multitud de funciones de gran utilidad para la programación de nuestro DSP. Entre ellas cabe destacar la detección de errores y "warnings" y su explicación, así como la indicación del archivo y la línea de código en el que se encuentran, la posibilidad de comprobar los valores de las variables y de los distintos registros una vez ejecutado el programa en el microprocesador, y la capacidad de introducir puntos de ruptura en el código ("breakpoints") para encontrar errores y corregirlos cuando no existen incongruencias en el código detectables por el propio programa.

4.1 Programación de la Función Principal

En la función principal se configuran los relojes y los pines necesarios para la transmisión de las señales, poniendo '0' ó '1' en los diferentes registros que correspondan en el caso de los pines, y seleccionando una frecuencia y un periodo para los relojes. Además, se incluyen las llamadas a las funciones de inicio y de

funcionamiento de cada dispositivo conectado al DSP, las interrupciones necesarias, y también se programa el mapa de rejilla de ocupación.

Entre las interrupciones programadas en la función principal se encuentra interrupción por flanco en el "echo" del sensor, conectado al GPIO58 de nuestro DSP, en la cual se inicia la cuenta atrás del temporizador y activando la bandera cuando se detecta un flanco de subida de dicha señal, y en caso de que el flanco sea de bajada se para el temporizador y se pone la bandera a nivel bajo. La interrupción del propio temporizador también se incluye en este archivo, siendo necesaria para calcular el tiempo que debe transcurrir necesariamente entre cada medición del sensor de ultrasonidos.

4.1.1 Mapa de Rejilla de Ocupación

El mapa de rejilla de ocupación creado el robot sea autónomo se construye en función de la distancia medida por el sensor de ultrasonidos a los obstáculos presentes en la habitación, así como del giro medido por el giroscopio, el cual nos permitirá conocer en qué casilla de la matriz que conforma el mapa se encuentra el robot.

El método de mapeado con rejilla de ocupación se basa en estimaciones de la probabilidad de encontrar un obstáculo en una posición determinada. Con el fin de simplificar este método, se decidió utilizar como valores posibles 1, 0.5 y 0:

- Valor 1: la distancia medida por el sensor de ultrasonidos es lo suficientemente pequeña como para poder afirmar con total seguridad que el obstáculo está muy próximo. El robot evitará circular por estas casillas.
- Valor 0.5: este valor se utiliza como un paso intermedio entre el 1 y el 0 para garantizar la seguridad del robot y evitar posibles colisiones o cambios de velocidad muy bruscos. El sensor de ultrasonidos detecta un obstáculo a una distancia considerablemente pequeña, pero con un margen para continuar avanzando. El robot circulará por estas casillas a una velocidad reducida.
- Valor 0: la distancia medida por el sensor de ultrasonidos es lo suficientemente grande como para permitir el movimiento sin riesgo alguno de colisión. El robot circulará por estas casillas a la velocidad normal.

Conforme el robot avanza, varía la casilla de la matriz en la que se encuentra en ese instante. De esta forma, puesto que el robot mide constantemente la distancia a los obstáculos, si la distancia al obstáculo es mayor de 3 metros, se guardará un 0 en la casilla correspondiente; si la distancia a la que se encuentra es menor que 3 metros pero mayor que 1 metro, se guardará un 0.5 en la casilla de la matriz, indicando que el obstáculo se encuentra cerca y por tanto influyendo en el comportamiento de las señales PWM; y si la distancia es menor de 1 metro, en la próxima casilla de la matriz se escribirá un 1, siendo estas casillas intransitables por el robot, puesto que el obstáculo con total determinación se encuentra en dicha posición.

De esta forma, una vez que se haya rellenado completamente el mapa de rejilla de ocupación, el robot se podrá mover con total autonomía por la superficie reconocida, utilizando el sensor de ultrasonidos para conocer la distancia recorrida por el robot y por tanto la casilla en la que se encuentra, pero guiando su movimiento por el valor almacenado en todo momento en la próxima casilla de la matriz según el movimiento del robot.

4.2 Programación del Sensor de Ultrasonidos

En primer lugar, programamos el sensor HC-SR04 para su correcto funcionamiento. Para ello crearemos dos funciones dentro del archivo "sensor.c", ("sensor_init" y "sensor_do_cycle"), las cuales se usarán en la función principal para activar el sensor e iniciar la máquina de estados que lo controlará.

En dicha máquina de estados existen 5 estados diferentes:

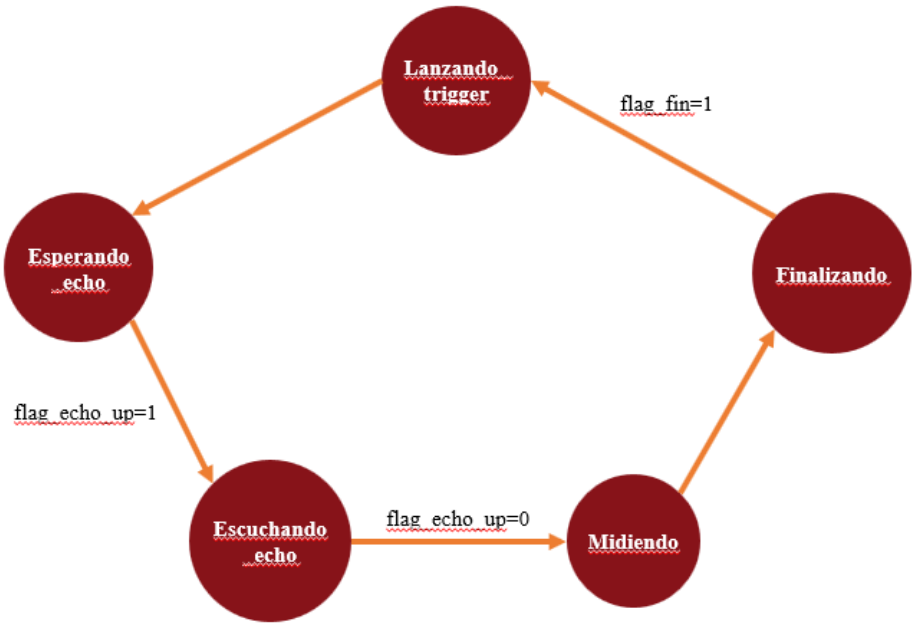


Figura 4-2. Máquina de estados del sensor de ultrasonidos

- Lanzando_trigger: en este estado producimos el flanco de 10 microsegundos en el "trigger" para iniciar el funcionamiento del sensor de ultrasonidos.
- Esperando_echo: este estado es transitorio, ya que en él únicamente esperamos a que el "echo" reciba un nivel alto, activándose la bandera, y pasamos al siguiente estado.
- Escuchando_echo: este estado es también transitorio, y en él esperamos a que desaparezca el nivel alto en el "echo", bajamos la bandera y pasamos al estado de medición.
- Midiendo: desde que aparece el nivel alto en el "echo" del sensor de ultrasonidos hasta que desaparece, se encuentra en funcionamiento un temporizador, cuyo tiempo medido se recoge. Esta medida pasa por diferentes operaciones hasta obtener la medición del tiempo del pulso "echo" a nivel alto en segundos, y dicho tiempo se utiliza para obtener la distancia al obstáculo utilizando la fórmula descrita en el capítulo 4.1.2.
- Finalizando: para evitar errores en el sensor, debe transcurrir un tiempo entre una medición y la siguiente, siendo esta la finalidad de este estado, tras el cual es vuelve a lanzar el "trigger" y se repite la operación.

```
170 static void midiendo (void)
171 {
172     medida = CpuTimer0Regs.TIM.all / 150;       // us
173     resta = (periodomax - medida);              // Duración del flanco en us
174     casi = (float)resta / 1000;                 // Cálculo intermedio
175     distancia =  casi *340 / 2 ;
176     distancia = distancia / 1000;               // Distancia en m
177
178     ConfigTimer0Fin(&CpuTimer0);            // Configuración de Timer para 60ms de periodo
179     CpuTimer0Regs.TCR.bit.TSS = 0;          // Se inicia la cuenta atrás del Timer0
180
181     estoy = finalizando;
182 }
```

Figura 4-3. Cálculo de la distancia al obstáculo

4.3 Programación del Giroscopio

El acelerómetro / giroscopio BMI088 tiene la posibilidad de comunicarse con el DSP mediante un protocolo I2C o un protocolo SPI. En nuestro caso usaremos el periférico de comunicación I2C [18] del microprocesador para recibir la información del giroscopio, por lo que usaremos dos pines de conexión únicamente, uno para la transmisión de datos y otro para la transmisión del reloj.

El bus I2C es un estándar que facilita la comunicación entre microcontroladores, memorias y otros dispositivos con cierto nivel de inteligencia, sólo requiere de dos líneas de señal y un común o masa. Fue diseñado a este efecto por Philips y permite el intercambio de información entre muchos dispositivos. La metodología de comunicación de datos del bus I2C 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) es la línea de los pulsos de reloj que sincronizan el sistema.
- SDA (System Data) es la línea por la que se mueven los datos entre los dispositivos.
- GND (Masa) común de la interconexión entre todos los dispositivos conectados al bus.

Puesto que este dispositivo integra un giroscopio y un acelerómetro, es necesario seleccionar el modo de funcionamiento normal para la función de giroscopio. De esta forma las lecturas obtenidas se corresponderán con la velocidad angular de giro del robot medidas en LSB, las cuales habrá que convertir mediante un factor de conversión el cual depende del rango programado en el giroscopio en grados por segundo (*º/s*) y, multiplicando por el tiempo desde que comenzó la maniobra, obtendremos el ángulo de giro.

Este periférico funcionará según la máquina de estado compuesta por los siguientes estados:

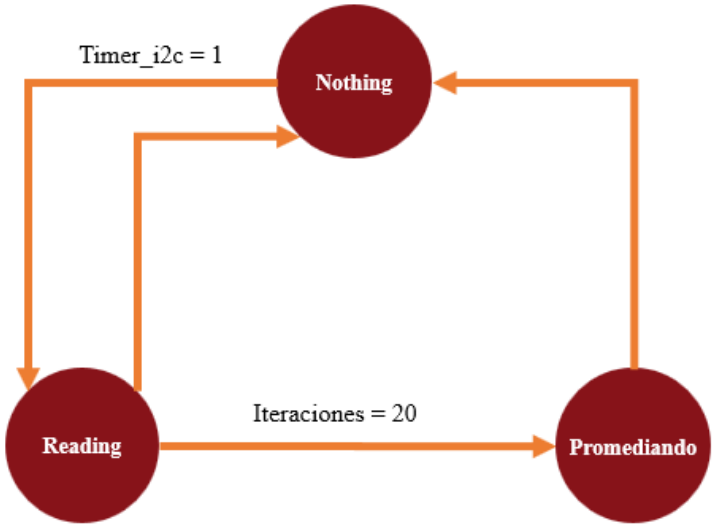


Figura 4-4. Máquina de estados del giroscopio

- Nothing: el dispositivo se mantiene a la espera de aparezca un nivel alto en la línea de transmisión de datos (SDA), lo cual significa que el robot está girando y necesitamos la información del giroscopio para realizar el giro, activando la bandera del temporizador del I2C y cambiando al siguiente estado.
- Reading: el DSP lee los datos del giróscopo/acelerómetro y los filtra para obtener las medidas deseadas. Cuando el número de muestreos (variable “iteraciones”) es igual a 20, se pasa al estado promediando. En caso contrario, se vuelve al estado anterior a la espera de recibir un dato nuevo.
- Promediando: al alcanzar los 20 muestreos, se realiza un promedio de estos para minimizar el error cometido en las mediciones por el giróscopo, ya que de no proceder así dicho error se acumularía en las mediciones posteriores. Una vez finalizado el promedio se pasa automáticamente al estado de espera.

```
80 void bmi088_Gyro_Reading(int *gyro_x, int *gyro_y, int *gyro_z){
81     *gyro_x = I2C_read_one_byte(RATE_X_MSB) * 256 + I2C_read_one_byte(RATE_X_LSB);
82     delay_loop(100);
83
84     *gyro_y = I2C_read_one_byte(RATE_Y_MSB) * 256 + I2C_read_one_byte(RATE_Y_LSB);
85     delay_loop(100);
86
87     *gyro_z = I2C_read_one_byte(RATE_Z_MSB) * 256 + I2C_read_one_byte(RATE_Z_LSB);
88     delay_loop(100);
89 }
90
91 void bmi088_Filtering(int accel_x, int accel_y, int accel_z, int gyro_x,  int gyro_y, int
92
93     accel_x = accel_x * pow(2,(ACC_RANGE + 1)) * 1.5;
94     accel_y = accel_y * pow(2,(ACC_RANGE + 1)) * 1.5;
95     accel_z = accel_z * pow(2,(ACC_RANGE + 1)) * 1.5;
96
97     gyro_x = gyro_x / 131.072;
98     gyro_y = gyro_y / 131.072;
99     gyro_z = gyro_z / 131.072;
100
101 }
```

Figura 4-5. Cálculo de la velocidad angular

4.4 Programación del Módulo PWM

Para el control de los motores se utilizan señales PWM (“Pulse-Width Modulation”) [20]. La modulación por ancho de pulsos de una señal o fuente de energía es una técnica en la que se modifica el ciclo de trabajo de una señal periódica (una senoidal o una cuadrada, por ejemplo), ya sea para transmitir información a través de un canal de comunicaciones o para controlar la cantidad de energía que se envía a una carga.

La construcción típica de un circuito PWM se lleva a cabo mediante un comparador con dos entradas y una salida. Una de las entradas se conecta a un oscilador de onda dientes de sierra, mientras que la otra queda disponible para la señal moduladora. En la salida la frecuencia es generalmente igual a la de la señal dientes de sierra y el ciclo de trabajo está en función de la portadora.

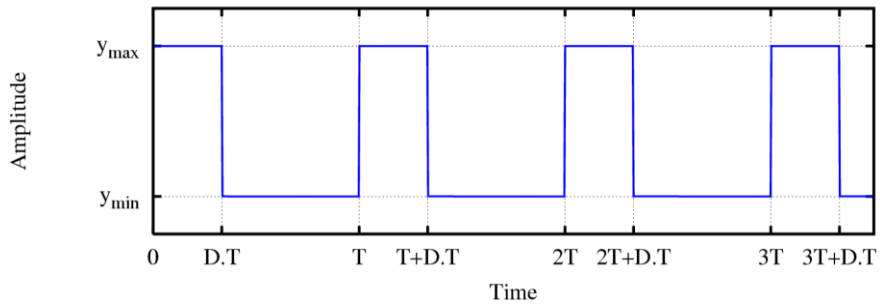


Figura 4-6. Modulación PWM

El DSP TMS320F28335 cuenta con dos módulos PWM que permiten generar dos señales PWM diferentes. Variando el tiempo de trabajo de cada una de estas señales se pueden llegar a obtener movimientos diversos movimientos y a diferentes velocidades.

Duty Cycle		Bit de Dirección	Estado	Descripción
Motores Izquierda	**Motores Derecha**			
0.25	0.25	0	Adelante	Marcha adelante a velocidad normal
0.15	0.15	0	Adelante_cuidado	Marcha adelante a velocidad reducida
0.35	0.1	0	Derecha	Giro a la derecha a velocidad constante
0.1	0.35	1	Atras	Marcha atrás

Tabla 4–1 Duty cycle módulo PWM

```
98 static void pwm_derecha(void)
99 {
100         GpioDataRegs.GPACLEAR.bit.GPIO10 = 1;
101         EPwm1Regs.CMPA.half.CMPA = EPwm1Regs.TBPRD*0.35;
102         EPwm1Regs.CMPB = EPwm1Regs.TBPRD*0.1;
103         EPwm2Regs.CMPA.half.CMPA = EPwm2Regs.TBPRD*0.35;
104         EPwm2Regs.CMPB = EPwm2Regs.TBPRD*0.1;
```

Figura 4-7. Programación PWM giro a la derecha

Para la programación del módulo PWM que controlará el movimiento de los motores, debemos crear dos máquinas de estado. La primera, que servirá para hacer el reconocimiento de la superficie en que se encuentre el robot, y en la que el sistema únicamente usará la distancia calculada al obstáculo para decidir el tiempo de trabajo ("duty cicle") de cada una de las 4 señales PWM que generamos; y la segunda, en la cual usaremos el mapa, que en nuestro caso será la rejilla de ocupación, para decidir qué deben hacer los motores del robot según en qué casilla de dicha rejilla se encuentre en cada instante.

En la primera máquina de estados, cuando no tenemos aún construido el mapa de la superficie, comenzamos circulando hacia delante siempre y cuando la distancia al obstáculo más cercano sea superior a 1.5 metros. Si la distancia medida es menor, el coche circulará hacia delante con menor velocidad, hasta encontrarse a 0.75 metros del obstáculo, y entonces girará hacia la derecha 90º. Si durante este giro en algún momento el sensor determina una distancia inferior a 0.3 metros a un obstáculo, el coche frenará y circulará marcha atrás hasta que dicha distancia sea de nuevo superior a 0.5 metros y así poder completar la maniobra sin riesgo de colisión.

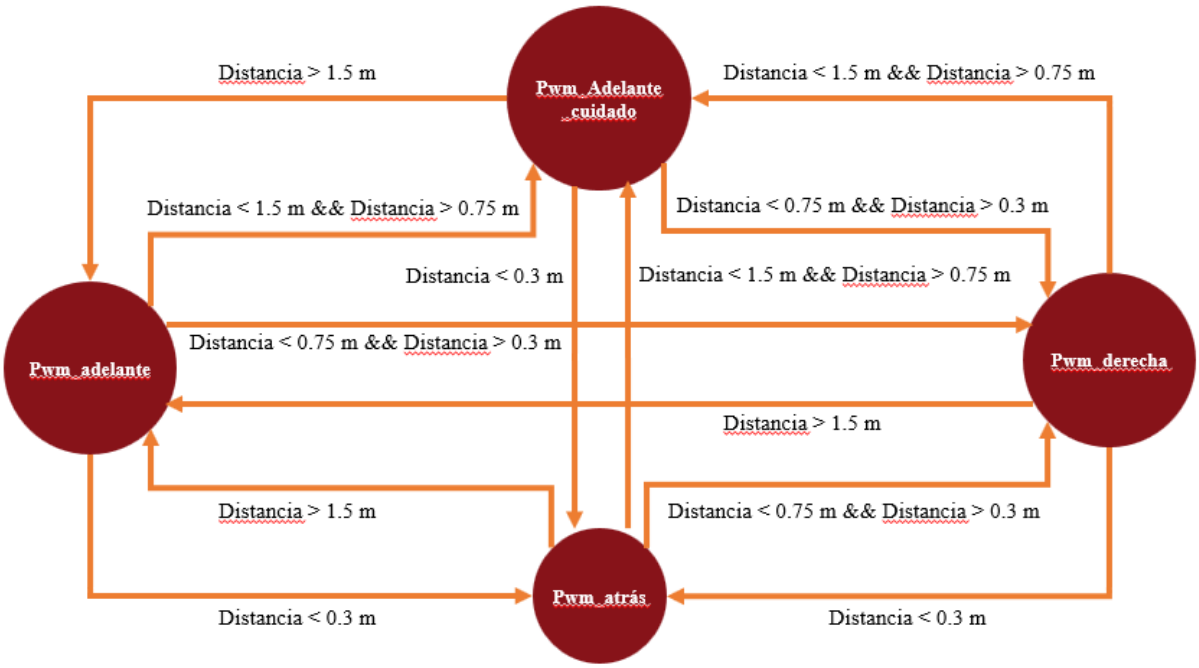


Figura 4-8. Primera máquina de estados PWM

En la segunda máquina de estados, la condición para pasar de un estado a otro pasa a ser el valor recogido en la siguiente casilla de la rejilla de ocupación que le correspondería al robot siguiendo la dirección de su movimiento. De esta forma el robot comienza circulando hacia delante hasta que la próxima casilla tenga un valor de 0.5, circulando entonces en la misma dirección, pero reduciendo la velocidad debido a que en el reconocimiento de la superficie se detectó un obstáculo a una distancia entre 3 metros y 1 metro de su posición actual. Mientras el robot se encuentre circulando con velocidad reducida, si la próxima casilla tuviera un valor de 0 se volvería a la velocidad normal. Seguidamente, si el valor la próxima casilla es de 1, el robot girará a la derecha para evitar el obstáculo. Durante el proceso de maniobra, si se detectase un valor de 1 en la próxima casilla retrocederíamos hasta ser dicho valor igual a 0.5, continuando entonces el giro hasta finalmente encontrar de nuevo un valor de 0 en la siguiente posición de la rejilla de ocupación, volviendo a circular hacia delante a la velocidad normal programada.

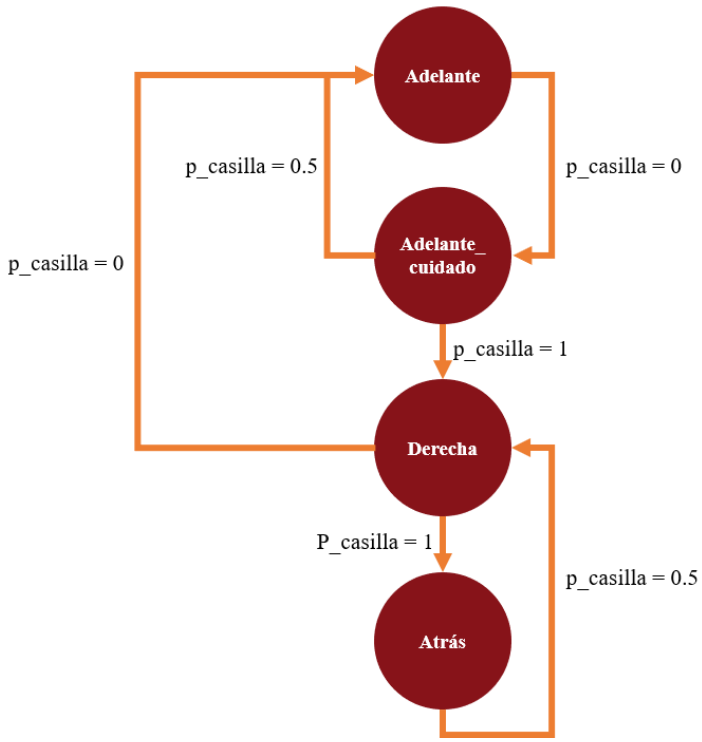


Figura 4-9. Segunda máquina de estados PWM

5 PRUEBAS DEL ROBOT

Antes de poder probar el robot con su montaje final, es necesario hacer varias pruebas con el osciloscopio y el voltímetro para comprobar que las señales que envía el DSP son las correctas, con el fin de evitar errores que podrían llegar a resultar fatales a la hora de probar el funcionamiento final.

5.1 Pruebas PWM

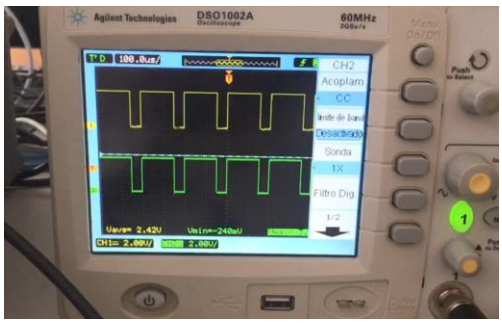
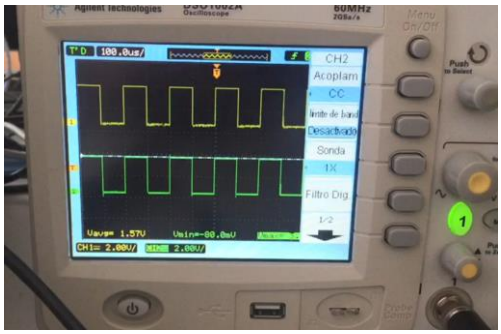
En primer lugar, se realizaron las pruebas de las señales PWM que se generan por los pines GPIO0, GPIO1, GPIO2 y GPIO3 de la placa de experimentación de nuestro DSP TMS320F28335. Para ello programamos un código de pruebas en el que, usando una función que crea un bucle para mantener el sistema inactivo durante un tiempo ("delay_loop"), se variaba el tiempo de trabajo de las señales PWM simulando un movimiento rectilíneo con velocidad alta y baja, y movimiento hacia la derecha y hacia la izquierda.

```
358 static void delay_loop(long int j)
359 {
360     long int i;
361     for(i=0;i<j;i++);
362 }
```

Figura 5-1. Función "delay_loop"

A continuación, una vez cargado el código en el DSP, se conecta la tierra del osciloscopio a la tierra del DSP y, usando dos sondas, analizamos en primer lugar las señales PWM1 generadas, comprobando que todo momento se cumpla el comportamiento antes descrito, y siendo la señal verde el PWM1A y la señal amarilla el PWM1B.

También se realizaron las mismas pruebas con el PWM2, siendo en este caso la señal verde el PWM2A y la señal amarilla el PWM2B, obteniendo también los resultados deseados.

Estado PWM	Señales en Osciloscopio
Adelante	
Adelante_cuidado	

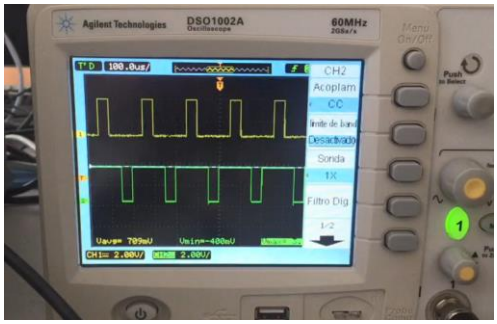
Estado PWM	Señales en Osciloscopio
Derecha	

Tabla 5–1 Pruebas PWM

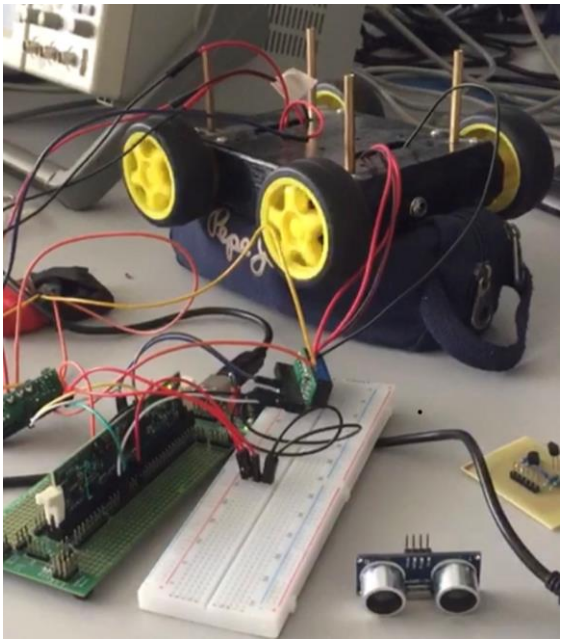


Figura 5-2. Pruebas de los motores

5.2 Pruebas Sensor Ultrasonidos

Para comprobar el correcto funcionamiento del sensor de ultrasonidos HC-SR04 se realizaron varias pruebas simples que consisten en el posicionamiento del sensor a distancias medidas con exactitud para comprobar la precisión del dispositivo y detectar los posibles errores que pudieran presentarse y el mismo error que presenta el sensor ya sea por interferencias en los ultrasonidos o por limitaciones de este.

Como se puede observar en las siguientes imágenes, el sensor mide con una precisión considerable las distancias al obstáculo más cercano (en las pruebas realizadas dicho obstáculo era una pared), y por ello es adecuado para su integración en el sistema, ya que el error que comete es lo suficientemente pequeño como para no provocar errores en el robot a la hora de moverse con autonomía y evitando cualquier colisión. Además, debe tenerse en cuenta que el sensor proporciona un pulso “echo” con una duración determinada, y el propio DSP introduce errores a la hora de calcular primero el tiempo que dicho pulso se mantiene a nivel alto y también en los cálculos posteriores de la distancia.

Expression	Type	Value
(x)= distancia	float	0.4836499

Figura 5-3. Distancia medida a 0.5 m

Expression	Type	Value
(x)= distancia	float	0.96968

Figura 5-4. Distancia medida a 1 m

Expression	Type	Value
(x)= distancia	float	1.43361

Figura 5-5. Distancia medida a 1.5 m

5.3 Pruebas Giroscopio

Para realizar las pruebas del giroscopio/acelerómetro BMI088 fue necesario en primer lugar comprobar el correcto funcionamiento de la máquina de estados de la comunicación I2C y de su correcta configuración. Tras esto, se conectó el dispositivo y se probaron distintos giros para comprobar que el sistema mide la velocidad angular correctamente.

A la hora de integrar el giroscopio en el robot, es necesario tener en cuenta la necesidad de activar un temporizador que controle la duración del giro, para así poder calcular el ángulo de giro a partir de los datos proporcionados por el dispositivo. Este temporizador lo proporciona el BMI088, siendo un contador de 24 bits dividido en 3 registros de 8 bits cada uno, e incrementando su valor cada registro al completar un ciclo el registro anterior, como si de un reloj normal se tratase.

Debido a la situación predeterminada de los ejes en el giroscopio, al girar hacia la derecha la velocidad angular medida es negativa, lo cual se debe tener en cuenta a la hora de integrar el dispositivo en el sistema.

Expression	Type	Value
(x)= v_angular	float	-8.380482
(x)= flag_adelante	int	0
(x)= time_adelante	float	5.0079
(x)= flag_giro	int	1
(x)= time_giro	float	5.8396
(x)= giroz	float	-11.3678
(x)= angulo	float	-48.93866

Figura 5-6. Ángulo de giro calculado y velocidad angular medida

5.4 Pruebas Integrales

Para las pruebas integrales del robot se ha recurrido a la división del proyecto en dos partes.

Esta división se debe a la poca memoria disponible para el código de programación que tiene el DSP TMS320F28335, ya que al integrar todos los archivos que componen el trabajo con sus correspondientes ficheros ".h" (también llamados "headers"), desbordamos dicha memoria y se nos muestra un error que impide la compilación del código y por tanto la implementación en el DSP.

- Movimiento con autonomía: se probó que el robot fuera capaz de desplazarse sin colisionar por un recinto gracias al sensor de ultrasonidos.

  Durante la realización de estas pruebas se encontró que al poner en funcionamiento el módulo PWM juntamente con el sensor de ultrasonidos, se detenía la máquina de estados del sensor debido a problemas con las interrupciones generados por el ruido producido por los cables y por la frecuencia y alto ciclo de trabajo de las señales PWM. Para detectar el problema se utilizó un osciloscopio, comprobando las señales "trigger" y "echo" del sensor HC-SR04 con el objetivo de descartar errores en el código. Una vez observado que el principal problema se basaba en los ruidos que se introducían en la señal del "echo" del sensor, se probó a filtrar esta señal con un condensador de 100 nF, el cual no se integró finalmente ya que generaba lecturas erróneas al suavizar demasiado los flancos de subida.

  Además, se decidió añadir un reset a la máquina de estado del sensor de ultrasonidos en caso de bloquearse de nuevo una vez solucionado el problema, y se comprobó que reduciendo el tiempo de trabajo de las señales PWM se reducían los ruidos del "echo", logrando así el correcto funcionamiento del sistema con el sensor y los motores conectados simultáneamente.

  Cabe destacar aun así que el sensor de ultrasonidos tiene dificultad para detectar obstáculos con una inclinación mayor de 15º, con lo que para lograr un mejor funcionamiento se debería integrar el giroscopio o en su defecto sensores adicionales.

- Giro controlado: para comprobar la posibilidad de integrar el giroscopio en el sistema se recurrió a probar su funcionamiento junto al módulo PWM y eliminando el sensor de ultrasonidos (debido a la falta de memoria de código del DSP). Para ello se modificó la máquina de estados del PWM, alternando el movimiento hacia delante con el giro a la derecha, de forma que el tiempo de giro dependiera del ángulo girado.

  El giroscopio BMI088 de Bosch únicamente proporciona la velocidad angular del robot, y no el ángulo de giro, por lo que fue necesario la programación de un temporizador para el cálculo de este. Por esto, ya que se excluyó el código del sensor de ultrasonidos, se utilizó el "timer 0" del DSP para medir, usando interrupciones internas, en primer lugar el tiempo que el robot debía avanzar hacia delante, y tras ello, mientras el robot se encontraba girando, se calculó el ángulo de giro en cada instante a partir de la lectura de velocidad angular del giroscopio y del temporizador, pudiendo comprobar así el correcto funcionamiento del giroscopio y que, en caso de disponer de memoria, hubiera sido posible su integración con el resto del sistema.

6 POSIBLES AMPLIACIONES

En este proyecto podrían haberse realizado numerosas ampliaciones con las que complementar la información recogida por el robot e incluso simplificar los algoritmos gracias a la mayor adquisición de datos, las cuales no se han incluido por falta de medios y por la falta de memoria en el microprocesador.

A continuación, se presentan varias posibles ampliaciones:

- **Utilización de encoders**: un encoder es un dispositivo electromecánico que permite codificar el movimiento mecánico en distintos tipos de impulsos eléctricos: digitales, binarios, analógicos en función de una onda, pulsos, etcétera. De este modo, un encoder es una interfaz entre un dispositivo mecánico móvil y un controlador.

  El encoder en este caso nos permitiría capturar la distancia recorrida en cada instante por las ruedas del robot, mejorando así la capacidad de localización de este en el entorno, y por consiguiente dentro del mapa de rejilla de ocupación. La incorporación de los encoders abriría por tanto un abanico de posibilidades a la hora de realizar el mapa de la superficie, ya que sin estos nos es imposible ubicar inicialmente el robot y se debe definir una posición inicial para poder comenzar el algoritmo.

- **Incorporación de un sensor lateral:** la integración de un segundo sensor en el lateral del robot permitiría aumentar la velocidad del mapeo, ayudando a reducir las veces que se debe realizar un recorrido para lograr que el valor de todas las casillas de la matriz de ocupación tenga un valor de 0 o 1. Con este sensor se aumentaría la eficiencia del robot, pero al igual que sucedería con la incorporación de los encoders, la memoria del DSP TMS320F28335 no es suficiente para sumar al código los ficheros y funciones resultantes de su implementación.

7 CONCLUSIONES

En este proyecto se ha estudiado y realizado el montaje y programación de un robot autónomo capaz de recorrer una superficie sin colisionar con obstáculos y de almacenar el mapa de la habitación o recinto en el que se encuentra.

Con todo lo desarrollado en este trabajo se ha podido llegar al objetivo deseado, aunque no sin lidiar con numerosas adversidades y contratiempos, como puede ser la amplia cantidad de documentación necesaria para la programación de los dispositivos que componen los sensores y actuadores del sistema, así como de las señales necesarias para su correcto funcionamiento, o la falta de memoria del microprocesador DSP TMS320F28335, que ha impedido la realización de pruebas íntegras del sistema, pudiendo únicamente probar su funcionamiento parcialmente. Así mismo, el disponer de todos los componentes necesarios y el correcto conexionado entre ellos y su programación, ambos absolutamente necesarios para la implementación del robot, ha resultado una tarea más compleja de lo esperado, debido a la necesidad de adquirir algunos componentes.

Este trabajo de fin de grado ha resultado para mí un reto personal de superación, puesto que en incontables ocasiones me he encontrado en situaciones adversas, teniendo que encontrar por mis propios medios soluciones, aunque siempre con la inestimable ayuda de mi tutor y de otros profesores. Además, considero que, por encima de la utilización de los conocimientos adquiridos a lo largo del grado, ha sido imprescindible una buena organización del tiempo, fuerza de voluntad y constancia, debido a la necesidad de compaginar el proyecto con las asignaturas del curso, lo cual incrementa aún más si cabe la dificultad de su realización.

Finalmente, el giroscopio/acelerómetro BMI088 que adquirí para este trabajo fin de grado los entrego al Departamento de Electrónica de la ETSI de la Universidad de Sevilla para su utilización en otros futuros proyectos.

REFERENCIAS

[1]	Barrientos A , Balaguer C, Felipe L y Aracil R. *Fundamentos de robótica 2ED.* McGraw-Hill, 2007. ISBN 9788448156367
[2]	*Leyenda del "Hombre de Palo"*. LeyendasdeToledo.com [consulta: 01 de junio de 2019]. Disponible en: https://www.leyendasdetoledo.com/leyenda-hombre-de-palo-infantil/
[3]	OLLERO BATURONE, A. *Robótica: Manipuladores y robots móviles 1ED*. Marcombo, 2005. ISBN 9788426713131
[4]	MORIELLO, S. Robots sociales, La Nueva Generación en Tendencias21. Revista Electrónica de Ciencia, Tecnología, Sociedad y Cultura. [consulta: 26 de mayo de 2019]. Disponible en: https://www.tendencias21.net/Robots-sociales-la-nueva-generacion_a2833.html
[5]	Honda-Robotics. Honda [consulta: 26 de mayo de 2019]. Disponible en: https://www.honda.mx/asimo/#
[6]	National Aeronautics and Space Administration. NASA. [consulta: 26 de mayo de 2019]. Disponible en https://robonaut.jsc.nasa.gov/R2/
[7]	ANGOLETTA, M.E. Digital signal procesor fundamentals and system desing. En: Digital Signa Processing CAS 2007: 31 de mayo. Ginebra, CERN, 2008, pp. 167-229
[8]	THRUN, Sebastian, et al. Robotic mapping: A survey. *Exploring artificial intelligence in the new millennium*, 2002, vol. 1, no 1-35, p. 1.
[9]	BIRK A, CARPIN S. Merging occupancy grid maps from multiple robots. En: *Proceedings of the IEEE.* 2006 Aug 21;94(7):1384-97.
[10]	ARJONA, David González. Generación automática de mapas en espacios cerrados mediante robots móviles. Escuela Politécnica Superior, Universidad Autónoma de Madrid, 2011.
[11]	Microprocesador DSP TMS320F28335. Texas Instruments [consulta: 30 de marzo de 2019]. Diponible en http://www.ti.com/product/TMS320F28335
[12]	Kit de experimentación TMDSDOCK28335. Texas Instruments [consulta: 30 de marzo de 2019]. Disponible en http://www.ti.com/tool/TMDSDOCK28335
[13]	Sensor ultrasonidos HC-SR04. Naylamp Mechatronics [consulta : 30 de marzo de 2019]. Disponible en https://naylampmechatronics.com/sensores-proximidad/10-sensor-ultrasonido-hc-sr04.html

[14]	Driver dual para motores DRV8835. Pololu [consulta: 30 de marzo de 2019]. Disponible en https://www.pololu.com/product/2753
[15]	Giróscopio / acelerómetro BMI088. Bosch Sensortec [consulta: 30 de marzo de 2019]. Disponible en https://www.bosch-sensortec.com/bst/products/all_products/bmi088_1
[16]	Code Composer Studio (CCS). Wikipedia La Enciclopedia Libre [consulta: 17 de junio de 2019]. Disponible en https://en.wikipedia.org/wiki/Code_Composer_Studio#Code_Composer_Studio
[17]	EAGLE. Autodesk [consulta: 17 de junio de 2019]. Disponible en https://www.autodesk.com/products/eagle/overview#
[18]	Fundamentos del protocolo I2C. TBem Robótica y Electrónica, ©2019 [consulta: 17 de junio de 2019]. Disponible en https://teslabem.com/nivel-intermedio/fundamentos-del-protocolo-i2c-aprende/#
[19]	BMI088 Data Sheet. Bosch Sensortec [consulta: 18 de junio de 2019]. Disponible en https://ae-bst.resource.bosch.com/media/_tech/media/datasheets/BST-BMI088-DS001.pdf
[20]	Modulación por Ancho de Pulsos. Wikipedia La Enciclopedia Libre [consulta: 19 de junio de 2019]. Disponible en https://es.wikipedia.org/wiki/Modulaci%C3%B3n_por_ancho_de_pulsos

GLOSARIO

UNE: Una Norma Española

AC: Alternate Current

DC: Direct Current

P: Potencia Activa

PWM: Pulse Width Modulation

ANEXO I: CÓDIGO FUNCIÓN PRINCIPAL

```
/* Includes ------------------------------------------------------------------*/
#include "main.h"

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

// Prototipado de funciones internas
static void Gpio_select(void);
static void PieConfig(void);
static void Setup_ePWM(void);

// Prototipado de funciones internas de INTERRUPCIÓN

interrupt void medir_echo (void);                  // Interrupción que indica flanco
de bajada de la línea ECHO
interrupt void cpu_timer0_isr (void);        // Interrupción del Timer0

//###########################################################################
//                                       main code
//###########################################################################
void main(void)
{
      InitSysCtrl();      // Inicicalización del sistema (DSP2833x_SysCtrl.c):
                          // Desabilita watchdog
                          // Habilita todos los relojes de los periféricos
                          // SYSCLK = 150MHz y LSPCLK=37.5MHz

      Gpio_select();      // Selección de los GPIO que se usarán

      DINT;               // Se deshabilitan todas las interrupciones durante la
configuración de las PIE

      Setup_ePWM();

      InitCpuTimers();    //Inicialización del timer0

      ConfigTimer0Echo (&CpuTimer0);  // Configuración de Timer para 10ns de
periodo; frec = 150MHz

      InitPieCtrl();      // Inicialización de la PIE (DSP2833x_PieCtrl.c)

      InitPieVectTable(); // Inicialización de la tabla PIE (DSP2833x_PieVect.c)

      PieConfig ();       // Modificación de la PIE para direccionar las rutinas de
interrupción

      EINT;               // Se habilita la interrupción global INTM

      sensor_init ();     // Inicializamos la máquina de estados del sensor de
ultrasonidos
```

```
	pwm_init();
	fsm_i2c_init ();

	while(1)
	{
		sensor_do_cycle ();
		pwm_do_cycle();
		fsm_i2c_cycle();
	}
}

static void Gpio_select(void)
{
	EALLOW;

	GpioCtrlRegs.GPBMUX1.bit.GPIO32 = 1;
	GpioCtrlRegs.GPBMUX1.bit.GPIO33 = 1;

	GpioCtrlRegs.GPBPUD.bit.GPIO32 = 0;
	GpioCtrlRegs.GPBPUD.bit.GPIO33 = 0;

	GpioCtrlRegs.GPBQSEL1.bit.GPIO32 = 3;
	GpioCtrlRegs.GPBQSEL1.bit.GPIO33 = 3;

	GpioCtrlRegs.GPAMUX1.bit.GPIO4 = 0;    // GPIO4 como TRIGGER --> sensor HC-
SR04   (GPIO I/O)
	GpioCtrlRegs.GPADIR.bit.GPIO4 = 1;     // GPIO4 como salida  (Out)

	GpioCtrlRegs.GPBMUX2.bit.GPIO58 = 0;   // GPIO58 como ECHO
	GpioCtrlRegs.GPBDIR.bit.GPIO58 = 0;    // GPIO58 como entrada  (In)
	GpioIntRegs.GPIOXINT3SEL.bit.GPIOSEL = 58;          // Línea 3 de interrupción
por GPIO (XINT3)

	GpioCtrlRegs.GPAMUX1.bit.GPIO0 = 1;          // Gpios de PWM
	GpioCtrlRegs.GPAMUX1.bit.GPIO1 = 1;
	GpioCtrlRegs.GPAMUX1.bit.GPIO2 = 1;
	GpioCtrlRegs.GPAMUX1.bit.GPIO3 = 1;
	GpioCtrlRegs.GPAMUX1.bit.GPIO10 = 0;
	GpioCtrlRegs.GPADIR.bit.GPIO10 = 1;

	EDIS;

	GpioDataRegs.GPACLEAR.bit.GPIO10 = 1;
}

static void PieConfig()
{
	EALLOW;

	PieVectTable.TINT0 = &cpu_timer0_isr;  // Se direcciona la rutina de
interrupción del Timer0

	PieVectTable.XINT3 = &medir_echo;      /* Se direcciona la rutina de
interrupción del ECHO por la línea de interrupción externa XINT3 */
	EDIS;

	PieCtrlRegs.PIEIER1.bit.INTx7 = 1;     // Se habilita la interrupción en la
PIE, INT1.7, fuente 7 (TINT0)
	PieCtrlRegs.PIEIER12.bit.INTx1 = 1;    // Se habilita la interrupción en la
PIE, INT12.1, fuente 1 (XINT3)
```

```
	XIntruptRegs.XINT3CR.bit.ENABLE = 1;   // Habilitamos la interrupción externa
XINT3
	XIntruptRegs.XINT3CR.bit.POLARITY = 3;  // Interrupción generada por flancos
de subida y bajada
	IER = 0x901;
}

// Definición de funciones de interrupción:

interrupt void medir_echo (void)       // Salta cuando se detecta un flanco en el
echo
{
	if (flag_echo_up == 0)          //Flanco de subida detectado
	{
		CpuTimer0Regs.TCR.bit.TSS = 0;  // Se inicia la cuenta atrás del Timer0
		flag_echo_up = 1;
	}
	else                                    //Flanco de bajada detectado
	{
		CpuTimer0Regs.TCR.bit.TSS = 1;  // Se detiene la cuenta atrás del
Timer0
		flag_echo_up = 0;               //Bajamos la bandera
	}

	PieCtrlRegs.PIEIFR12.bit.INTx1 = 0;
	PieCtrlRegs.PIEACK.bit.ACK12 = 1;
}

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

	PieCtrlRegs.PIEACK.bit.ACK1 = 1;
}

static void Setup_ePWM(void)
{
	//Configuracion modulo Time-Base
	//Pwm1
	EPwm1Regs.TBCTL.bit.CLKDIV = 0;
	EPwm1Regs.TBCTL.bit.HSPCLKDIV = 1;
	EPwm1Regs.TBPRD = 7500;
	EPwm1Regs.TBCTL.bit.CTRMODE = 2;
	EPwm1Regs.TBCTL.bit.PRDLD = 1;     //Un cambio en TBPRD se cambia directamente
	EPwm1Regs.TBCTL.bit.SYNCOSEL = 3;
	//Pwm2
	EPwm2Regs.TBCTL.bit.CLKDIV = 0;
	EPwm2Regs.TBCTL.bit.HSPCLKDIV = 1;
	EPwm2Regs.TBPRD = 7500;
	EPwm2Regs.TBCTL.bit.CTRMODE = 2;
	EPwm2Regs.TBCTL.bit.PRDLD = 1;     //Un cambio en TBPRD se cambia directamente
	EPwm2Regs.TBCTL.bit.SYNCOSEL = 3;

	// Configuracion del modulo Compare-Counter
	//Pwm1
	EPwm1Regs.CMPA.half.CMPA = EPwm1Regs.TBPRD*0.75;
	EPwm1Regs.CMPB = EPwm1Regs.TBPRD*0.75;
```

```
        EPwm1Regs.CMPCTL.bit.SHDWAMODE = 1;
        EPwm1Regs.CMPCTL.bit.SHDWBMODE = 1;
        EPwm1Regs.CMPCTL.bit.LOADAMODE = 0;
        EPwm1Regs.CMPCTL.bit.LOADBMODE = 0;
        //Pwm2
        EPwm2Regs.CMPA.half.CMPA = EPwm2Regs.TBPRD*0.75;
        EPwm2Regs.CMPB = EPwm2Regs.TBPRD*0.75;
        EPwm2Regs.CMPCTL.bit.SHDWAMODE = 1;
        EPwm2Regs.CMPCTL.bit.SHDWBMODE = 1;
        EPwm2Regs.CMPCTL.bit.LOADAMODE = 0;
        EPwm2Regs.CMPCTL.bit.LOADBMODE = 0;


        //Configuracion del modulo Action-Qualifier
        EPwm1Regs.AQCTLA.all = 0;
        EPwm1Regs.AQCTLA.bit.CAD = 2;        //EPWM1A a low en CMPA down
        EPwm1Regs.AQCTLA.bit.CAU = 1;        //EPWM1A a high en CMPA up
        EPwm1Regs.AQCTLB.all = 0;
        EPwm1Regs.AQCTLB.bit.CBD = 2;        //EPWM1B a low en CMPB down
        EPwm1Regs.AQCTLB.bit.CBU = 1;        //EPWM1B a high en CMPB up

        EPwm2Regs.AQCTLA.all = 0;
        EPwm2Regs.AQCTLA.bit.CAD = 2;        //EPWM2A a low en CMPA down
        EPwm2Regs.AQCTLA.bit.CAU = 1;        //EPWM2A a high en CMPA up
        EPwm2Regs.AQCTLB.all = 0;
        EPwm2Regs.AQCTLB.bit.CBD = 2;        //EPWM2B a low en CMPB down
        EPwm2Regs.AQCTLB.bit.CBU = 1;        //EPWM2B a high en CMPB up

        //Configuracion del modulo Dead Band
        EPwm1Regs.DBCTL.bit.OUT_MODE = 0;    //Modulo desactivado
        EPwm2Regs.DBCTL.bit.OUT_MODE = 0;

        //Configuracion del modulo PWM_Chopper
        EPwm1Regs.PCCTL.bit.CHPEN = 0;       //Modulo desactivado
        EPwm2Regs.PCCTL.bit.CHPEN = 0;

        //Configuracion del modulo Trip Zone
        EPwm1Regs.TZCTL.bit.TZA = 3;         //No hacer nada
        EPwm1Regs.TZCTL.bit.TZB = 3;         //No hacer nada
        EPwm1Regs.TZEINT.all = 0;            //Interrupcion deshabilitada
        EPwm2Regs.TZCTL.bit.TZA = 3;         //No hacer nada
        EPwm2Regs.TZCTL.bit.TZB = 3;         //No hacer nada
        EPwm2Regs.TZEINT.all = 0;            //Interrupcion deshabilitada

        //Configuracion del modulo Event Trigger
        EPwm1Regs.ETSEL.bit.SOCAEN = 0;      //Generacion SOCA deshabilitado
        EPwm1Regs.ETSEL.bit.SOCBEN = 0;      //Generacion SOCB deshabilitado
        EPwm1Regs.ETSEL.bit.INTEN = 0;       //Generacion de señal de interrupcion
deshabilitado
        EPwm2Regs.ETSEL.bit.SOCAEN = 0;
        EPwm2Regs.ETSEL.bit.SOCBEN = 0;
        EPwm2Regs.ETSEL.bit.INTEN = 0;
}


//===========================================================================
// End of SourceCode.
//===========================================================================
```

```
//HEADER MAIN.H

#ifndef MAIN_H_
#define MAIN_H_

#include "DSP2833x_Device.h"
#include <string.h>
#include <stdio.h>
#include "main.h"
#include "compartiresvivir.h"
#include "sensor.h"
#include "drivers.h"
#include "fsm_i2c.h"
#include "i2c.h"
#include "bmi088.h"


#endif /* MAIN_H_ */
```

ANEXO II: CÓDIGO MÓDULO PWM

```
#include "drivers.h"

static void (*state_func)(void);

static void pwm_reposo(void);
static void pwm_adelante(void);
static void pwm_adelante_cuidado(void);
static void pwm_derecha(void);
static void pwm_atras(void);
static void delay_loop(long);

void pwm_init(void)
{
	state_func = pwm_reposo;
}

void pwm_do_cycle(void)
{
	state_func();
}

static void pwm_reposo(void)
{
	GpioDataRegs.GPACLEAR.bit.GPIO10 = 1;
	EPwm1Regs.CMPA.half.CMPA = EPwm1Regs.TBPRD*0.0;
	EPwm1Regs.CMPB = EPwm1Regs.TBPRD*0.0;
	EPwm2Regs.CMPA.half.CMPA = EPwm2Regs.TBPRD*0.0;
	EPwm2Regs.CMPB = EPwm2Regs.TBPRD*0.0;

	delay_loop(100000);

	//TRANSICIÓN

	if (distancia > 1.5)
	{
		state_func = pwm_adelante;
	}
	else
	{
		if (distancia < 1.5 && distancia > 0.75)
		{
			state_func = pwm_adelante_cuidado;
		}
		else
		{
			if (distancia < 0.75 && distancia > 0.3)
			{
				state_func = pwm_derecha;
			}
			else
			{
				state_func = pwm_atras;
			}
		}
	}
}
```

```
static void pwm_adelante(void)
{
	GpioDataRegs.GPACLEAR.bit.GPIO10 = 1;
	EPwm1Regs.CMPA.half.CMPA = EPwm1Regs.TBPRD*0.75;
	EPwm1Regs.CMPB = EPwm1Regs.TBPRD*0.75;
	EPwm2Regs.CMPA.half.CMPA = EPwm2Regs.TBPRD*0.75;
	EPwm2Regs.CMPB = EPwm2Regs.TBPRD*0.75;

	if (distancia > 1.5)
	{
		state_func = pwm_adelante;
	}
	else
	{
		if (distancia < 1.5 && distancia > 0.75)
		{
			state_func = pwm_adelante_cuidado;
		}
		else
		{
			if (distancia < 0.75 && distancia > 0.3)
			{
				state_func = pwm_derecha;
			}
			else
			{
				state_func = pwm_atras;
			}
		}
	}
}

static void pwm_derecha(void)
{
	GpioDataRegs.GPACLEAR.bit.GPIO10 = 1;
	EPwm1Regs.CMPA.half.CMPA = EPwm1Regs.TBPRD*0.75;
	EPwm1Regs.CMPB = EPwm1Regs.TBPRD*0.25;
	EPwm2Regs.CMPA.half.CMPA = EPwm2Regs.TBPRD*0.75;
	EPwm2Regs.CMPB = EPwm2Regs.TBPRD*0.25;

	if (distancia > 1.5)
	{
		state_func = pwm_adelante;
	}
	else
	{
		if (distancia < 1.5 && distancia > 0.75)
		{
			state_func = pwm_adelante_cuidado;
		}
		else
		{
			if (distancia < 0.75 && distancia > 0.3)
			{
				state_func = pwm_derecha;
			}
			else
			{
```

```
                        state_func = pwm_atras;
                }
            }
        }
}

static void pwm_atras(void)
{
        GpioDataRegs.GPASET.bit.GPIO10 = 1;
        EPwm1Regs.CMPA.half.CMPA = EPwm1Regs.TBPRD*0.75;
        EPwm1Regs.CMPB = EPwm1Regs.TBPRD*0.75;
        EPwm2Regs.CMPA.half.CMPA = EPwm2Regs.TBPRD*0.75;
        EPwm2Regs.CMPB = EPwm2Regs.TBPRD*0.75;

        if (distancia > 1.5)
        {
            state_func = pwm_adelante;
        }
        else
        {
            if (distancia < 1.5 && distancia > 0.75)
            {
                state_func = pwm_adelante_cuidado;
            }
            else
            {
                if (distancia < 0.75 && distancia > 0.3)
                {
                    state_func = pwm_derecha;
                }
                else
                {
                    state_func = pwm_atras;
                }
            }
        }
}

static void pwm_adelante_cuidado(void)
{
        GpioDataRegs.GPACLEAR.bit.GPIO10 = 1;
        EPwm1Regs.CMPA.half.CMPA = EPwm1Regs.TBPRD*0.50;
        EPwm1Regs.CMPB = EPwm1Regs.TBPRD*0.50;
        EPwm2Regs.CMPA.half.CMPA = EPwm2Regs.TBPRD*0.50;
        EPwm2Regs.CMPB = EPwm2Regs.TBPRD*0.50;

        if (distancia > 1.5)
        {
            state_func = pwm_adelante;
        }
        else
        {
            if (distancia < 1.5 && distancia > 0.75)
            {
                state_func = pwm_adelante_cuidado;
            }
            else
            {
                if (distancia < 0.75 && distancia > 0.3)
                {
```

```
                        state_func = pwm_derecha;
                    }
                    else
                    {
                        state_func = pwm_atras;
                    }
                }
        }
}

static void delay_loop(long end)
{
        long i;
        for (i = end; i > 0 ; i--){
                asm(" NOP");
        }
}
```

```
//HEADER DRIVERS.H

#ifndef DRIVERS_H_
#define DRIVERS_H_

#include "DSP2833x_Device.h"
#include "compartiresvivir.h"

void pwm_init(void);
void pwm_do_cycle(void);

#endif /* DRIVERS_H_ */
```

ANEXO III: CÓDIGO SENSOR ULTRASONIDOS

```
/* Principio de funcionamiento:
 *
 *                 En primer lugar se pone a  nivel alto el trigger y se activa la
cuenta atrás del timer0, que
 *           contará 10us y activará la bandera del timer0 mediante su propia
interrupción. Cuando esto ocurra,
 *           bajaremos a 0 el trigger de nuevo, con lo que se enviarán las ondas
ultrasonidos y se recibirá el echo
 *         correspondiente. El echo pondrá a nivel alto TTL la entrada GPIO58 a la
que está conectado, activando
 *         la interrupción externa XINT3.En ese instante, activamos de nuevo el
timer, contabilizando hasta que
 *         baje el echo de nuevo.
 *
 *         El intervalo de tiempo medido(us) será dividido por 58 para obtener la
distancia en cm.
 *
 *         OJO: la configuración del timer0 para el TRIGGER será distinta que para la
escucha del ECHO:
 *
 *           -> TRIGGER: interrupción a los 10us
 *           -> ECHO:   1. Deshabilitamos interrupción
 *                           2. Cambiamos periodo
 */

// Includes

#include "sensor.h"       // Fichero con las variables necesarias en estas funciones
                          // y declaración de las funciones a compartir.

// Declaración de funciones privadas
void Delay_loop(int);

static void ( *estoy) (void);     // Función de estado del SENSOR

static void lanzando_trigger (void);   // Esta acción finaliza cuando la interrupción
del timer 0 activa la bandera
                  //(cuando han transcurrido 10ns del trigger a nivel alto)
static void esperando_echo (void);
static void escuchando_echo (void);
static void midiendo (void);    // En este estado inicializamos el contador del ECHO
para medir la distancia
static void finalizando (void);

// Declaración de variables privadas

int flag_comprobacion = 0;
float casi = 0;
long int resta = 0;
long int medida = 0;
```

```
//#########################################################
// Definición de funciones compartidas con el main
void ConfigTimer0Echo (struct CPUTIMER_VARS *Timer)       // Configuración del timer
para medida del echo (sensor)
{
    Uint32  temp;
    // Initialize timer period

    Timer->CPUFreqInMHz = 150;
    Timer->PeriodInUSec = periodomax;
    temp = (long) (150*periodomax);
    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;

    // Reset interrupt counter

    Timer->InterruptCount = 0;
}

void ConfigTimer0Fin (struct CPUTIMER_VARS *Timer)        // Configuración del timer
para finalizar el ciclo de medida
{
    Uint32  temp;

    // Initialize timer period

    Timer->PeriodInUSec = 60000;
    temp = (long) (150*60000);
    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 = 1;

    // Reset interrupt counter

    Timer->InterruptCount = 0;
}
void sensor_init (void)
{
      // Inicializamos las bandera de la interrupción
      flag_echo_up = 0;
      flag_fin = 0;

      estoy = lanzando_trigger;             // Comienza la secuencia del sensor
}

void sensor_do_cycle (void)
{
      (*estoy) ();
}

// Definición de funciones privadas
void Delay_loop(int us)                     // Retraso de us
{
      long ciclos = 150 * us;
      long i;
      for(i=0 ; i<ciclos ; i++)
      {
            asm ("NOP");
      }
}

static void lanzando_trigger (void)
{
      //Esta función sólo se ejecuta una vez cada secuencia de medida

      GpioDataRegs.GPACLEAR.bit.GPIO4 = 1;   //Nos aseguramos de que el trigger está
a nivel bajo
      GpioDataRegs.GPASET.bit.GPIO4 = 1;     // Ponemos el TRIGGER  a nivel alto
      Delay_loop (10);                                          // Flanco de 10us
para el trigger
      GpioDataRegs.GPACLEAR.bit.GPIO4 = 1;

      estoy = esperando_echo;                       // Cambia el estado
inmediatamente!
}
static void esperando_echo (void)
{
      if (flag_echo_up == 1)
      {
            estoy = escuchando_echo;
      }
}
```

```
static void escuchando_echo (void)
{

	if (flag_echo_up == 0)
	{
		estoy = midiendo;
	}
}


static void midiendo (void)
{
	medida = CpuTimer0Regs.TIM.all / 150;		// us
	resta = (periodomax - medida);			// Duración del flanco en us
	casi = (float)resta / 1000;			// Cálculo intermedio
	distancia =  casi *340 / 2 ;
	distancia = distancia / 1000;			// Distancia en m

	ConfigTimer0Fin(&CpuTimer0);			// Configuración de Timer para
60ms de periodo
	CpuTimer0Regs.TCR.bit.TSS = 0;			// Se inicia la cuenta atrás del
Timer0

	estoy = finalizando;
}

static void finalizando (void)		// Espera de 60ms hasta realizar una nueva
medida, evitando así errores (ver DATASHEET)
{
	if (flag_fin == 1)
	{
		flag_fin = 0;				// Reseteo bandera
		ConfigTimer0Echo (&CpuTimer0);		// Configuración de Timer para
periodomax sin interrupción

		estoy = lanzando_trigger;	// Vuelta a empezar...
	}
}
```

```
//HEADER SENSOR.H

#ifndef SENSOR_H_
#define SENSOR_H_

#include "DSP2833x_Device.h"
#include "compartiresvivir.h"
#include <string.h>
#include <stdio.h>

void ConfigTimer0Echo (struct CPUTIMER_VARS *);           // Configuración del timer
para medida del echo (sensor)
void ConfigTimer0Fin (struct CPUTIMER_VARS *);            // Configuración del timer
para finalizar el ciclo de medida
void sensor_init (void);
void sensor_do_cycle (void);


#endif /* SENSOR_H_ */
```

ANEXO IV: CÓDIGO GIROSCOPIO

```
//GIROSCOPIO//

#include "bmi088.h"

void bmi088_reg_conf(void){
	//escritura de los registros que queremos tocar
	I2C_write_one_byte(0x68, GYRO_RANGE, 0x03); // Range +-250 degrees/s
	delay_loop(2);
	I2C_write_one_byte(0x18, ACC_PWR_CTRL, 0x00);
	delay_loop(2);
	I2C_write_one_byte(0x18, ACC_PWR_CONF, 0x03);
	delay_loop(2);
	I2C_write_one_byte(0x68, GYRO_BANDWIDTH, 0x03);
	delay_loop(2);
	I2C_write_one_byte(0x68, GYRO_LMP1, 0x00);
	delay_loop(2);
	I2C_write_one_byte(0x18, ACC_CONF, 0xA8);
	delay_loop(2);
	I2C_write_one_byte(0x18, ACC_RANGE, 0x00); // Range +-2g
	delay_loop(2);
}

Uint16 bmi088_test_connection(void){
	//	Send START and set pointer to temperature - register
	I2caRegs.I2CCNT     = 1;
	I2caRegs.I2CDXR           = GYRO_CHIP_ID;
	// Send start as master transmitter
		I2caRegs.I2CMDR.all = 0x6660;
	/*	Bit15 = 0;   no NACK in receiver mode
		Bit14 = 1;  FREE on emulation halt
		Bit13 = 1;  STT  generate START
		Bit12 = 0;   reserved
		Bit11 = 0;  STP  not generate STOP
		Bit10 = 1;  MST  master mode
		Bit9  = 1;  TRX  master - transmitter mode
		Bit8  = 0;   XA   7 bit address mode
		Bit7  = 0;  RM   nonrepeat mode, I2CCNT determines # of bytes
		Bit6  = 1;  DLB  no loopback mode
		Bit5  = 1;  IRS  I2C module enabled
		Bit4  = 0;  STB  no start byte mode
		Bit3  = 0;  FDF  no free data format
		Bit2-0: 0;  BC   8 bit per data byte   */
	while(I2caRegs.I2CSTR.bit.ARDY == 0);              // wait for Access ready
condition

	I2caRegs.I2CCNT     = 1;
	I2caRegs.I2CMDR.all = 0x6C20;
	/*	Bit15 = 0;   no NACK in receiver mode
		Bit14 = 1;  FREE on emulation halt
		Bit13 = 1;  STT  generate START
		Bit12 = 0;   reserved
		Bit11 = 1;  STP  generate STOP
		Bit10 = 1;  MST  master mode
		Bit9  = 0;  TRX  master - receiver mode
		Bit8  = 0;   XA   7 bit address mode
		Bit7  = 0;  RM   nonrepeat mode, I2CCNT determines # of bytes
```

```
            Bit6   = 0;  DLB  no loopback mode
            Bit5   = 1;  IRS  I2C module enabled
            Bit4   = 0;  STB  no start byte mode
            Bit3   = 0;  FDF  no free data format
            Bit2-0: 0;  BC   8 bit per data byte   */

      while(I2caRegs.I2CSTR.bit.RRDY == 0);
      while(I2caRegs.I2CSTR.bit.SCD == 0);               // wait for STOP condition
      I2caRegs.I2CSTR.bit.SCD = 1;
      return(I2caRegs.I2CDRR);
}

void bmi088_Accel_Reading(int *accel_x, int *accel_y, int *accel_z)
{
      *accel_x = I2C_read_one_byte(0x18, ACC_X_MSB) * 256 + I2C_read_one_byte(0x18,
ACC_X_LSB);
      delay_loop(100);

      *accel_y = I2C_read_one_byte(0x18, ACC_Y_MSB) * 256 + I2C_read_one_byte(0x18,
ACC_Y_LSB);
      delay_loop(100);

      *accel_z = I2C_read_one_byte(0x18, ACC_Z_MSB) * 256 + I2C_read_one_byte(0x18,
ACC_Z_LSB);
      delay_loop(100);
}

void bmi088_Gyro_Reading(int *gyro_x, int *gyro_y, int *gyro_z)
{
      *gyro_x = I2C_read_one_byte(0x68, RATE_X_MSB) * 256 + I2C_read_one_byte(0x68,
RATE_X_LSB);
      delay_loop(100);

      *gyro_y = I2C_read_one_byte(0x68, RATE_Y_MSB) * 256 + I2C_read_one_byte(0x68,
RATE_Y_LSB);
      delay_loop(100);

      *gyro_z = I2C_read_one_byte(0x68, RATE_Z_MSB) * 256 + I2C_read_one_byte(0x68,
RATE_Z_LSB);
      delay_loop(100);
}

void bmi088_Filtering(int accel_x, int accel_y, int accel_z, int gyro_x,  int gyro_y,
int gyro_z)
{
      accel_x = accel_x * pow(2,(ACC_RANGE + 1)) * 1.5;
      accel_y = accel_y * pow(2,(ACC_RANGE + 1)) * 1.5;
      accel_z = accel_z * pow(2,(ACC_RANGE + 1)) * 1.5;

      girox = (float)gyro_x / 131.072;
      giroy = (float)gyro_y / 131.072;
      giroz = (float)gyro_z / 131.072;

}

void bmi088_init(void)
{
      int version;
```

```
        bmi088_reg_conf();
        version=bmi088_test_connection();
        delay_loop(100);
        version=I2C_read_one_byte(0x68, GYRO_BANDWIDTH);
        delay_loop(100);
        version=I2C_read_one_byte(0x68, GYRO_RANGE);
        delay_loop(100);
        version=I2C_read_one_byte(0x18, ACC_PWR_CTRL);
        delay_loop(100);
        version=I2C_read_one_byte(0x18, ACC_PWR_CONF);
        delay_loop(100);
}
```

```
//MAQUINA DE ESTADOS I2C//

#include "fsm_i2c.h"

static void nothing (void);
static void reading (void);
static void promediado (void);
static void (*estado_i2c)(void);

int iteraciones = 0;

float suma = 0.0;

void fsm_i2c_init(void)
{
	I2CA_Init();
	bmi088_init();
	bmi088_reg_conf();

	gyro_x=0;
	gyro_y=0;
	gyro_z=0;

	accel_x=0;
	accel_y=0;
	accel_z=0;

	girox = 0;
	giroy = 0;
	giroz = 0;
	v_angular = 0;

	timer_i2c=1;
	estado_i2c=nothing;
	delay_loop(200000);
}

void fsm_i2c_cycle (void)
{
	estado_i2c();
}

static void nothing (void)
{
	if (timer_i2c==1)
	{
		estado_i2c=reading;
		timer_i2c = 0;
	}
}
static void reading (void)
{
	bmi088_Accel_Reading(&accel_x, &accel_y, &accel_z);
	bmi088_Gyro_Reading(&gyro_x,&gyro_y,&gyro_z);
	iteraciones++;
	bmi088_Filtering(accel_x, accel_y, accel_z, gyro_x, gyro_y, gyro_z);
	timer_i2c=1;
	suma = suma + giroz;
```

```
	estado_i2c = nothing;

	if (iteraciones==20)
	{
		estado_i2c=promediado;
	}
	else if (iteraciones>=20)
	{
		iteraciones=0;
	}
	else
	{
	estado_i2c= nothing;
	}
}

static void promediado (void)
{
	v_angular = suma / 20;
	v_angular = v_angular - 0.09;		//calibración
	suma = 0;

	timer_i2c = 1;
	estado_i2c=nothing;
}
```

```
//CONFIGURACION I2C//

#include "i2c.h"

Uint16 direccion;

void I2CA_Init(void){

      //configuracion del reloj y del adress
      //el reloj se dwja igual que en los ejemplos

      I2caRegs.I2CMDR.all = 0x0000;    // Reset the I2C module
      // I2C Prescale Register
      I2caRegs.I2CPSC.all = 14;       // Internal I2C module clock = SYSCLK/(PSC +1)
                                                  // = 10 MHz

      I2caRegs.I2CCLKL = 95;                // Tmaster = (PSC +1)[ICCL + 5 + ICCH +
5] / 150MHz
      I2caRegs.I2CCLKH = 95;                // Tmaster =  15 [ICCL + ICCH + 10] /
150 MHz
                                                  // d = 5  for IPSC >1

                                                  // for I2C 50 kHz:
                                                  // Tmaster = 20 µs -- 20
µs * 150 MHz / 15 = 200 = (ICCL + ICCH +10)
                                                  // ICCL + ICCH = 190
                                                  // ICCL = ICH = 190/2 = 95

      //I2caRegs.I2CIER.all = 0x24;           // Enable SCD & ARDY interrupts  0010
0100
      I2caRegs.I2CIER.bit.AAS = 0;          // Addressed as slave interrupt
enable bit
      I2caRegs.I2CIER.bit.SCD = 1;          // Stop condition detected interrupt
enable bit
      I2caRegs.I2CIER.bit.XRDY = 1;         // Transmit-data-ready interrupt
enable bit
      I2caRegs.I2CIER.bit.XRDY = 1;         // Receive-data-ready interrupt
enable bit
      I2caRegs.I2CIER.bit.ARDY = 1;         // Register-access-ready interrupt
enable bit
      I2caRegs.I2CIER.bit.NACK = 0;         // No-acknowledgment interrupt enable
bit
      I2caRegs.I2CIER.bit.ARBL = 0;         // Arbitration-lost interrupt enable
bit

      //I2caRegs.I2CMDR.all = 0x0020;         // Take I2C out of reset,Stop I2C
when suspended
      I2caRegs.I2CMDR.bit.NACKMOD = 0;      // NACK mode bit
      I2caRegs.I2CMDR.bit.FREE = 0;         // Stop I2C when suspended
      I2caRegs.I2CMDR.bit.STT = 0;          // START condition bit
      I2caRegs.I2CMDR.bit.STP = 0;          // STOP condition bit
      I2caRegs.I2CMDR.bit.MST = 0;          // Slave mode
      I2caRegs.I2CMDR.bit.TRX = 0;          // Receiver mode
      I2caRegs.I2CMDR.bit.XA = 0;           // 7-bit addressing mode
      I2caRegs.I2CMDR.bit.RM = 0;           // Nonrepeat mode
      I2caRegs.I2CMDR.bit.DLB = 0;          // Digital loopback mode is disabled
      I2caRegs.I2CMDR.bit.IRS = 1;          // The I2C module is enabled
      I2caRegs.I2CMDR.bit.STB = 0;          // The I2C module is not in the START
byte mode
```

```
      I2caRegs.I2CMDR.bit.FDF = 0;              // Free data format mode is disabled
      I2caRegs.I2CMDR.bit.BC = 0;               // 8 bits per data byte

}

Uint16 I2C_read_one_byte(Uint16 address, Uint16 registro)
{
    I2caRegs.I2CMDR.bit.IRS = 1;                // reset I2C

    // Make sure I2C is not busy and has stopped
    while (I2caRegs.I2CSTR.bit.BB == 1);        // busy loop
    I2caRegs.I2CSTR.bit.SCD = 1;                // Clear the SCD bit (stop condition
bit)
    while(I2caRegs.I2CMDR.bit.STP == 1);        // stop bit loop

    I2caRegs.I2CSAR = address;              // I2C slave address

    while (I2caRegs.I2CSTR.bit.BB == 1);        // still busy?

    //I2caRegs.I2CMDR.all = 0x2620;             // start, no stop bit, master, tx,
reset I2C 00100110
    I2caRegs.I2CMDR.bit.NACKMOD = 0;            // NACK mode bit
    I2caRegs.I2CMDR.bit.FREE = 0;               // Stop I2C when suspended
    I2caRegs.I2CMDR.bit.STT = 1;                // START condition bit
    I2caRegs.I2CMDR.bit.STP = 0;                // STOP condition bit
    I2caRegs.I2CMDR.bit.MST = 1;                // Master mode
    I2caRegs.I2CMDR.bit.TRX = 1;                // Transmitter mode
    I2caRegs.I2CMDR.bit.XA = 0;                 // 7-bit addressing mode
    I2caRegs.I2CMDR.bit.RM = 0;                 // Nonrepeat mode
    I2caRegs.I2CMDR.bit.DLB = 0;                // Digital loopback mode is disabled
    I2caRegs.I2CMDR.bit.IRS = 1;                // The I2C module is enabled
    I2caRegs.I2CMDR.bit.STB = 0;                // The I2C module is not in the START
byte mode
    I2caRegs.I2CMDR.bit.FDF = 0;                // Free data format mode is disabled
    I2caRegs.I2CMDR.bit.BC = 0;                 // 8 bits per data byte

    I2caRegs.I2CCNT     = 1;                    // assume register address is one
byte

    while(I2caRegs.I2CSTR.bit.XRDY == 0);       // Do nothing till bus is free
    I2caRegs.I2CDXR     = registro;                 // register address of the
sensor (1 byte)

    while(!I2caRegs.I2CSTR.bit.ARDY);           // all ready?
    //I2caRegs.I2CMDR.all = 0x2C20;             // start, stop bit when CNT =0,
master, rx, reset I2C 00101100
    I2caRegs.I2CMDR.bit.NACKMOD = 0;            // NACK mode bit
    I2caRegs.I2CMDR.bit.FREE = 0;               // Stop I2C when suspended
    I2caRegs.I2CMDR.bit.STT = 1;                // START condition bit
    I2caRegs.I2CMDR.bit.STP = 1;                // STOP condition bit
    I2caRegs.I2CMDR.bit.MST = 1;                // Master mode
    I2caRegs.I2CMDR.bit.TRX = 0;                // Receiver mode
    I2caRegs.I2CMDR.bit.XA = 0;                 // 7-bit addressing mode
    I2caRegs.I2CMDR.bit.RM = 0;                 // Nonrepeat mode
    I2caRegs.I2CMDR.bit.DLB = 0;                // Digital loopback mode is disabled
    I2caRegs.I2CMDR.bit.IRS = 1;                // The I2C module is enabled
    I2caRegs.I2CMDR.bit.STB = 0;                // The I2C module is not in the START
byte mode
    I2caRegs.I2CMDR.bit.FDF = 0;                // Free data format mode is disabled
    I2caRegs.I2CMDR.bit.BC = 0;                 // 8 bits per data byte
```

```
    I2caRegs.I2CCNT     = 1;                    // only read one byte data

    if(I2caRegs.I2CSTR.bit.NACK == 1)
    {
         I2caRegs.I2CSTR.all = 0x0002;
    }
    I2caRegs.I2CMDR.bit.STP = 1;                    // stop bit when CNT=0

    while(!I2caRegs.I2CSTR.bit.SCD);                // stop bit detected?

    return(I2caRegs.I2CDRR);
}

void I2C_write_one_byte(Uint16 address, Uint16 registro, Uint16 data)
{
    I2caRegs.I2CMDR.bit.IRS = 1;                // reset I2C

    // Make sure I2C is not busy and has stopped
    while (I2caRegs.I2CSTR.bit.BB == 1);        // busy loop
    I2caRegs.I2CSTR.bit.SCD = 1;                // Clear the SCD bit (stop condition
bit)
    while(I2caRegs.I2CMDR.bit.STP == 1);        // stop bit loop

    I2caRegs.I2CSAR = address;                  // I2C slave address
    while (I2caRegs.I2CSTR.bit.BB == 1);        // still busy?

    I2caRegs.I2CCNT = 2;                        // assume register address = 1 byte,
and data is 1 byte

    //I2caRegs.I2CMDR.all = 0x6E20;                 // start, stop, no rm, reset i2c
01101110 00100000
    I2caRegs.I2CMDR.bit.NACKMOD = 0;            // NACK mode bit
    I2caRegs.I2CMDR.bit.FREE = 1;               // Run free I2C when suspended
    I2caRegs.I2CMDR.bit.STT = 1;                // START condition bit
    I2caRegs.I2CMDR.bit.STP = 1;                // STOP condition bit
    I2caRegs.I2CMDR.bit.MST = 1;                // Master mode
    I2caRegs.I2CMDR.bit.TRX = 1;                // Transmitter mode
    I2caRegs.I2CMDR.bit.XA = 0;                 // 7-bit addressing mode
    I2caRegs.I2CMDR.bit.RM = 0;                 // Nonrepeat mode
    I2caRegs.I2CMDR.bit.DLB = 0;                // Digital loopback mode is disabled
    I2caRegs.I2CMDR.bit.IRS = 1;                // The I2C module is enabled
    I2caRegs.I2CMDR.bit.STB = 0;                // The I2C module is not in the START
byte mode
    I2caRegs.I2CMDR.bit.FDF = 0;                // Free data format mode is disabled
    I2caRegs.I2CMDR.bit.BC = 0;                 // 8 bits per data byte

    while(I2caRegs.I2CSTR.bit.XRDY == 0);       // Do nothing till bus is free
    I2caRegs.I2CDXR = registro;                     // register address of the
sensor (1 byte)

    while(I2caRegs.I2CSTR.bit.XRDY == 0);       // Do nothing till bus is free
    I2caRegs.I2CDXR = data;                     // data to be sent (1 byte)

    I2caRegs.I2CMDR.bit.STP = 1;                // stop bit when CNT=0

    while(!I2caRegs.I2CSTR.bit.SCD);        // wait for STOP condition

}
```

```
//HEADER BMI088.h//

#ifndef __BMI088_H__
#define __BMI088_H__

#include "DSP2833x_Device.h"
#include "i2c.h"
#include "math.h"
#include "compartiresvivir.h"

#define ACC_PWR_CONF 0x7C
#define ACC_PWR_CTRL 0x7D
#define ACC_CONF 0x40
#define ACC_CHIP_ID 0x00
#define ACC_X_MSB 0x13
#define ACC_X_LSB 0x12
#define ACC_Y_MSB 0x15
#define ACC_Y_LSB 0x14
#define ACC_Z_MSB 0x17
#define ACC_Z_LSB 0x16
#define ACC_RANGE 0x41

#define GYRO_LMP1 0x11
#define GYRO_CHIP_ID 0x00
#define RATE_X_MSB 0x03
#define RATE_X_LSB 0x02
#define RATE_Y_MSB 0x05
#define RATE_Y_LSB 0x04
#define RATE_Z_MSB 0x07
#define RATE_Z_LSB 0x06
#define GYRO_RANGE 0x0F
#define GYRO_BANDWIDTH 0x10

void bmi088_reg_conf(void);
Uint16 bmi088_test_connection(void);
void bmi088_Accel_Reading(int *accel_x, int *accel_y, int *accel_z);
void bmi088_Gyro_Reading(int *gyro_x, int *gyro_y, int *gyro_z);
void bmi088_Filtering(int accel_x, int accel_y, int accel_z, int gyro_x,  int gyro_y,
int gyro_z);
void bmi088_init(void);

#endif
```

```
//HEADER FMS_I2C.h//

#ifndef __FSM_I2C_H__
#define __FSM_I2C_H__

#include "DSP2833x_Device.h"
#include "compartiresvivir.h"
#include "bmi088.h"
#include "i2c.h"


void fsm_i2c_init (void);
void fsm_i2c_cycle (void);

#endif
```

```
//HEADER I2C.h//

#ifndef __I2C_H__
#define __I2C_H__

#include "DSP2833x_Device.h"
#include "compartiresvivir.h"

void I2CA_Init(void);
Uint16 I2C_read_one_byte(Uint16 address, Uint16 registro);
void I2C_write_one_byte(Uint16 address, Uint16 registro, Uint16 data);
void delay_loop(long us);

#endif
```