Universidad Zaragoza
1542

Trabajo Fin de Grado

Título del trabajo: Controlador de bajo nivel de un robot móvil todo terreno con ruedas

English tittle: Low level controller for an all-terrain vehicle with wheels

Autor/es

Bruno Gallán Farina

Director/es

José Luis Villarroel Salcedo

ESCUELA DE INGENIERÍA Y ARQUITECTURA

Año: 2017

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. Bruno Gallán Farina,

con nº de DNI 25203585R en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado, (Título del Trabajo) Controlador de bajo nivel de un robot móvil todo terreno con ruedas,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 21/11/2017

Fdo: Bruno Gallán Farina

RESUMEN

En este apartado se ofrece un resumen del trabajo que se ha realizado, desde el planteamiento del proyecto hasta la conexión del sistema de control al vehículo todoterreno real.

El objetivo del proyecto consiste en el desarrollo de los bucles de control de velocidad de las ruedas y de posición de la dirección de la plataforma móvil con ruedas todo terreno *RobuCar-TT*. La motivación que impulsó el proyecto es la búsqueda de un sistema de control cuyo mantenimiento y adaptación sean más sencillos que los que traía el *Robucar* por defecto.

El robot se apoya sobre cuatro ruedas distribuidas en dos ejes. Ambos utilizan la geometría de dirección Ackermann, permitiendo varios modos de movimiento. En este proyecto se controlará el tren delantero del robot en modo coche, que consiste en mantener rectas las ruedas traseras y girar las delanteras.

El sistema recibirá las consignas de velocidad lineal y angular desde el exterior, calculando la velocidad de cada rueda por medio del modelo cinemático del robot. Teniendo en cuenta que los motores del *Robucar* son de corriente continua, para controlar la velocidad de los motores se utilizará un regulador PI, y para el ángulo de la dirección un proporcional.

Forma parte de los requisitos del proyecto que esté se realice sobre el *LaunchPad F28377S de Texas Instruments.* Debido a ciertas limitaciones del mismo, habrá que utilizar dos. Será necesario preparar el hardware adecuado para la conexión entre el *LaunchPad* y el *Robucar.* Así mismo, hay que identificar y comprender el sistema de cableado del robot y los drivers de potencia que lleva incorporados para alimentar a los motores.

Respecto del software, primero se realiza una recopilación de las herramientas necesarias en sus versiones adecuadas. El proyecto se desarrollará sobre la versión 7.1.0 de *CodeComposerStudio* y se utilizará la versión 2.16.1.14 de *TI-RTOS*, un conjunto de herramientas entre las que se incluye *SYS/BIOS*, un sistema operativo de tiempo real.

Una vez recopiladas las herramientas, se plantea la estructura de tareas y se divide entre los dos *LaunchPad*. Tras lo cual se procede a la implementación de los bucles de control. Para ello será necesario el desarrollo de dos funciones de apoyo, una para calcular la velocidad y posición de las ruedas y otra que haga el cálculo de los reguladores. Estas funciones son explicadas con mayor detalle en sus respectivas secciones.

Para facilitar el testeo del programa, se realizarán pruebas sobre un pequeño motor de *Lego Mindstorm,* el cual necesitará también cierto hardware para su conexión. Una vez testeado el programa, se acude al robot con el hardware y el software listos.

Finalmente, se ofrece una sección con un resumen de lo que se ha conseguido realizar y las conclusiones a las que se ha llegado.

Contenido

Índice de figuras

1 INTRODUCCIÓN

1.1 Marco

El proyecto se lleva a cabo dentro del Grupo de Robótica, Percepción y Tiempo Real de la Universidad de Zaragoza, el cual está en posesión de un robot del tipo *RobuCar-TT* fabricado por la empresa francesa *Robosoft.*

Se pretende desarrollar un sistema de control de velocidad y dirección para dicho robot. El sistema recibirá ordenes de velocidad lineal y angular que trasladará a los bucles de control por medio del modelo cinemático.

1.2 Motivación y objetivos

La plataforma móvil con ruedas todo terreno RobuCar-TT incorpora por defecto un sistema de control cerrado el cual presenta cierta dificultad en su mantenimiento y adaptación. Es por ello que surge la motivación de desarrollar un nuevo sistema de control más accesible dentro del grupo.

1.3 Requisitos

El nuevo sistema debe ser capaz de recibir consignas de velocidad desde el exterior, interactuar adecuadamente con los *drivers* de potencia y sensores que manejan los motores. Además, debe ser respetuoso con el anterior sistema, permitiendo la fácil conexión y desconexión de ambos.

El control se realizará únicamente para el tren delantero del *Robucar* en modo coche (ver apartado “2. Desciepción del robot”). Se usará *LaunchPad F28377S de Texas Instruments* y el sistema operativo de tiempo real *SYS/BIOS*.

1.4 Estructura de la memoria

A continuación se ofrece un breve resumen de las distintas secciones principales que componen la memoria del proyecto.

Descripción del robot

Se ofrece una descripción del robot y se expone brevemente el modelo cinemático del mismo basándose en el proyecto fin de carrera de Carlos Gracia Sola [1]. Para mayor profundidad sobre el modelo consultar dicho trabajo.

Diseño arquitectural

En este apartado se establece el diseño arquitectural del sistema. Se muestran unos sencillos esquemas que ofrecen una visión global de lo que se pretende construir en este proyecto.

Estructura de los bucles de control

Aquí se exponen los esquemas de control que se utilizaran para controlar la velocidad de las ruedas, el ángulo de dirección y se justifica porqué.

Desarrollo

Este apartado abarca el grueso de la memoria, se detalla por separado el desarrollo del *hardware* y el *software* necesarios para el proyecto.

Respecto al *hardware,* se tratan los siguientes puntos:

- Se introduce al *LaunchPad* F28377S de Texas Instruments, plataforma sobre la que se va a desarrollar el proyecto.
- Se desarrollan los circuitos necesarios para conectar el *LaunchPad* al todoterreno.
- Se explica cómo conectar el *LaunchPad* al todoterreno.

Respecto al software:

- Se recopila todo lo necesario para poder empezar a programar.
- Se explica el funcionamiento del sistema operativo de tiempo real.
- Se explica el sistema de tareas.
- Implementación de los bucles de control.

Finalmente, se presentan los resultados de algunas pruebas realizadas.

2 DESCRIPCIÓN DEL ROBOT

Como se indica en el proyecto de Carlos Gracia Sola [1], el robot sobre el que se basa el proyecto es un *Robucar*, propiedad del Grupo de Robótica, Percepción y Tiempo Real de la Universidad de Zaragoza y fabricado por la empresa francesa *Robosoft*. En concreto, es un *Robucar TT* modificado. Se le han añadido varios elementos como más sensores láser, un brazo robótico y una cámara térmica. La Figura 1 muestra un *Robucar* básico y la Figura 2 del modificado para el grupo.

Figura 1. Robucar básico [1]

Figura 2. Robucar modificado (el nuestro) [1]

2.1 Modelo cinemático

El robot se apoya sobre cuatro ruedas distribuidas en dos ejes. Cada rueda es accionada por un motor de corriente continua. Para permitir el giro del robot se utiliza la geometría de dirección de *Ackermann*. En la Figura 3 se puede ver un esquema de esa geometría para el eje delantero de un vehículo. En el *Robucar* ambos ejes utilizan esta disposición, lo que permite varios modos de movimiento.

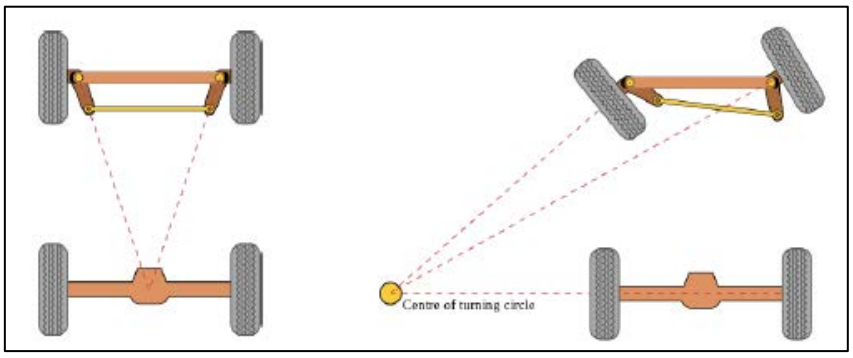

Figura 3. Geometría de dirección Ackermann

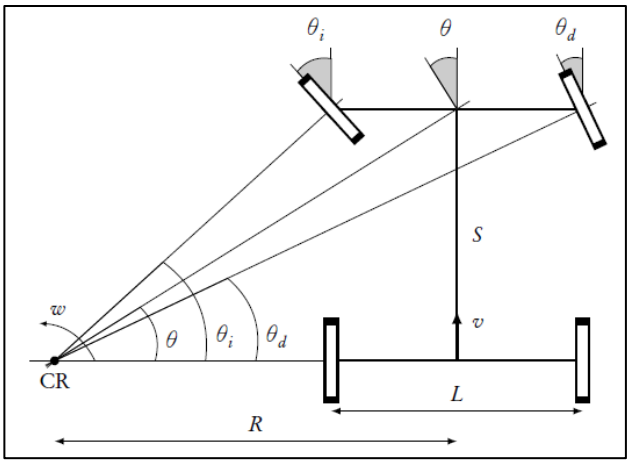

Figura 4. Esquema del robot en modo coche

Medida	Resultado (m)
Longitud del robot	2,90
Anchura del robot	1,30
Altura hasta la tabla	0,93
Altura hasta el punto más alto	2,12
Separación entre ejes	1,20
Longitud de los ejes	1,20
Anchura de las ruedas	0,10
Radio teórico de las ruedas	0,30

Figura 5. Medidas de la geometría del robot

Gracias a su doble geometría *Ackermann*, el *Robucar* puede funcionar en los siguientes modos: coche, dual y cangrejo o *crab*, a continuación se expresan únicamente los resultados para las velocidades de cada rueda y ángulo de giro de las ruedas para el modo coche, el utilizado para este trabajo, para más detalles consultar el proyecto fin de carrera de Carlos Gracia Sola [1].

En el modo coche el robot mantiene rectas las ruedas traseras y gira las delanteras. Como se puede ver en la Figura 4, el ángulo que tienen que girar esas ruedas no es el mismo, algo que permite la geometría de Ackermann.

$$w = \frac{v}{R}.$$

$$\theta_i = \tan^{-1}\left(\frac{S}{R-\frac{L}{2}}\right)$$

$$\theta_d = \tan^{-1}\left(\frac{S}{R+\frac{L}{2}}\right)$$

$$v_{f,i} = \sqrt{\left(v - w\cdot\frac{L}{2}\right)^2 + (w\cdot S)^2}$$

$$v_{t,i} = v - w\cdot\frac{L}{2}$$

$$v_{f,d} = \sqrt{\left(v + w\cdot\frac{L}{2}\right)^2 + (w\cdot S)^2}$$

$$v_{t,d} = v + w\cdot\frac{L}{2}$$

Las ecuaciones anteriores constituyen el modelo cinemático. Para obtener la velocidad angular de cada rueda basta con dividir su velocidad lineal entre el radio de las ruedas (las dimensiones del robot se muestran en la Figura 5). En la sección “5.1.3. conexión al todoterreno” se indica cómo hacer las conexiones al robot.

3 DISEÑO ARQUITECTURAL

3.1 Nuestro sistema

Para ofrecer una idea más clara de lo que se quiere construir, en la Figura 6 se muestra un esquema del sistema de control.

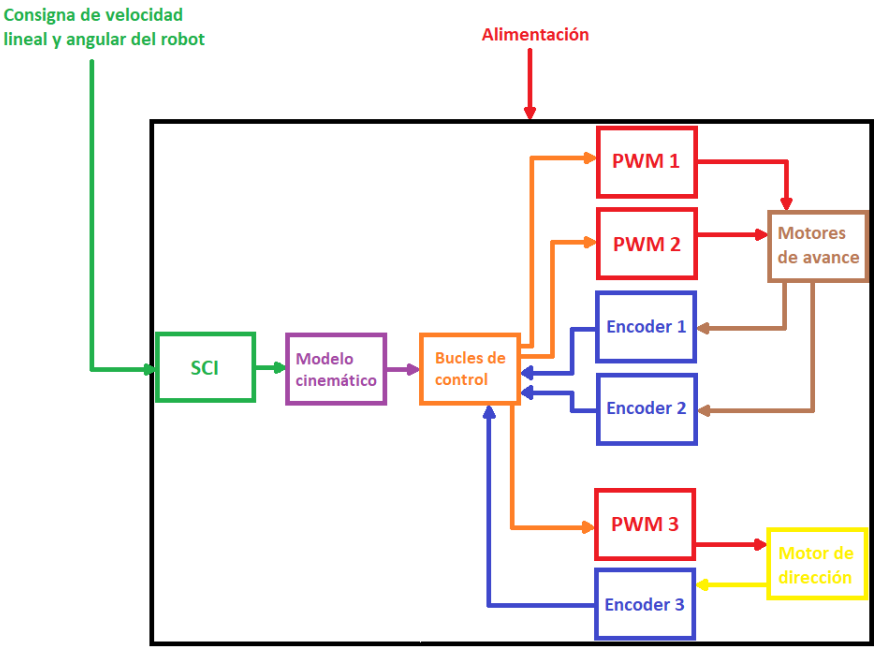

Figura 6. Esquema del sistema

El sistema recibirá las consignas de velocidad lineal y angular del *Robucar* desde el exterior. Tras lo cual, por medio del modelo cinemático expuesto en el apartado anterior, se calcula la velocidad angular correspondiente para cada rueda y el ángulo de giro para el motor de dirección, que serán las consignas de los tres bucles de control. En cada bucle de control, el regulador calcula la acción que es generada mediante PWMs. La velocidad y posición de los motores se realimenta con los *encoders* que llevan incorporados.

3.2 El robucar

Se va a controlar la cabecera del robot en modo coche, lo que implica regular la velocidad angular de las ruedas delanteras y su ángulo de giro, dejando las dos ruedas traseras desconectasas y sus frenos desactivados.

En la Figura 7 y en la Figura 8 se muestra un sencillo esquema del *Robucar*.La señal PWM generada por el sistema de control se lleva a los módulos de potencia que traía por defecto el robot, estos finalmente son los que se encargan de alimentar a los motores. Para más detalles acerca del cableado y los drivers de potencia consultar la sección “5.1.3. conexión al todoterreno”.

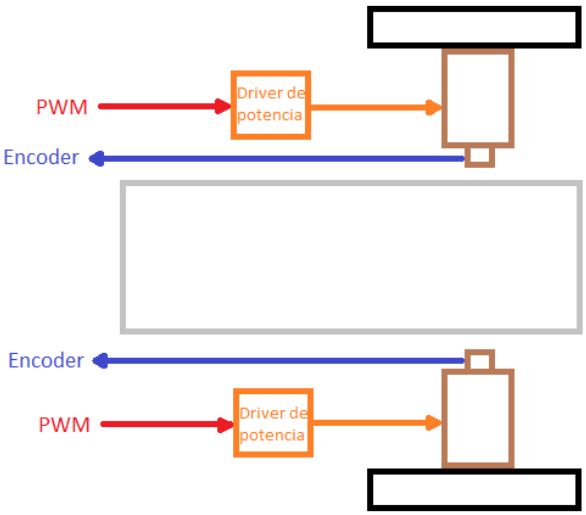

Figura 7. Conexión al *robucar*, motores de velocidad

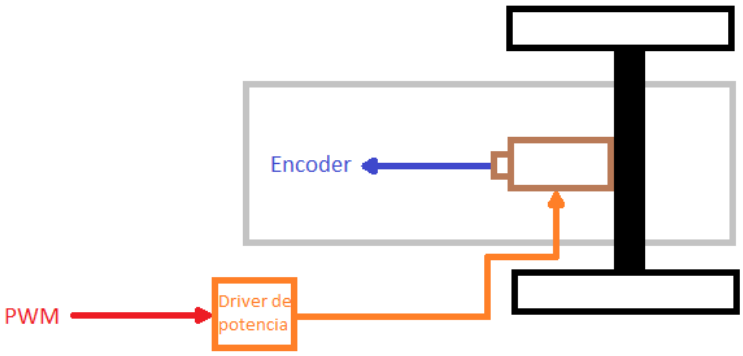

Figura 8. Conexión al *robucar*, motor de dirección

4 ETRUCTURA DE LOS BUCLES DE CONTROL

4.1 Bucle para el control de velocidad

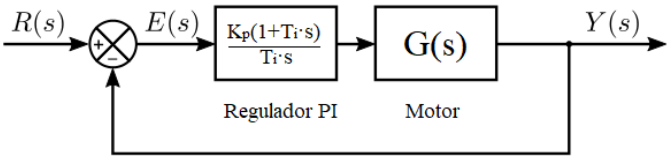

Figura 9. Esquema regulador PI

La estructura de los bucles de control a implementar es la de un regulador PI clásico como el que se muestra en la Figura 9. La función de transferencia que relaciona la tensión de entrada y la velocidad angular de un motor de corriente continua es un primer orden, por lo que con este regulador se pretende conseguir error de posición nulo y un buen tiempo de respuesta. Se asumirá que el sistema toma muestras lo suficientemente rápido. Para más detalles consultar los apartados "5.2.3.6 implementación de los reguladores".

Como se detalla en el apartado "5.1.4 Identificación de la dinámica del robot", la función de transferencia que relaciona la tensión de entrada en voltios a los módulos de potencia de los motores con la velocidad de las ruedas en radianes por segundo es:

$$G(s) = \frac{1.78}{1 + 0.4s}$$

$$R(s) \cdot G(s) = k \cdot \frac{1 + \tau_i s}{\tau_i s} \cdot \frac{1.78}{1 + 0.4s}$$

Para cancelar el efecto del polo del sistema se le asigna un valor a $\tau_i = 0.4$. Para obtener la ganancia k, se calcula el bucle cerrado y se ajusta para obtener un tiempo de respuesta de 0.5 segundos. Lo que implica que la constante de tiempo del sistema en bucle cerrado sea $3\tau = 0.5 \Rightarrow \tau = 0.166$.

$$F_{BC} = \frac{1.78k}{\tau_i s + 1.78k} = \frac{1}{1 + \frac{\tau_i s}{1.78k}} \Rightarrow \frac{\tau_i}{1.78k} = \frac{0.4}{1.78k} = \frac{0.224}{k} = 1.166 \Rightarrow k = 0.1921$$

Entonces, finalmente se obtiene:

$$R(s) = 0.1921 \frac{1 + 0.4s}{0.4s}$$

4.2 Bucle para el control de posición

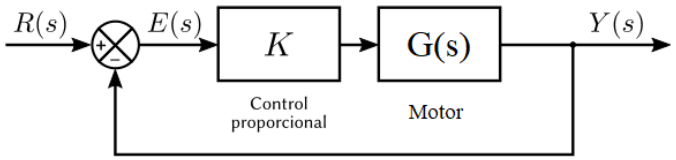

Figura 10. Esquema regulador proporcional

Para el control de posición de la dirección se usará un regulador proporcional como el mostrado en la Figura 10. La función de transferencia que relaciona la tensión de entrada y la posición angular de un motor de corriente continua ya incluye un integrador, por lo que no hace falta añadirlo en el regulador, para más detalles consultar los apartados "5.1.4 Identificación de la dinámica del robot" y "5.2.3.6 implementación de los reguladores".

En este caso tenemos:

$$G(s) = \frac{0.1}{s(1 + 0.25s)}$$

$$R(s) \cdot G(s) = k \cdot \frac{0.1}{s(1 + 0.25s)}$$

$$F_{BC} = \frac{0.1k}{s(1 + 0.25s) + 0.1k} = \frac{0.1k}{0.25s^2 + s + 0.1k} = \frac{0.4k}{s^2 + \frac{s}{0.25} + 0.4k}$$

Para un tiempo de respuesta de 0.5 segundos y un comportamiento críticamente amortiguado se obtiene:

$$0.5 = \frac{4.75}{w_n} \Rightarrow w_n = 9.5 \Rightarrow 0.4k = 9.5^2 \Rightarrow k = 225.625$$

Entonces, finalmente resulta:

$$R(s) = 225.625$$

5 DESARROLLO

5.1 Hardware

5.1.1 LaunchPad F28377S

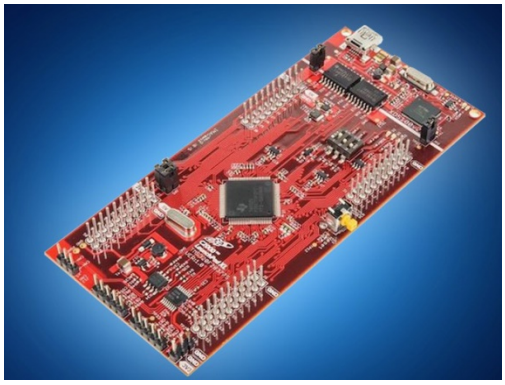

Figura 11. LaunchPad F28377S

El sistema de control que se va a desarrollar en este proyecto está pensado para ejecutarse sobre el *LaunchPad F28377S de Texas Instruments* [2], mostrado en la Figura 11. Es una placa de desarrollo de bajo coste para los dispositivos *Texas Instruments Delfino F2837xS*. Incluye todas las herramientas de hardware necesarias para desarrollar aplicaciones basadas en los microcontroladores de la familia F2837xS, esta placa concretamente, utiliza el micro *TMS320F28377S* [3].

Ofrece interesantes características como la posibilidad de migrar los proyectos a dispositivos de todavía menor coste. Tiene un conector JTAG que permite conectar fácilmente el *LaunchPad* al PC por medio de un cable USB, facilitando en gran medida la programación, depuración y evaluación de los programas.

5.1.1.1 Características VS necesidades

Como se ha indicado en el apartado “3 Diseño arquietctural”, son necesarias 3 señales PWM y 3 módulos de lectura de *encoders*. El *LaunchPad* F28377S tiene suficientes señales PWM pero únicamente posee dos módulos de lectura de *encoders*, lo cual obliga al uso de dos placas, una se encargará de recibir las consignas de velocidad, calcular el modelo cinemático y controlar la velocidad de las ruedas, mientras que la segunda de controlar la posición de la dirección. Ambas placas se comunicarán mediante una conexión SCI. La Figura 12 muestra el esquema de la conexión a los *LaunchPad*.

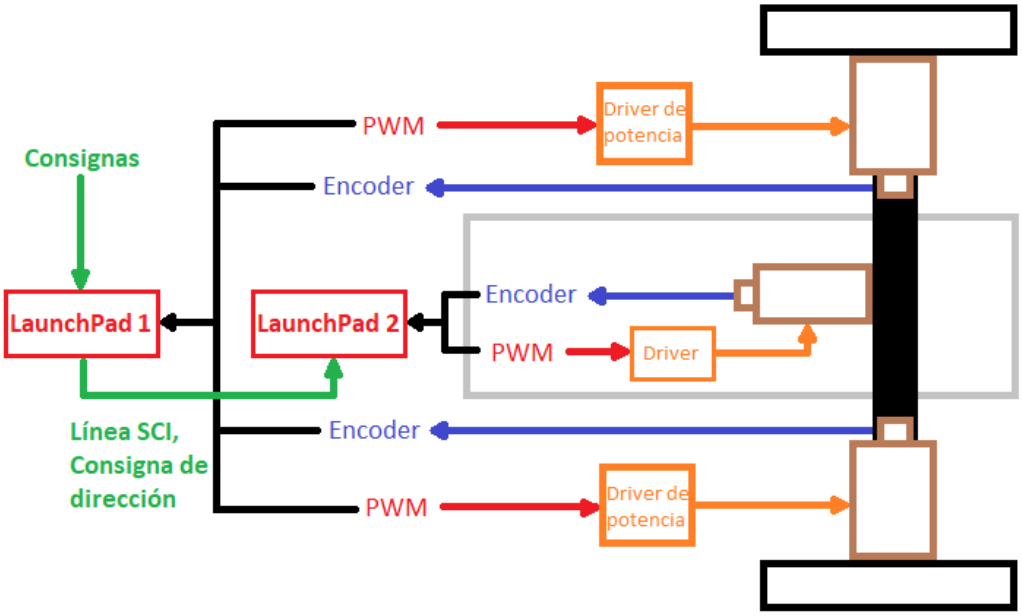

Figura 12. Esquema completo

5.1.2 Circuito de alimentación y adaptación de tensiones

El circuito que se va a describir en esta sección será el mismo que se utilice para alimentar al *LaunchPad* durante los ensayos con el todoterreno. A continuación se describen brevemente los materiales necesarios para alimentar a la placa y montar el circuito de alimentación. Al final de la sección, se mostrará el circuito final y como alimentar el *LaunchPad* externamente.

No es posible alimentar a todos los componentes necesarios únicamente a través del ordenador. La conexión USB 2.0 puede proporcionar una corriente máxima de 500 mA, teniendo en cuenta qué hay que alimentar a dos *LaunchPad,* los cuales necesitan 250 mA cada uno aproximadamente, no deja margen para los tres *encoders* y el circuito de adaptación de tensiones.

Además, esta opción resultaría delicada a la hora de realizar los ensayos en el vehículo todoterreno real, porque implica conectar la masa del ordenador portátil utilizado en las pruebas con el terminal negativo de la batería del *Robucar*. Entonces, teniendo en cuenta estos inconvenientes, se precisa buscar un medio de alimentación externo.

5.1.2.1 Cargador

Figura 13. Cargador de 5V

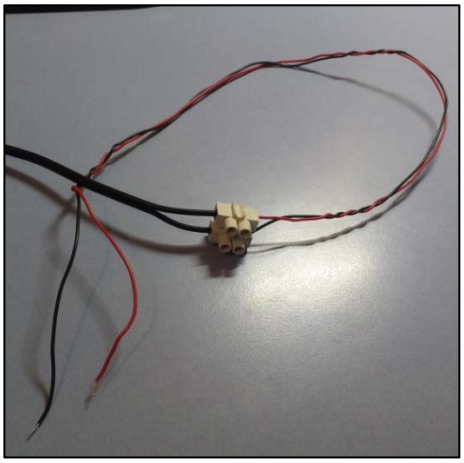

Figura 14. Ficha eléctrica del cargador

Para alimentar al circuito de adaptación y al *LaunchPad* se barajaron varias posibilidades, finalmente se optó por utilizar un cargador de 5V (Figura 13) capaz de suministrar hasta 3 amperios como máximo, más que suficiente para las placas y el resto de componentes.

Para facilitar la conexión del cargador, se modificó su terminación como se muestra en la Figura 14.

5.1.2.2 Fuente conmutada DC/DC

Durante las etapas finales del proyecto surgió la necesidad de encontrar un sistema de alimentación sin necesidad de conexión a la red eléctrica. Para ello se localizó en varios puntos del todo terreno una tensión de 12,5V, entonces, mediante el conversor DC/DC mostrado en la Figura 15, se reduce a 5V con los que se alimentará el sistema de la misma manera que con el cargador.

Dicho conversor tendrá que alimentar hasta dos *LaunchPads* y tres *encoders*. Según los manuales, los *encoders* tienen un consumo típico de 60 mA. Según los foros de *Texas Instruments* los *LaunchPads* de 250 mA, sumando 680mA en total. Por consiguiente, se escoge el conversor de montaje en panel SD-15A-5 de *Mean Well* (3A, 15W), el más barato disponible en *Farnell* para este rango de tensiones y corrientes.

Figura 15. Conversor DC/DC

5.1.2.3 Fijador de tensión

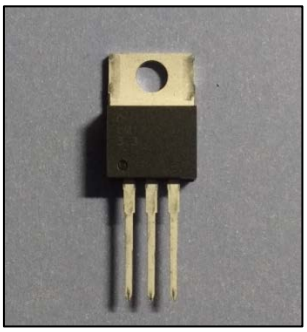

Figura 16. Fijador de tensión a 3,3V

El cargador y el conversor DC/DC proporcionan 5 voltios de tensión, mientras que el *LaunchPad* necesita 3,3V para funcionar, por consiguiente es necesario reducir la tensión de alimentación. Para ello se ha optado por utilizar un fijador de tensión. Se barajó la posibilidad de limitar la tensión de entrada mediante diodos o montando un circuito con transistores pero, finalmente, por sencillez y eficacia se optó por el fijador.

El fijador de tensión elegido es el incluido en el integrado LM1117 de *Texas Instruments* (Figura 16), su encapsulado permite una conexión directa a la placa blanca de montaje rápido sin necesidad de ningún tipo de adaptación adicional.

5.1.2.4 Adaptación de nivel de tensión

Hay un problema con la tensión que proporciona el *LaunchPad* con su señal PWM. Estos llegan hasta 3,3V, mientras que los drivers de potencia utilizan señales de 5V. Para solucionar este problema se barajaron varias posibilidades como la utilización de un transistor como interruptor pero, finalmente, por eficacia y sencillez se optó por la utilización del integrado GTL2003 de *NXP Semiconductors*. Desafortunadamente, en el momento de realizar la compra solo existía disponibilidad con encapsulado de tipo TSSOP20 por lo que para posibilitar la conexión del mismo sobre la placa de montaje rápido fue necesario soldarlo a un adaptador a 20DIP.

En la Figura 17 se muestra el traductor de tensión con el adaptador a 20DIP ya soldado y en la Figura 18 el esquema de conexión proporcionado por el manual del integrado para una traducción de baja a alta tensión.

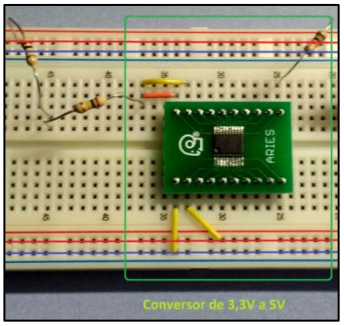

Figura 17. Conversor de 3,3 a 5V

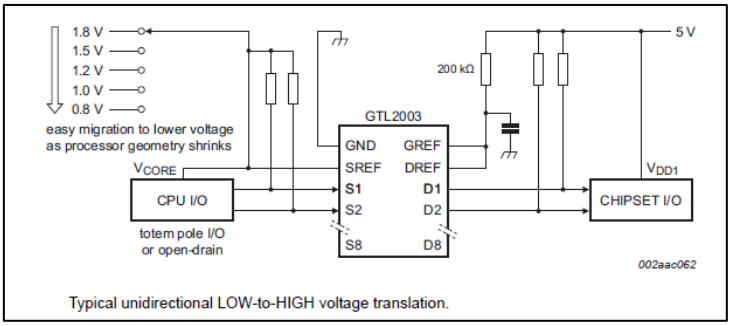

Figura 18. Esquema de conexión del conversor

La señal del PWM del *LaunchPad* de 3,3V se lleva a una de las entradas del integrado, en su salida correspondiente se obtiene la misma señal traducida a 5V; es necesario conectar una resistencia de pull-up a la salida.

5.1.2.5 Circuito final

En la Figura 19 se muestra el circuito de alimentación completo. El cable positivo del cargador con los 5V se conecta a la primera línea horizontal de la placa, mientras que el cable negativo se conecta a la segunda. El fijador de tensión se conecta como se ve en la imagen, llevando su salida con los 3,3V a la penúltima línea. Mediante un cable largo se une la segunda línea con la última para que ambas estén conectadas a masa.

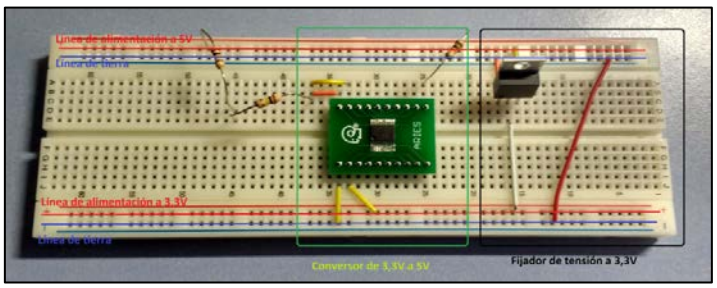

Figura 19. Circuito de alimentación detallado

En la Figura 20 se muestra el esquemático del circuito de alimentación y traducción del PWM. Cabe destacar que es necesario añadir resistencias de *pull-up* a la salida del traductor.

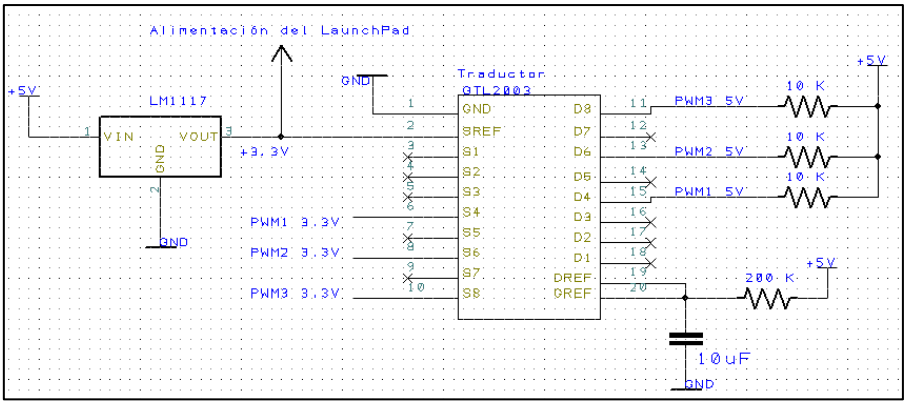

Figura 20. Esquemático del circuito de alimentación y traducción

5.1.2.6 *Alimentación externa del LaunchPad*

Alimentar mediante una fuente externa al *LaunchPad* es relativamente sencillo, basta con retirar los dos *jumpers* mostrados en los recuadros azules de la Figura 21. En la Figura 22 se muestra la placa con los dos *jumpers* desconectados. Esto es necesario para aislar eléctricamente los circuitos de alimentación provenientes del conector USB del resto de la placa. Si no se toma esta precaución, se podrían producir graves daños tanto en la placa como en el ordenador al que esté conectada por USB.

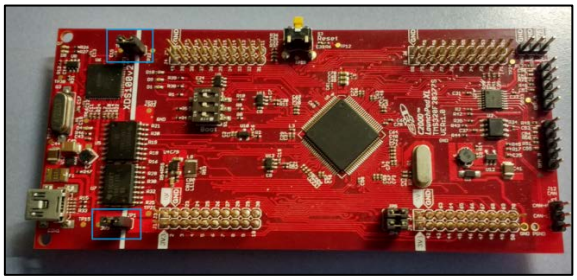

Figura 21. LaunchPad con los jumpers conectados

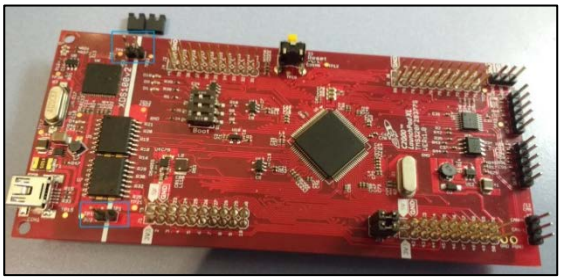

Figura 22. LaunchPad con los jumpers desconectados

Finalmente, en la Figura 23 se muestra el *LaunchPad* con los cables de alimentación ya conectados y los *jumpers* retirados, estando ya todo preparado para realizar la conexión a la alimentación externa.

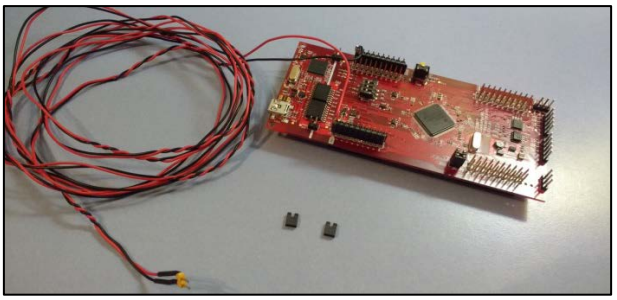

Figura 23. LaunchPad preparado para ser conectado externamente

5.1.3 Conexión al todoterreno

Este apartado del proyecto costó mucho más tiempo del esperado al no disponer de información previa sobre el cableado y programación del vehículo todoterreno. Para poder completar el proyecto, los objetivos principales de esta sección son la lectura de los *encoders* y la actuación sobre los drivers de potencia que controlan las ruedas.

5.1.3.1 Módulo de conexiones wago

El cableado actual se realiza a través de los módulos de conexión *WAGO* (Figura 24), su función es la de acoplar y transmitir por una red *CAN* las señales de los sensores y actuadores. Existen muchos tipos de módulos, para más información se puede consultar la siguiente dirección: *http://www.wago.com/wagoweb/documentation/index_e.htm*. En dicha dirección están disponibles los manuales de cada módulo donde indican sus entradas y salidas.

Los que contiene nuestro robot son:

- 750-430: 8 entradas digitales.
- 750-530: 8 salidas digitales.
- 750-550: 2 salidas analógicas.
- 750-630: Interfaz de transmisión SSI.
- 750-637: Módulo para *encoders* incrementales.
- 750-600: Módulo de finalización.

Concretamente, para este proyecto, nos interesan los dos módulos de lectura de *encoders* y el de salidas analógicas, en los dos siguientes apartados se explican con más detalle cómo están realizadas las conexiones.

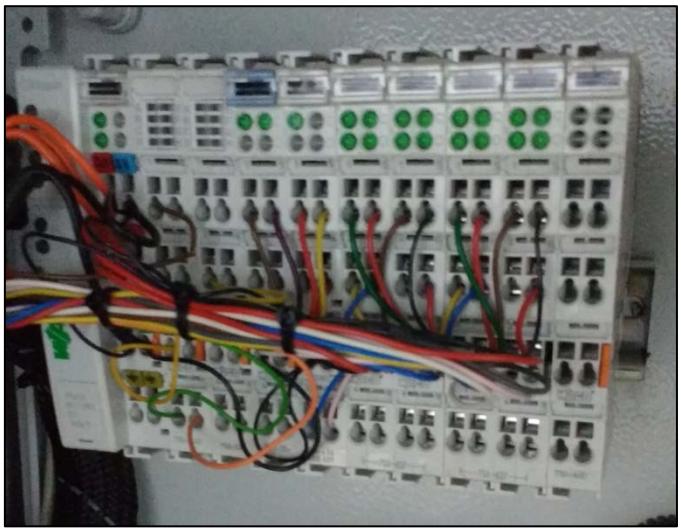

Figura 24. Módulo de conexión *WAGO*

5.1.3.2 Lectura de los encoders

Para poder calcular la velocidad y posición de las ruedas resulta imprescindible conseguir leer los *encoders* del vehículo. Al no disponer de ningún tipo de documentación, lo primero que hay que hacer es identificar el modelo de sensor que incorpora el robot. En la Figura 25 se muestra el *encoder* del todoterreno con, una pegatina que indica su modelo (Baumer BHK 16.05-500-E6-5).

La información extraída del manual es la siguiente:

- *Encoder* incremental con canales A, B y paso por cero.
- 500 pulsos por vuelta.
- Requiere alimentación a 5 voltios.
- Señales de salida a 5 voltios.

Sin embargo, a la hora de la hora de realizar las pruebas en el todoterreno, se leyeron unos 85000 pulsos por vuelta. La gran cantidad de pulsos por vuelta que finalmente se obtuvo es debida a que los motores incorporan un reductor. A pesar de esto, finalmente fue posible leer correctamente los sensores.

Figura 25. Encoder del todoterreno

En la Figura 26 se muestra el esquema de conexión del módulo de lectura de *encoders* WAGO, mientras que en la Figura 27, la referencia de conexión de los sensores, donde indican la asignación de colores para los cables de cada señal. A partir de aquí, con ayuda de un osciloscopio se comprobaron estas señales calzando con un gato al todoterreno y haciendo girar manualmente una de sus ruedas.

La conexión que trae el robot por defecto es como sigue, los canales A, B y C están llevados a sus respectivas entradas; la alimentación se toma de la salida de 24V y la masa del sensor esta llevada a la entrada *"Latch"*. Se comprobó que todos los cables llevaban la señal correspondiente al color asignado en el manual de los *encoders* (Figura 27)

Para poder leer estas señales con el *LaunchPad*, simplemente se conectaron tres cables al módulo *WAGO* junto con los que ya había previamente, concretamente a las señales A, B y masa, dichos cables son los que finalmente utiliza el *LaunchPad* para leer las señales.

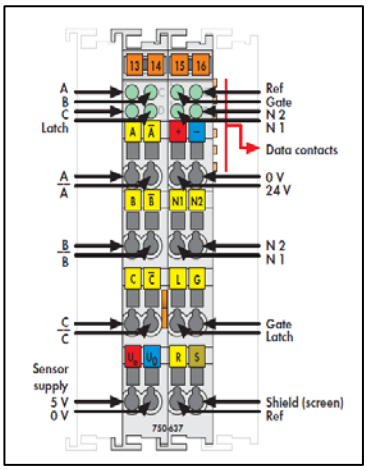

Figura 26. Esquema de conexión del módulo de lectura de encoders

for connection reference **-5**			
05A		**24K**	
Core colour	Signals	Core colour	Signals
brown	+Vs	brown	+Vs
green	CHA	green	CHA
red	CHA compl.	–	–
yellow	CHB	yellow	CHB
blue	CHB compl.	–	–
pink	CHN	pink	CHN
grey	CHN compl.	–	–
white	0 V	white	0 V
Cable data	8 x 0.14 mm²	5 x 0.14 mm²	
Screen	connected to flange		

Figura 27. Referencia de conexión de encoders

En las etapas finales del proyecto, surgió la difícil y delicada problemática de que las referencias de tensión que utiliza el *Robucar* por defecto para los *encoders* y los módulos de potencia de las ruedas no es la misma, por consiguiente, resultó imposible conectar la señal PWM a los motores a la vez que los módulos de lectura de *encoders* del *LaunchPad* al módulo WAGO para leerlos. Para afrontar este obstáculo, preparamos nuestros propios conectores siguiendo el esquema de la Figura 28.

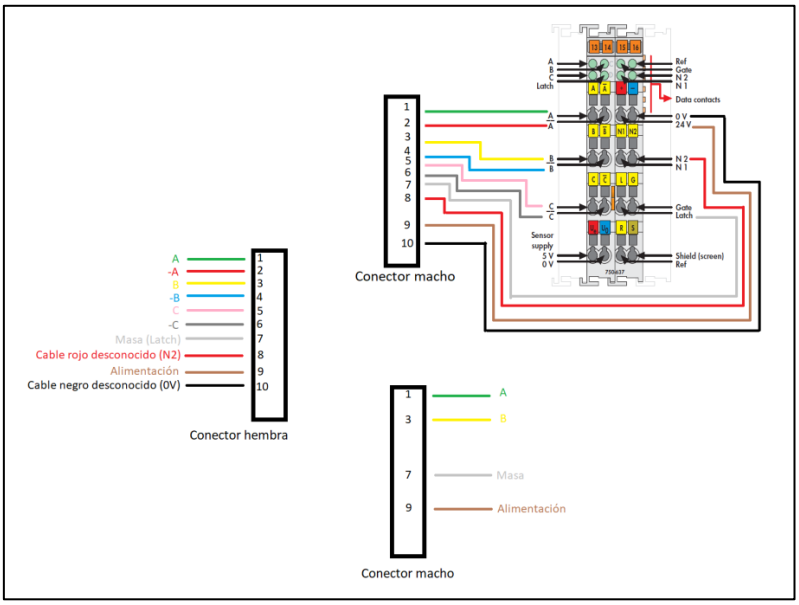

Figura 28. Esquema conector encoders

De esta manera, se busca conectar de una manera sencilla los *encoders* a la misma referencia de tensión que el PWM de los drivers de potencia. Además, gracias a los conectores, es muy sencillo alternar entre nuestro sistema y el que venía por defecto en el robot, cumpliendo así el requisito de no afectar al sistema inicial del *Robucar*. Se muestran en la Figura 29.

Figura 29. Conectores circulares para los *encoders*

5.1.3.3 Actuación sobre los drivers de potencia

5.1.3.4 Drivers de velocidad

Figura 30. Drivers de potencia de las ruedas

Nuevamente, al no disponer de información previa sobre el funcionamiento de los drivers de potencia, lo primero que se hizo fue identificarlos. Se muestran en la Figura 30, concretamente son el modelo 1227-4102 de Curtis Instruments.

Al no disponer del programador para los drivers, es necesario entender el montaje que trae por defecto el robot para poder hacerlos funcionar. Concretamente, trae conectados los conectores 1, 2, 3, 4, 8 y 15 del bornero, a continuación se indica para que se utilizan.

La Figura 31 extraída del manual del driver [4] detalla la conexión para poder controlar las ruedas con una señal externa entre 0 y 5V (el PWM generado con el *LaunchPad*). Dicha señal, la cual debe estar referenciada al menos de la batería, se lleva directamente al conector dos,

mientras que para este modo de operación es necesario conectar entre 1 y 3 una resistencia de 4,7KΩ.

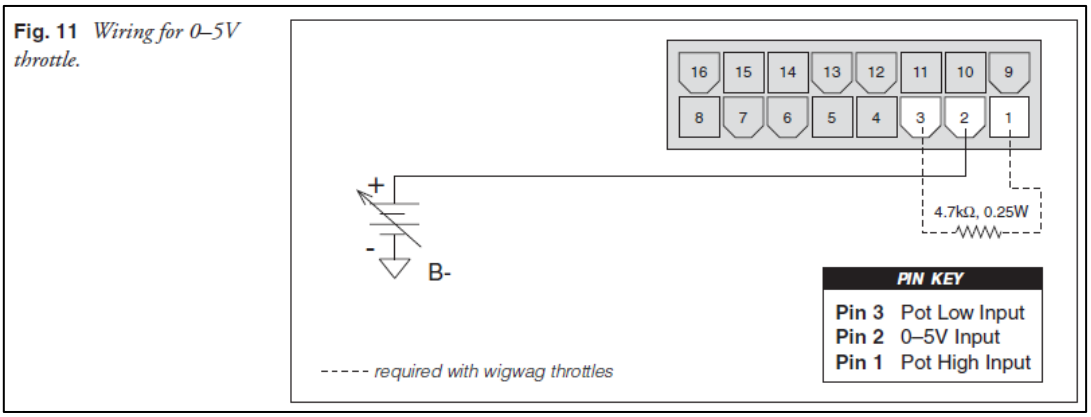

Figura 31. Esquema de conexión del bornero del driver de potencia para control con señal externa entre 0 y 5V

Para permitir que el robot pueda alcanzar su velocidad máxima, es necesario conectar directamente los terminales 1 y 4 (Figura 32). Está velocidad máxima es programada de fábrica.

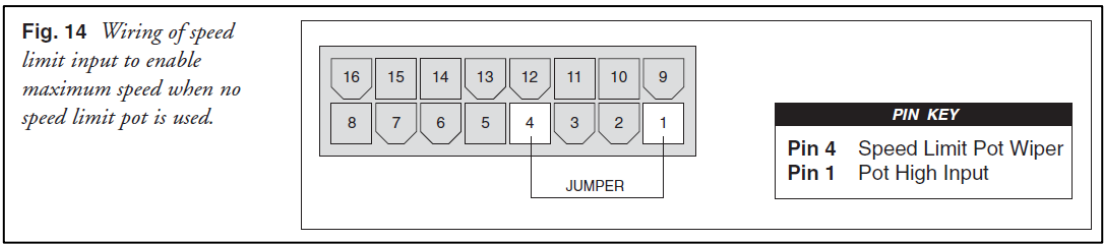

Figura 32. Habilitación de velocidad máxima

Respecto al pin 8, es el *main contactor* y el 15 es la alimentación del driver (el positivo de la batería).

Para facilitar las labores de montaje y testeo sobre los drivers, montamos nuestro propio conector *16-pin Molex Mini-Fit Jr* (el que usan los drivers) mostrado en la Figura 33.

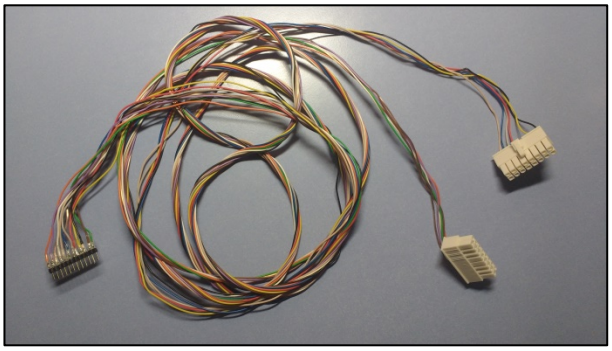

Figura 33. Conector *Molex* completo

Ya con el conector montado se realizan las conexiones indicadas anteriormente, cabe destacar que para que todo funcione, es necesario llevar 2,5 voltios al pin 2 antes de encender el robot para evitar que el sistema de seguridad bloquee los drivers.

5.1.3.5 Driver de posición de la dirección

El driver (Figura 35) es algo más sencillo que el anterior, como se muestra en la Figura 34 extraída del manual del driver [5]. Es en esencia, un puente en H. Para su funcionamiento basta con conectar el PWM del *LaunchPad* a su entrada PWM IN+ y un pin que se ponga en alto o en bajo según corresponda para la dirección en la entrada DIR IN+, a las entradas PWM IN- y DIR IN- se lleva la masa del *LaunchPad*.

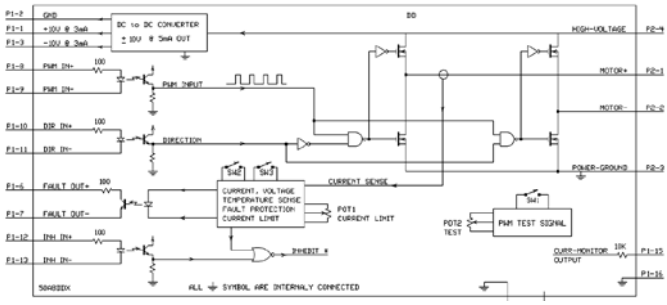

Figura 34. Diagrama de bloques del driver

Figura 35. Driver para la dirección de giro de las ruedas

5.1.4 Identificación de la dinámica del robot

Para el cálculo de los reguladores ha sido necesario realizar un experimento para la identificación de la dinámica del robot. Para realizar dicha caracterización, se ha colocado el *Robucar* en una zona libre de obstáculos y se le ha hecho avanzar con una tensión de 0.32 V voltios durante 6 segundos. Tras lo cual se ha representado la velocidad frente al tiempo obteniéndose la gráfica de la Figura 36.

Es necesario aclarar que debido a problemas de cableado, finalmente no se consiguió unificar la referencia de los *encoders* con la referencia de los PWM (más detalles en el apartado "5.1.3.2 Lectura de los encoders"). Entonces, la tensión de 0.32 V anteriormente mencionada se suministró con el sistema que traía por defecto el robot. Las medidas de velocidad se realizaron directamente con el *LaunchPad*.

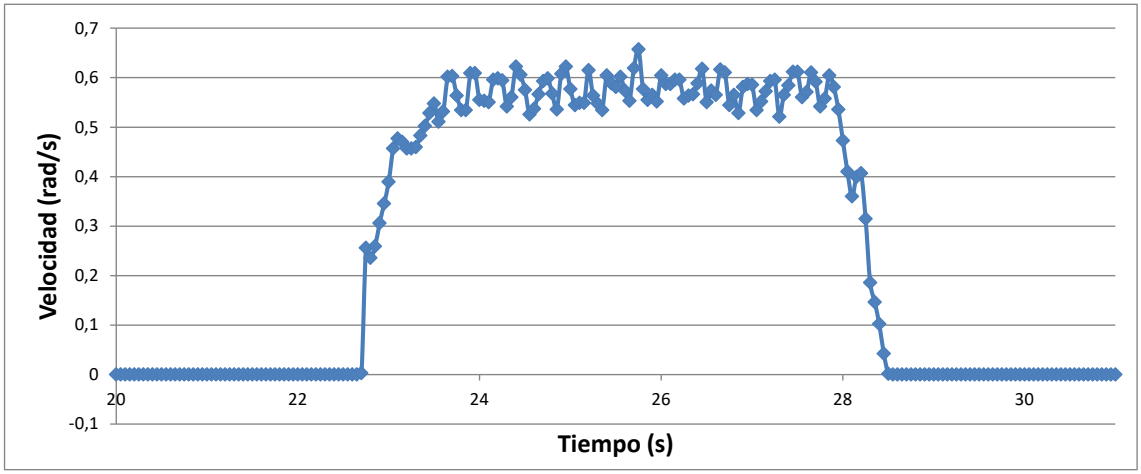

Figura 36. Gráfica resultado del experimento de identificación

De la gráfica se obtienen los siguientes datos:

- La ganancia k del sistema: $k = \frac{Velocidad\ en\ permanente}{Consigna\ de\ tensión} = \frac{0.57\ rad/s}{0.32\ V} = 1.78$
- Se mide un tiempo de respuesta de aproximadamente 1.2 segundos por consiguiente $\tau = \frac{1.2}{3} = 0.4$.

Entonces, la función de transferencia que relaciona la tensión de entrada en voltios a los módulos de potencia de los motores con la velocidad de las ruedas en radianes por segundo es:

$$G(s) = \frac{1.78}{1 + 0.4s}$$

De la misma manera se ha identificado el motor encargado de controlar la dirección, obteniéndose una velocidad de giro en permanente de 0.3 rad/s y un tiempo de respuesta de 0.75 segundos. Entonces:

- Ganancia $del\ motor\ k = \frac{Velocidad\ en\ permanente}{Consigna\ de\ tensión} = \frac{0.3\ rad/s}{3\ V} = 0.1$
- $\tau = \frac{0.75}{3} = 0.25$

Se obtiene la siguiente función de transferencia:

$$G(s) = \frac{0.1}{1 + 0.25s}$$

Estos parámetros y funciones se usan en el apartado “4. Estructura de los bucles de control” para el cálculo de los reguladores.

5.2 Software

5.2.1 Herramientas necesarias

El programa se va a desarrollar para el *LaunchPad F28377S de Texas Instruments*. Para poder comenzar a trabajar sobre dicha plataforma es necesario reunir y configurar un conjunto de herramientas. A continuación se recopila todo lo necesario para poder empezar a programar el sistema de control del todo terreno.

5.2.1.1 Entorno de programación.

CodeComposerStudio (CCS) es el entorno de programación para dispositivos de *Texas Instruments*, incluye un set de herramientas para el desarrollo y depuración de aplicaciones embebidas con posibilidad de expansión por medio de complementos (*add-ons*).

Con el fin de obtener la máxima compatibilidad posible con el *LaunchPad*, se utilizará la versión 7.1.0 de *CodeComposerStudio*, la más moderna disponible a la hora de iniciar este proyecto. Aunque es posible trabajar con versiones anteriores del programa, estas no son del todo compatibles con el microcontrolador, con el *LaunchPad* en sí, o ambos, pudiendo producir los más insospechados y variopintos problemas.

Para facilitar la programación se recurre a ***controlSUITE™ for C2000™ microcontrollers*** **[6]**, un conjunto de software y ejemplos para dispositivos específicos. En este proyecto se ha utilizado la versión 3.4.5, de donde se han obtenido las librerías necesarias. Además de esto, en la página web oficial de *Texas Instruments* hay disponibles guías de instalación de estos productos además de enlaces a wikis y foros donde es posible encontrar información útil al respecto e incluso preguntar tus propias dudas.

5.2.1.2 Librerías y mapeado de la memoria

Debido a que *CodeComposer* no conoce de forma nativa la estructura de la memoria de cada microcontrolador, es necesario incluir un conjunto de librerías en el proyecto encargadas de indicar al compilador como gestionarla. Afortunadamente, *controlSuite* [6] ofrece un conjunto de librerías donde este trabajo ya está hecho.

```
MEMORY
{
PAGE 0 :  /* Program Memory */
          /* BEGIN is used for the "boot to FLASH" bootloader mode   */

   D01SARAM   : origin = 0x00B000, length = 0x001000

   /* Flash boot address */
   BEGIN    : origin = 0x080000, length = 0x000002

   /* Flash sectors */
   FLASHA  : origin = 0x080002, length = 0x001FFE  /* on-chip Flash */
   FLASHB  : origin = 0x082000, length = 0x002000  /* on-chip Flash */
   FLASHC  : origin = 0x084000, length = 0x002000  /* on-chip Flash */
   FLASHD  : origin = 0x086000, length = 0x002000  /* on-chip Flash */
   FLASHE  : origin = 0x088000, length = 0x008000  /* on-chip Flash */
   FLASHF  : origin = 0x090000, length = 0x008000  /* on-chip Flash */
   FLASHG  : origin = 0x098000, length = 0x008000  /* on-chip Flash */
   FLASHH  : origin = 0x0A0000, length = 0x008000  /* on-chip Flash */
   FLASHI  : origin = 0x0A8000, length = 0x008000  /* on-chip Flash */
```

Figura 37. Fichero TMS320F28377S.cmd

El fichero "**TMS320F28377S.cmd"** se encarga de describir la memoria y realizar las asignaciones de las secciones generadas por el compilador. Una parte de este fichero se muestra en la Figura 37.

A continuación, para poder utilizar el *LaunchPad* y sus periféricos es necesario definir los registros de cada uno, preferiblemente tal y como aparecen en el *datasheet* del micro. Para ello se recurre a las librerías mostradas en la Figura 38 (las contenidas en la carpeta *F28377S_Include*), todas ellas proporcionadas por *controlSuite* [6].

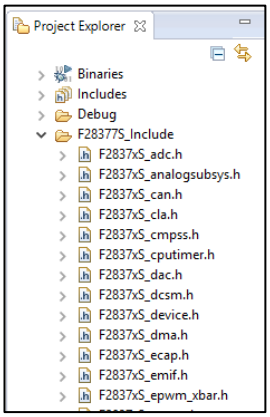

Figura 38. Librerías de registros de periféricos

```
MEMORY
{
 PAGE 0:    /* Program Memory */

 PAGE 1:    /* Data Memory */

   ADCA_RESULT   : origin = 0x000B00, length = 0x000020
   ADCB_RESULT   : origin = 0x000B20, length = 0x000020
   ADCC_RESULT   : origin = 0x000B40, length = 0x000020
   ADCD_RESULT   : origin = 0x000B60, length = 0x000020

   ADCA          : origin = 0x007400, length = 0x000080
   ADCB          : origin = 0x007480, length = 0x000080
   ADCC          : origin = 0x007500, length = 0x000080
   ADCD          : origin = 0x007580, length = 0x000080

   ANALOG_SUBSYS : origin = 0x05D180, length = 0x000080

   CANA          : origin = 0x048000, length = 0x000800
   CANB          : origin = 0x04A000, length = 0x000800

   CLA1          : origin = 0x001400, length = 0x000040

   CLB_XBAR      : origin = 0x007A40, length = 0x000040

   CMPSS1        : origin = 0x005C80, length = 0x000020
```

Figura 39. Fichero F2837Xs_Headers_BIOS.cmd

Por último es necesario incluir el fichero "**F2837Xs_Headers_BIOS.cmd"** (también proporcionado por *controlSuite* [6]) define la sección de memoria dedicada a los datos de periféricos, donde se colocaran los registros creados por las librerías de la Figura 38. Una parte de este fichero se muestra en la Figura 39.

Aparte de las ya citadas también se usaran librerías específicas para la inicialización y manejo de los periféricos, como los módulos PWM, SCI, servidores y de lectura de *encoders*. Estas son realizadas en este TFG, serán explicadas más adelante.

5.2.1.3 Sistema operativo de tiempo real

Debido a la naturaleza del proyecto se ha optado por la utilización de ***TI-RTOS***, un conjunto de herramientas entre las que se incluye un sistema operativo en tiempo real para microcontroladores de *Texas Instruments* llamado *SYS/BIOS*.

La versión de *TI-RTOS* utilizada en este proyecto es la 2.16.1.14, la más moderna disponible para la familia de microcontroladores C2000 en el momento de iniciar el trabajo. Es posible incluir únicamente con *SYS/BIOS* en sí, pero debido a problemas de compatibilidad, lo mejor es instalar íntegramente *TI-RTOS*.

En este proyecto concretamente, los módulos proporcionadas por *SYS/BIOS* que se van a utilizar son las relacionadas con la gestión del tiempo, tareas y semáforos, con el fin de coordinar de manera adecuada la ejecución de distintas tareas periódicas. En líneas generales, funcionan de la siguiente manera:

El módulo de tareas organiza el turno de ejecución de cada una basándose en la prioridad y el actual estado de ejecución. El nivel mínimo de prioridad de una terea es cero (reservado para los ciclos de espera) hasta 16.

Los semáforos son la herramienta de bloqueo básica proporcionada por *SYS/BIOS*, se utilizan para coordinar el acceso a un recurso compartido por varias tareas (en este caso, el tiempo del microcontrolador).

El módulo *clock* de *SYS/BIOS* es responsable del *tick* periódico que usa el *kernel* para contar el tiempo. La estrategia a seguir en este proyecto implica crear una instancia de reloj distinta pada cada tarea con el periodo que corresponda.

5.2.1.3.1 Configuración de SYS/BIOS

Para configurar de *SYS/BIOS*, se accede al módulo *Boot* del fichero *.cfg* (para más detalles consultar el manual de *SYS/BIOS* [7]), como se muestra en la Figura 40, es necesario marcar la casilla *"Add F2837x Boot management to my configuration*" y configurar el reloj principal del sistema, es muy importante hacerlo de esta manera en lugar de configurar los registros a mano, sino, debido a problemas de compatibilidad el reloj principal de *SYS/BIOS* no funcionará impidiendo la ejecución del programa. La configuración mostrada en la imagen es la que se utilizará en este proyecto, la cual determina una frecuencia de reloj de 200MHz. Además, se deshabilita el perro guardián y se habilita la carga del programa desde la flash.

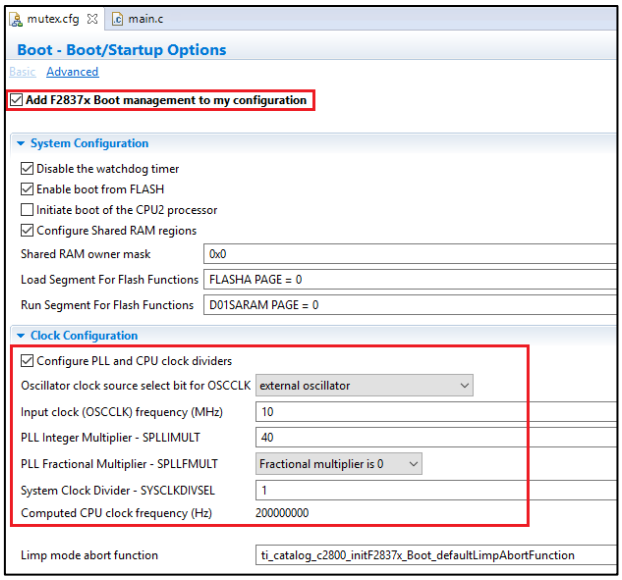

Figura 40. Configuración del módulo Boot

5.2.2 Estructura de tareas

En este apartado se pretende explicar la estructura de tareas y el funcionamiento general del programa. Para facilitar el control del sistema en tiempo real, se hace uso de las herramientas *SYS/BIOS* [7] [8] comentadas en el apartado anterior. Además, así queda abierta la posibilidad de una futura continuación del proyecto simplemente con el desarrollo y gestión de nuevas tareas sin necesidad de modificar la base del programa.

Concretamente, este proyecto consta de tres tareas que se dividirán entre los dos *LaunchPad*, una encargada de controlar la velocidad de las ruedas, otra del ángulo de giro y una tercera de la comunicación para recibir consignas del exterior (ambas tareas de control calculan también el modelo cinemático). Las variables para las velocidades de referencia están protegidas por un servidor que garantiza su correcta lectura y escritura. Esto se hace así para evitar conflictos en variables usadas por varias tareas diferentes.

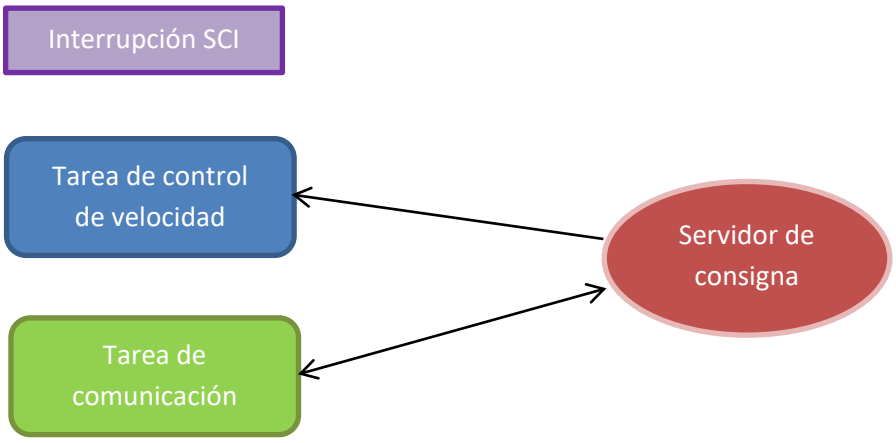

Figura 41. Tareas *LaunchPad* 1

En el montaje final se usarán dos *LaunchPad*, el primero se ocupará de regular la velocidad de las ruedas, recibirá las consignas desde el exterior y se las comunicará a la otra placa. Estas son las tareas mostradas en el esquema de la Figura 41. Respecto del segundo *LaunchPad,* este recibirá las consignas de la otra placa y regulará el ángulo de giro del *Robucar* (Figura 42).

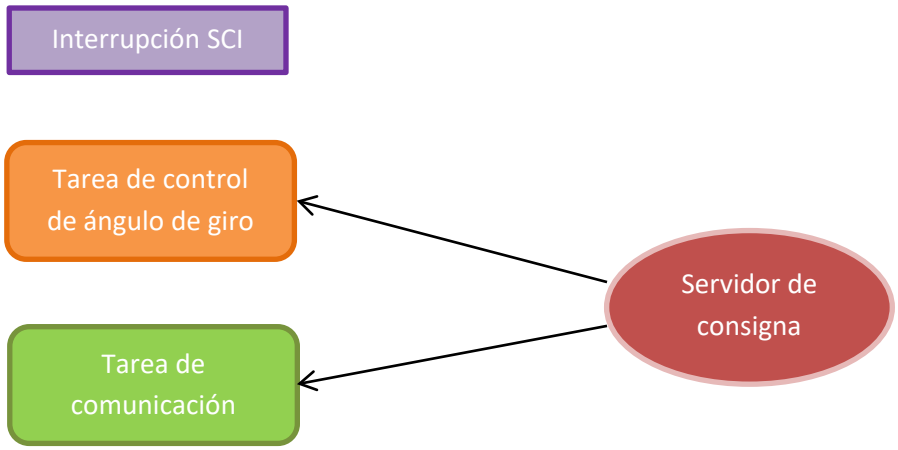

Figura 42. Tareas *LaunchPad* 2

La asignación de **prioridades** se realiza en función de la periodicidad de cada tarea, asignando las prioridades más altas a las que se ejecutan con mayor frecuencia. Respecto del *sci*, trabaja a una velocidad de 9600 bits por segundo, por lo que habrá una interrupción de prioridad hardware cada 1/9600 = 1,042 ms.

Se quiere conseguir un tiempo de respuesta del sistema de 0,5 segundos, por consiguiente, de acuerdo al teorema de Shanon, el periodo de muestro debe ser diez veces menor, 50 milisegundos. Este es el periodo con el que se ejecutarán las tareas de control de velocidad y ángulo de giro, por consiguiente, se les asignará una prioridad de 1.

Para enviar y recibir las consignas se utilizan tramas de 20 bits. Teniendo en cuenta que la *sci* recibe un dato cada 1,042ms, la tarea de comunicación será considerada esporádica con una separación mínima entre eventos de 1,042·20 ≈ 21 milisegundos.

Para dar **periodicidad** a las tareas, se utilizan las entidades de reloj y semáforos. A cada tarea se le asigna un semáforo binario, dicho semáforo es controlado por una entidad de reloj asociada a la misma tarea. Inicialmente, todas las tareas están listas para ejecutarse, cuando una tarea se ejecuta, pone a cero el contador de su semáforo, a partir de aquí, el contador de ese semáforo no se vuelve a establecer a uno hasta que expira el periodo de su reloj asociado, ejecutándose la función correspondiente a ese reloj, volviendo así a estar la tarea lista para ejecutarse.

5.2.3 Implementación de los bucles de control

5.2.3.1 Utilidades necesarias

El sistema de control necesita obtener datos sobre la velocidad y posición angular de las ruedas además de poder actuar sobre los dispositivos de potencia que controlan los motores. Para ello es necesario emplear los módulos de lectura de *encoders* y PWM. Ambos requieren una configuración previa a su uso que, por limpieza y organización, será llevada a cabo en librerías aparte del programa principal. Dichas librerías son versiones modificadas para este proyecto de las proporcionadas por el profesor José Luis Villarroel en la asignatura "Sistemas de tiempo real". En los dos siguientes apartados se detalla la configuración de estos módulos.

5.2.3.2 Configuración y uso del módulo PWM

La librería a cargo de gestionar el módulo PWM alberga las funciones, encargadas de inicializar y configurar las entidades PWM. A modo de aclaración, es posible crear varias "entidades" PWM independientes entre sí con el fin de gestionar varias salidas. El *LaunchPad* posee pines suficientes como para utilizar simultáneamente 5 de estas entidades, en este proyecto se usarán únicamente tres. Cada entidad posee dos canales, uno con la señal principal (A) y otro con la señal invertida (B).

Ambas entidades de PWM se configuran de la misma manera a excepción de la asignación de pines; a la entidad EPWM2 le corresponden los pines 80 (canal A) y 79 (canal B) del *LaunchPad* mientras que a EPWM7 se le asignan el 40 (canal A) y 39 (canal B).

En líneas generales, ambas entidades se configuran como sigue:

- Modo de contaje ascendente (up-count mode).
- Tamaño del registro de periodo establecido a 12 bits.
- Se utiliza una única base de tiempos para ambas entidades de PWM.
- El *prescaler* se establece a 4.

Hay que tener en cuenta que la frecuencia de la señal PWM debe ser mucho mayor que la frecuencia de corte del sistema a controlar. De esta manera, la señal PWM quedará atenuada a excepción de sus componentes de menor frecuencia, es decir, la tensión media. Entonces, se establece el valor del *preescaler* a 4 obteniéndose una frecuencia de 6,25 kHz. Es posible que sea necesario modificar este valor una vez identificados los motores.

Las otras funciones que componen esta librería se utilizan para modificar el valor del registro de comparación de su respectivo PWM. Estas son las funciones que se usarán en el programa principal para modificar el valor de salida de los PWM y actuar sobre los dispositivos de potencia asociados a los motores de las ruedas del todo terreno.

En principio, los valores de tensión que puede ofrecer el LaunchPad F28377S son 0 y 3,3V, sin embargo, los adaptadores de potencia del todo terreno requieren señales PWM entre 0 y 5V, por lo que es necesario realizar una adaptación de estos niveles de tensión, esta cuestión es detalla en la sección “adaptación de nivel de tensión”.

5.2.3.3 Configuración y uso de los módulos de lectura de encoders

El *LaunchPad* tiene dos módulos de lectura de *encoders* a 5V, ambos con pines para las señales A y B, el paso por cero I, alimentación y masa. El esquema del circuito de lectura se muestra en la Figura 43 y los pines de la placa real en la Figura 44. El sistema de control utilizara ambos módulos, para el control de las ruedas izquierda y derecha.

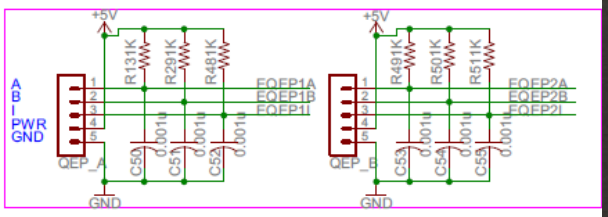

Figura 43. Esquema de módulos de lectura

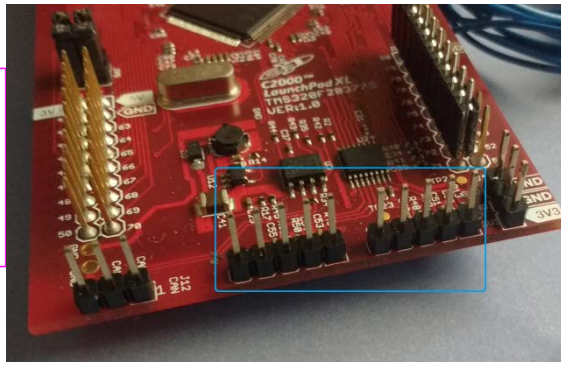
Figura 44. Pines para la lectura de encoders

Análogamente al caso del PWM, la librería a cargo de gestionar los módulos de lectura de los *encoders* contiene funciones que se encargan de la inicialización y configuración. Para facilitar el cálculo de posición y velocidad se configuran los módulos de la siguiente manera:

- Selección del modo “*cuadrature count mode*”.
- El módulo contará tanto los flancos de subida como de bajada, obteniéndose así el doble de resolución.
- Se sincronizan los módulos con el reloj del sistema.
- Se limita el valor del contador de pulsos a 85800, que son el número de pulsos que se cuentan en una vuelta (más detalles en la sección “5.2.3.5 Cálculo de posición y velocidad de las ruedas”).

Físicamente, el *encoder* instalado en el todoterreno proporciona 500 pulsos por vuelta, al utilizar sus dos canales A y B a la vez (“*cuadrature count mode*”) se contarán 1000. Al contar tanto flancos de subida como de bajada, hace un total de 2000 pulsos por vuelta. Gracias a la utilización de los dos canales del *encoder*, el microcontrolador puede determinar el sentido de giro del motor sin ninguna configuración adicional, ofreciendo dicho resultado en un registro destinado exclusivamente a tal fin.

Las otras dos funciones que componen esta librería son se utilizan para obtener el valor del contador de pulsos en el momento de llamar a la función. Estas funciones son las que se

utilizarán en el programa principal para realizar el cálculo de velocidad de las ruedas necesario para el control.

Para evitar problemas en la lectura del sensor, es importante destacar que la llamada a las funciones de lectura se realizara una única vez por tarea y su valor será almacenado en una variable, que será la que use el programa de control para realizar los cálculos. Esto es debido a la velocidad del sensor en sí.

5.2.3.4 Funciones de apoyo

El programa principal de control de las ruedas, además de la configuración de las utilidades explicadas en los apartados anteriores, necesita unas funciones de apoyo que serán utilizadas recurrentemente. Concretamente, se trata de las funciones encargadas del cálculo de velocidad de los motores y de calcular el regulador PI (o solo P en el caso de control de posición).

5.2.3.5 Cálculo de posición y velocidad de las ruedas

Este apartado está inspirado en el trabajo realizado en la asignatura electrónica industrial [9], donde se realizó el control de un motor de imanes permanentes. Se pretende utilizar la información proporcionada por el *encoder* para determinar en todo momento la posición y la velocidad de las ruedas. El *encoder* es del tipo incremental, que da un total de 500*4=2000 pulsos por vuelta. Esto es debido a que el *encoder* tiene dos líneas distintas desfasadas 90º como se muestra en la Figura 45, cada una proporciona 500 pulsos, por lo que al combinarlas contando tanto flancos de bajada como de subida se obtienen los 2000 pulsos por vuelta.

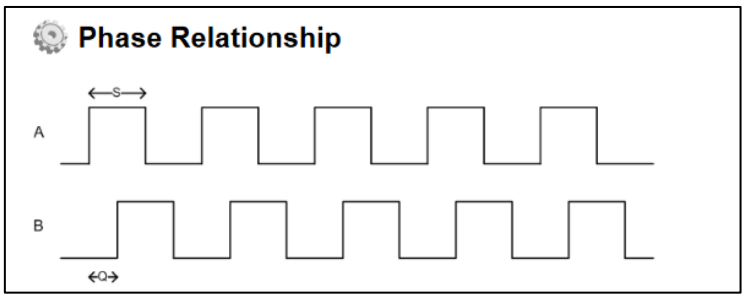

Figura 45. Señales del encoder

Tal como se han configurado los módulos de lectura de los *encoders*, el valor máximo de los registros del micro que cuentan las vueltas es de 2000 pulsos, se reinician al superar dicho valor o vuelve a 2000 si se está en la posición 0 con velocidad negativa. De esta manera se facilita en gran medida el cálculo de posición y velocidad. Entonces, cuando el registro contiene un valor de cero quiere decir que la rueda se encuentra en el origen (0 radianes), mientras que cuando dicho valor es 2000, quiere decir que la rueda se encuentra en la posición 2π.

Nº pulsos registro *encoder*	Angulo girado en radianes
0	0
85800	2π

Por consiguiente, para obtener la posición en radianes de las ruedas basta con calcular la siguiente regla de tres:

$$Ángulo_{Radianes} = \frac{2\pi}{85800} * Pulsos_{encoder}$$

De esta manera, la variable que contiene la posición de la rueda queda sencillamente acotada entre 0 y 2 π radianes.

Una vez que conocemos la posición, para calcular la velocidad solo hay que medir el incremento de ángulo girado durante un periodo de muestreo. Además se ha de tener en cuenta que es posible que el contador se reinicie cuando la rueda de una vuelta completa, por lo que habrá que identificar y gestionar estas situaciones en el programa.

Este método necesita conocer el sentido de giro de las ruedas, afortunadamente, el módulo de lectura del *encoder* calcula por su cuenta el sentido de giro y lo almacena en un registro. Con un sentido de giro según las agujas del reloj, devuelve un 1, con un sentido contrario a las agujas del reloj, devuelve un cero. Como aclaración, la velocidad de avance positiva de la rueda, a la hora de concebir este método, se consideró del sentido de las agujas der reloj, que es al revés del sentido creciente de los ángulos en grados o en radianes, lo cual quiere decir que este método interpreta los ángulos "al revés", sin embargo, esta cuestión es indiferente en el cálculo de la velocidad porque lo que se busca es una diferencia de ángulos.

Para identificar mejor el problema que se quiere programar, se procede a dibujar el valor de la posición de la rueda a una velocidad cualquiera frente al tiempo.

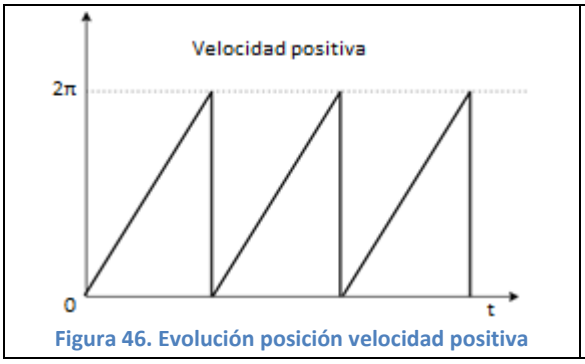

Figura 46. Evolución posición velocidad positiva

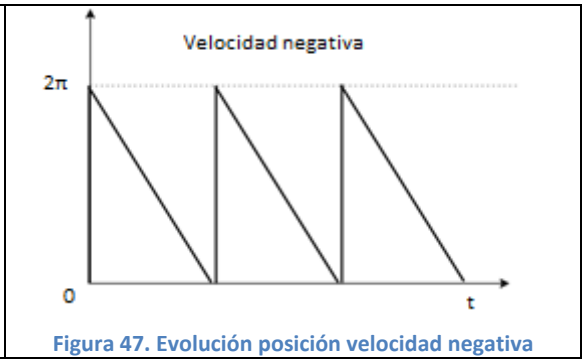

Figura 47. Evolución posición velocidad negativa

Entonces, la forma de actuar es la siguiente, si la rueda está avanzando con el sentido de las agujas del reloj (Figura 46):

- Normalmente, la velocidad se calcula como el ángulo recorrido durante el anterior periodo de muestreo dividido entre el tiempo que dura (0,05 segundos).

- Tarde o temprano el valor de la posición medida por el *encoder* se reiniciará al dar la rueda una vuelta completa mientras que el valor de la posición en el anterior periodo de muestreo no se haya reiniciado. Esto se detecta comparando ambos valores antes de realizar el cálculo de velocidad. Para compensar este desajuste, es necesario sumar 2π al valor de la posición actual a la hora de calcular la diferencia de ángulos.

Cuando la rueda avanza en sentido contrario a las agujas del reloj se procede de manera similar (Figura 47):

- Si no se produce desajuste, la velocidad se sigue calculando como el ángulo recorrido durante el anterior periodo de muestreo dividido entre el tiempo que dura.
- Cuando se produce el desajuste hay que restar 2π al valor de la posición actual a la hora de calcular la diferencia de ángulos. En este caso es la posición actual la que se reinicia mientras que la posición anterior no se ha reiniciado.

Para finalizar este apartado se muestra en la Figura 48 la implementación del cálculo de velocidad y posición, como aclaración, la variable tita0 es la posición en el instante actual y tita en el anterior periodo de muestreo.

```
////////////////////////////////////////////////////////////////////////////////
//////   Cálculo de la velocidad real en rad/s
////////////////////////////////////////////////////////////////////////////////
float cal_velocidad(float encoder,Uint_32 direccion, float *tita0, float *tita)
{
    float w, dtita=0;
    *tita0=0.00314316423*encoder;          // Regla de tres para calcular la posición de las ruedas

    if(direccion==1)
    {
        if(*tita0<*tita)
        {
            dtita=*tita0+dospi-*tita;      // Ajuste y cálculo de la velocidad en sentido positivo

        }else
        {
            dtita=*tita0-*tita;
        }
    }
    else
    {
        if(*tita0<=*tita)
        {
            dtita=*tita0-*tita;
        }else
        {
            dtita=*tita0-(*tita+dospi);    // Ajuste y cálculo de la velocidad en sentido negativo
        }
    }
    w = dtita/0.05;                        // Cálculo de la velocidad como espacio entre tiempo
    *tita=*tita0;
    return w;
}
```

Figura 48. Cálculo de velocidad

Es necesario indicar, que esta función devuelve únicamente el valor de la velocidad en radianes por segundo. Para que el resto del programa disponga de unos valores actualizados de la posición, es necesario introducir las variables tita0 y tita como punteros.

5.2.3.6 *Implementación de los reguladores*

En este apartado se explica la implementación de los reguladores calculados en el apartado "4. Estructura de los bucles de control".

Regulador PI de velocidad

Primero se realizará la discretización del regulador por medio del método de rectángulos mayorantes y después se planteará su ecuación en diferencias.

Rectángulos mayorantes implica sustituir s por $\frac{z-1}{Tz}$, siento T el periodo de muestro e igual a 0,5 milisegundos. Se quiere conseguir un tiempo de respuesta del sistema de 0,5 segundos, por consiguiente, de acuerdo al teorema de Shanon, el periodo de muestro debe ser diez veces menor, 50 milisegundos.

$$R(s) = k \cdot \frac{1+\tau_i s}{\tau_i s} = 0.1921 \frac{1+0.4s}{0.4s}$$

$$R(z) = k \cdot \frac{\tau_i z + Tz - \tau_i}{\tau_i (z-1)} = k \cdot \frac{\tau_i + T - \tau_i z^{-1}}{\tau_i (1-z^{-1})} = 0.48 \cdot \frac{0.45 - 0.4z^{-1}}{1-z^{-1}} = \frac{U(z)}{E(z)}$$

La ecuación en diferencias resulta:

$$U(k) = U(k-1) + 0.216E(k) - 0.192E(k-1)$$

El código del regulador para ambas ruedas queda como sigue:

```
float PI_Izda(float Wref, float W)
{
    E_Izda = Wref - W ;
    U_Izda = U_Izda + 0.216*E_Izda - 0.192*E_ant_Izda ;
    E_ant_Izda = E_Izda ;

    return U_Izda ;
}
```

Figura 49. Regulador rueda izquierda

```
float PI_Dcha(float Wref, float W)
{
    E_Dcha = Wref - W ;
    U_Dcha = U_Dcha + 0.216*E_Dcha - 0.192*E_ant_Dcha ;
    E_ant_Dcha = E_Dcha ;

    return U_Dcha ;
}
```

Figura 50. Regulador rueda derecha

Regulador proporcional de posición

En este caso basta con multiplicar el error de cada periodo de muestreo por la constante calculada en el apartado "4. Estructura de los bucles de control". La Figura 51 muestra el código.

$$U(k) = 225.625(k)$$

```
float P(float TitaRef, float Tita)
{
    E_Tita = TitaRef - Tita ;
    U_Tita = 20.28*E_Tita ;

    return U_Tita ;
}
```

Figura 51. Regulador proporcional de posición

5.2.3.7 Funcionamiento del bucle

Una vez preparadas las funciones para el cálculo de velocidad y posición de las ruedas y el regulador PI, ya es posible implementar la tarea de control. En la Figura 52 se muestra el control de velocidad de la rueda izquierda.

Para realizar el control de la rueda derecha se puede utilizar la misma estructura, lo único que hay que modificar son los parámetros con los que se llama a las funciones de cálculo de velocidad y regulador PI, utilizando los propios de la rueda derecha. Análogamente, para controlar posición, basta con modificar las entradas del regulador PI, sustituyendo velocidad por posición y hacer nula la acción integral.

Como se menciona al final de los apartados "5.2.3.5. cálculo de velocidad y posición de las ruedas" y "5.2.3.6. regulador PI", a la hora de llamar a la función de cálculo de velocidad es necesario darle las variables de la posición actual y anterior como punteros para que, tras el cálculo, el resto del programa disponga de dichas variables actualizadas. Lo mismo pasa con la función que calcula el regulador PI, es necesario introducir el actual estado de la integral como puntero para que se mantenga actualizado.

A continuación se explica más en detalle la tarea de control dividiéndola en bloques de código según sus distintos propósitos.

```
*  ======== Tarea de control ========
*/
Void Control(UArg arg0, UArg arg1)
{
    Init_EPWM2 () ;
    Init_Encoder_left ();
    GpioDataRegs.GPASET.bit.GPIO15 = 1;

    for (;;) {
        Semaphore_pend(Control_Sem, BIOS_WAIT_FOREVER);

/*==============CONTROL DE VELOCIDAD RUEDA IZQUERDA=================*/

        encoder_left=Get_Counter_left ();
        direncoder_left = EQep3Regs.QEPSTS.bit.QDF;

        //Lazo de Velocidad
        w_left=cal_velocidad(encoder_left, direncoder_left, &tita0_left, &tita_left);
        accion_left = PI(wref_left, w_left, kpw_left, kiw_left, Vmax, Vmin, Control_Periodo, &Iw_left);
        PWM_Left(accion_left); //en voltios

        if(accion_left<0){
            GpioDataRegs.GPBCLEAR.bit.GPIO58 = 1;
        } else{
            GpioDataRegs.GPBSET.bit.GPIO58 = 1;
        }
    }
}

Void Control_Clock(UArg arg0)
{
    Semaphore_post(Control_Sem);
}
```

1 2 3 4 5 6

Figura 52. Tarea de control detallada

1) Este bloque solo se ejecutará una vez, realiza una llamada a las funciones de configuración del módulo de lectura de *encoders* y señal PWM correspondientes para prepararlos para su uso en el resto de la tarea.
2) Semáforo asociado a la tarea de control, se encarga de que la tarea se ejecute periódicamente de acuerdo al periodo asignado al reloj que controla el semáforo.
3) Lo primero que se debe hacer durante una ejecución normal de la tarea es leer el valor del contador del *encoder* y su sentido de giro. Para evitar problemas con la velocidad de lectura del sensor, el valor de su registro se lee una única vez por cada ejecución de la tarea.
4) El esquema típico de un control con un regulador PI, primero se calcula la velocidad de la rueda a partir de la información proporcionada por el *encoder*, después se llama a la función del PI para que calcule la acción necesaria, y finalmente se aplica dicha acción por medio de la señal PWM. Esta señal es la que ataca directamente a los módulos de potencia del todo terreno.
5) En función del sentido de la acción se pone un pin del *LaunchPad* en alto o en bajo.
6) Tras una completa ejecución de la tarea, se espera a que termine el periodo de su reloj asignado para que vuelva a habilitar el semáforo.

6 PRUEBAS

6.1 Pruebas con motor lego

Una vez que se ha terminado de programar el bucle de control, como paso previo a los ensayos en el vehículo todoterreno, se testea el programa sobre un pequeño motor de *Lego Mindstorm* Figura 53. Como no es posible realizar una conexión directa entre el *LaunchPad* y el motor, es necesario preparar algunos componentes electrónicos para realizar dicha conexión, los problemas a abordar son:

- El *LaunchPad* no puede suministrar toda la corriente necesaria si es alimentado únicamente por la conexión USB, por lo que será necesaria alimentación externa.
- Los módulos de lectura de *encoders* no son capaces de leer directamente el que lleva incorporado el motor Lego, será necesario un circuito de adaptación de impedancias.
- La placa solo es capaz de suministrar tensiones positivas, para poder controlar completamente el motor, será necesaria la utilización de un puente en H.

Figura 53. Motor Lego Mindstorm

6.1.1 Circuito de conexión motor lego

Es necesario montar un circuito para la conexión entre dicho motor y el *LaunchPad*.

6.1.1.1 Material necesario

6.1.1.2 Puente en H

Figura 54. Puente en H

El motor Lego se controla mediante una señal PWM (Figura 54) entre -5 y 5V, gracias a lo cual puede girar en ambos sentidos. El *LaunchPad* solo es capaz de generar estas señales entre 0 y 3,3V, por consiguiente, es necesario utilizar un puente en H para adaptar la señal del *LaunchPad*.

Se utilizara el integrado LB1836M de ON Semiconductor, cuyo encapsulado es del tipo 14-pin MFP. Para poder montarlo en la placa blanca de montaje rápido junto al resto de los componentes, hay que soldarlo a un adaptador a 14-DIP.

6.1.1.3 Circuito de adaptación de impedancias

Para posibilitar la lectura de las señales proporcionadas por el *encoder* del motor Lego se necesita hacer uso del integrado para la adaptación de impedancias mostrado en la Figura 55.

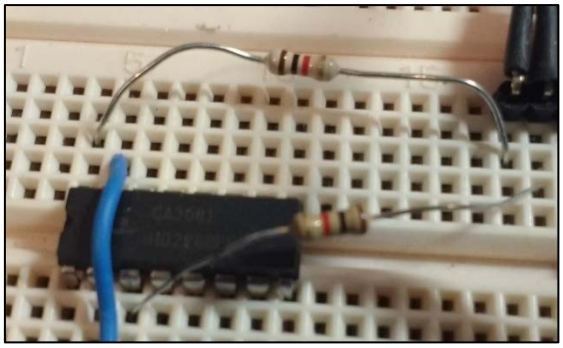

Figura 55. Circuito de adaptación de impedancias

6.1.1.4 Circuito final

El circuito final que permitirá la conexión entre el *LaunchPad* y el motor Lego se muestra en la Figura 56.

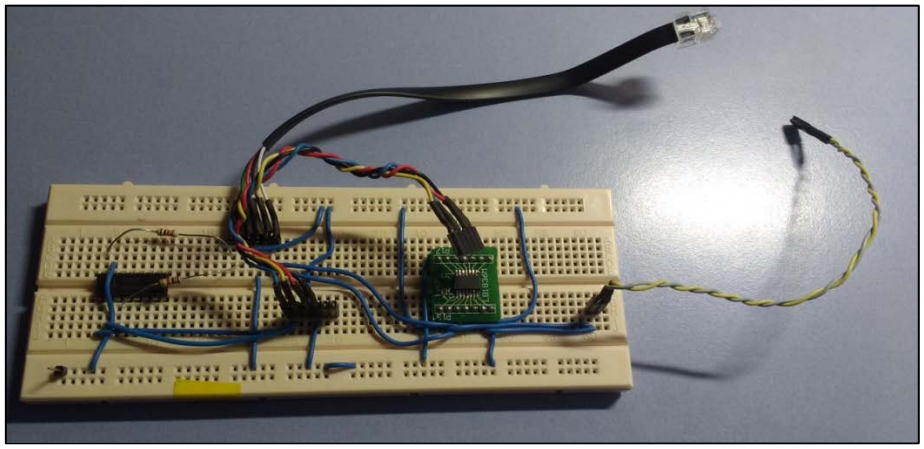

Figura 56. Circuito final de conexión entre el motor Lego y el LaunchPad

6.1.2 Ensayo de control de posición

Se modifica desde el *debugger* es la consigna de posición del motor, se observa como el motor se para al llegar a la posición deseada-

La Figura 57 muestra el control de posición sobre el motor lego para consignas de posición 0, 1, 2, 3, 2, 1, 0 radianes en ese orden.

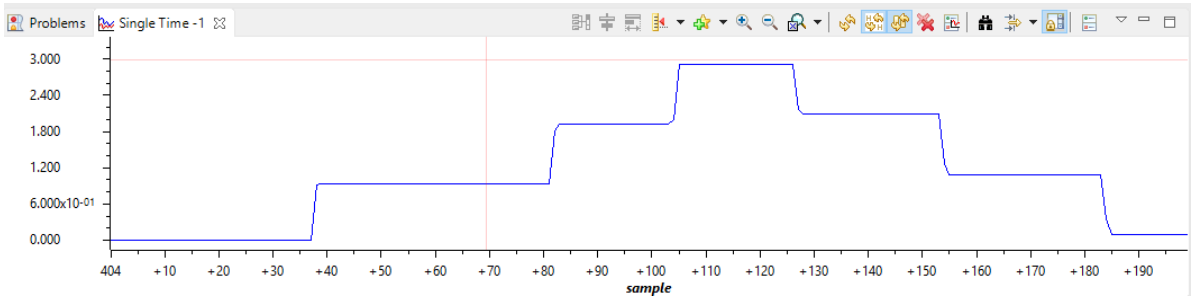

Figura 57. Control de posición del motor lego con varias consignas

6.1.3 Ensayo de control de velocidad

Para el control de velocidad, se envía una consigna en radianes por segundo. La Figura 58 y la Figura 59 muestran el funcionamiento del control para dos consignas distintas.

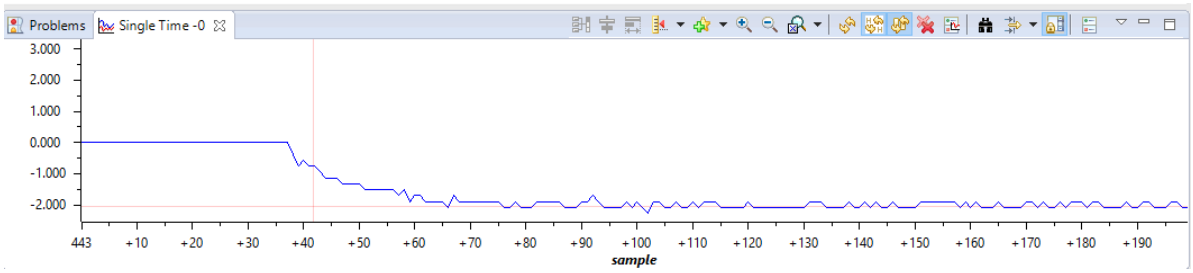

Figura 58. Control de velocidad del motor lego con Wref = -2rad/s

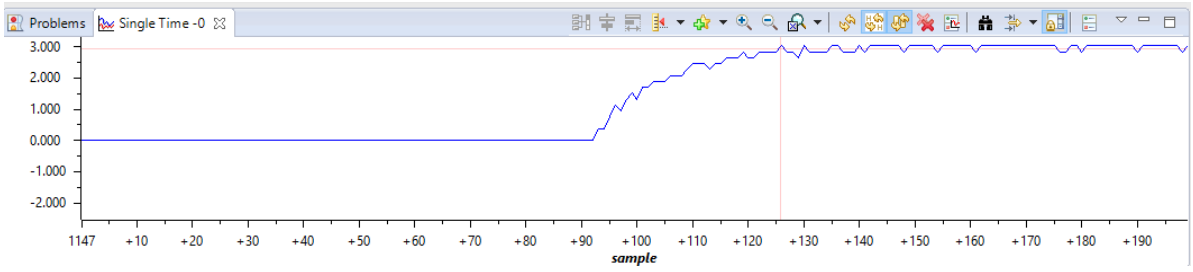

Figura 59. Control de velocidad del motor lego con Wref = 3rad/s

7 CONCLUSIONES

Finalmente, se ha conseguido preparar el software necesario para realizar el sistema de control. En síntesis, se ha preparado el código para generar una estructura de tareas con SYS/BIOS, el cálculo de velocidad de los motores, el cálculo de los reguladores y la generación de señales PWM.

Respecto al hardware, se ha preparado el necesario para la conexión entre el *LaunchPad F28377S de Texas Instruments* y el *Robucar*. A fecha de depósito de esta memoria, se pudo dar consigna de tensión a los motores del todoterreno y leer los *encoders*, pero no ambas cosas a la vez debido a un problema de tierras. Esto es debido a que la señal PWM de control de los motores debe de estar referenciada al menos de la batería de los drivers de potencia mientras que los *encoders* están cableados a otra masa distinta. A pesar de ello fue posible identificar la dinámica del robot.

Durante las pruebas sobre el todoterreno real, se ha echado muy en falta documentación sobre el vehículo, ha sido todo un desafío hacer las averiguaciones necesarias para poder conectar el sistema de control. En conclusión, se ha aprendido el valor de documentar bien el trabajo para posibles revisiones futuras.

Respecto a la parte de programación, la parte que más tiempo se llevó fue la preparación de la base del programa, más que el sistema de control en sí. Durante la carrera, en las asignaturas en las que se requiere trabajar con microcontroladores, los profesores siempre proporcionaban la base para empezar a programar debido a su falta de tiempo, interés académico y el estar fuera del alcance de las asignaturas. Resulto ciertamente complicado reunir todas las herramientas y librerías en sus versiones adecuadas para poder empezar a trabajar.

8 BIBLIOGRAFÍA

[1] C. Gracia: “Modelado y Simulación de un robot Robucar TT”. Proyecto de fin de carrera en la Universidad de Zaragoza, Mayo 2013.

[2] Texas Instruments: “LAUNCHXL F28377S User’s Guide”. http://www.ti.com/lit/ug/sprui25c/sprui25c.pdf

[3] Texas Instruments: “TMS320F2837xS Delfino Microcontrollers Technical Reference Manual”. http://www.ti.com/lit/ug/spruhx5e/spruhx5e.pdf

[4] Curtis Instruments: “Manual”. http://curtisinstruments.com/Uploads/DataSheets/1223%20(11C).pdf

[5] ADVANCED Motion Controls: “Datasheet 30A8DD PWM servo amplifier”. https://www.servo2go.com/support/downloads/Advanced_Motion_Controls_30a8dd.pdf

[6] Texas Instruments: “controlSUITE™ for C2000™ microcontrollers”. http://www.ti.com/tool/CONTROLSUITE

[7] Texas Instruments: “SYS/BIOS User’s Guide”. http://processors.wiki.ti.com/index.php/Category:SYSBIOS.

[8] J.L Villarroel: “Material de las asignaturas del grado en ingeniería electrónica y automática robots autónomos y sistemas de tiempo real”. Plataforma digital de la Universidad de Zaragoza, curso 2016-2017.

[9] E. Oyarbide: “Material de la asignatura del grado en ingeniería electrónica electrónica industrial”. Plataforma digital de la Universidad de Zaragoza, curso 2016-2017.

ANEXO

CÓDIGO

```
#include <xdc/std.h>
#include <xdc/runtime/System.h>

#include <ti/sysbios/BIOS.h>
#include <ti/sysbios/knl/Clock.h>
#include <ti/sysbios/knl/Task.h>
#include <ti/sysbios/knl/Semaphore.h>

#include <math.h>

#include <xdc/cfg/global.h>
#include <ti/uia/events/UIABenchmark.h>

#include "F2837xS_Device.h"
#include "pwm.h"
#include "servidores.h"
#include "servos.h"
#include "encoder.h"

//==============DEFINICIÓN TAREAS====================

//Definiciones tarea de control

Void Control(UArg arg0, UArg arg1);
Void Control_Clock(UArg arg0);

Void Posicion(UArg arg0, UArg arg1);
Void Posicion_Clock(UArg arg0);

Semaphore_Handle Control_Sem;
Task_Handle Control_Handle;
Clock_Handle Control_Reloj;

Semaphore_Handle Posicion_Sem;
Task_Handle Posicion_Handle;
Clock_Handle Posicion_Reloj;

//==============DEFINICIÓN VARIABLES====================

float Control_Periodo=50;
float Posicion_Periodo=50;
float Vmax=5;
float Vmin=0;
int contador=0;
#define dospi 6.283185

//Variables rueda IZQUIERDA


float w_left=0;                //Velocidad real (angular rad/s)
float wref_left=0;             //Velocidad referencia (angular rad/s)
float accion_left=2.5;             //Accion
float tita0_left=0;            //Posición rueda
float tita_left=0;             //Posición rueda anterior
Uint_32 encoder_left=0;          //Pulsos encoder
Uint_32 direncoder_left=0;     //Dirección de giro



//Variebles rueda DERECHA


float w_right=0;                //Velocidad real (angular rad/s)
float wref_right=0;             //Velocidad referencia (angular rad/s)
float accion_right=2.5;             //Accion
float tita0_right=0;            //Posición rueda
float tita_right=0;             //Posición rueda anterior
Uint_32 encoder_right=0;          //Pulsos encoder
```

```
Uint_32 direncoder_right=0;     //Dirección de giro


//Variables REGULADORES

float E_Izda = 0.0, E_ant_Izda = 0.0, U_Izda = 0.0 ;
float E_Dcha = 0.0, E_ant_Dcha = 0.0, U_Dcha = 0.0 ;
float E_Tita = 0.0, E_ant_Tita = 0.0, U_Tita = 0.0 ;

//==============VARIABLES MODELO CINEMÁTICO====================

float v_coche=0;
float w_coche=0;
float tita_coche;

float v_i = 0;
float v_d = 0;
float w_i = 0;
float w_d = 0;
float R = 0; //radio de giro del robot (v/w)

float L=1.2;   // en metros, longitud de los ejes del robot
float S=1.2;   // en metros, separación entre ejes
float r=0.3;   // en metros, radio de las ruedas

//==============DEFINICIÓN DE FUNCIONES====================


void Init_PeripheralClocks(void);
void Gpio_select(void);

void PWM_Left (float U);
void PWM_Right (float U);
float cal_velocidad(float encoder,Uint_32 direccion, float *tita0, float *tita);
float PI_Izda(float Wref, float W);
float PI_Dcha(float Wref, float W);
float P(float TitaRef, float Tita);

/*
 *  ======== main ========
 */
Int main()
{
/* Creación tarea de control de velocidad*/

    Task_Params taskParams;

    Control_Sem = Semaphore_create(1, NULL, NULL);

    Task_Params_init(&taskParams);
    taskParams.priority = 1;
    Control_Handle = Task_create (Control, &taskParams, NULL);

    Clock_Params clkParams;

    Clock_Params_init(&clkParams);
    clkParams.period = Control_Periodo;
    clkParams.startFlag = TRUE;
    Control_Reloj=Clock_create(Control_Clock, 5, &clkParams, NULL);

/* Creación tarea de control de posición de la dirección*/

    Task_Params_init(&taskParams) ;

    Posicion_Sem = Semaphore_create(1, NULL, NULL);

    Task_Params_init(&taskParams);
    taskParams.priority = 1;
```

```
    Posicion_Handle = Task_create (Posicion, &taskParams, NULL);


    Clock_Params_init(&clkParams);
    clkParams.period = Posicion_Periodo;
    clkParams.startFlag = TRUE;
    Posicion_Reloj=Clock_create(Posicion_Clock, 5, &clkParams, NULL);

    Init_PeripheralClocks();
    Gpio_select();

    Server_Create();

    BIOS_start();
    return(0);
}

/*
 *  ====================== Tarea de control de la velocidad======================
 */
Void Control(UArg arg0, UArg arg1)
{

    /*======================INICIALIZACIONES=========================*/

    Init_EPWM2 ();
    Init_Encoder_left ();

    Init_EPWM7 ();
    Init_Encoder_right ();

    GpioDataRegs.GPASET.bit.GPIO15 = 1;

    for (;;) {
        Semaphore_pend(Control_Sem, BIOS_WAIT_FOREVER);



/*======================LECTURA DE ENCODERS=========================*/

        encoder_left=Get_Counter_left ();
        direncoder_left = EQep3Regs.QEPSTS.bit.QDF;

        encoder_right=Get_Counter_right ();
        direncoder_right = EQep1Regs.QEPSTS.bit.QDF;

//==============CÁLCULO DEL MODELO CINEMÁTICO (modo coche)====================

        R=v_coche/w_coche;


        v_i=sqrt(((v_coche-(w_coche*(L/2.0)))*((v_coche-(w_coche*(L/2.0))))+((w_coche*S)*(w_co
        che*S))));

        v_d=sqrt(((v_coche+(w_coche*(L/2.0)))*((v_coche+(w_coche*(L/2.0))))+((w_coche*S)*(w_co
        che*S))));
        w_i=v_i/r;
        w_d=v_d/r;

        tita_coche=atan(S/R);

/*==============CONTROL DE VELOCIDAD RUEDA IZQUERDA=================*/


        w_left=cal_velocidad(encoder_left, direncoder_left, &tita0_left, &tita_left);
        accion_left = PI_Dcha(w_i, w_left);
        PWM_Left(accion_left);
```

```
/*==============CONTROL DE VELOCIDAD RUEDA DERECHA=================*/


        w_right=cal_velocidad(encoder_right, direncoder_right, &tita0_right, &tita_right);
        accion_right = PI_Izda(w_d, w_right);
        PWM_Right(accion_right);

    }
}

Void Control_Clock(UArg arg0)
{
    Semaphore_post(Control_Sem);
}


/*
 *  ================ Tarea de control de posición de la dirección ================
 */

//Esta tarea se ejecutaría en otro LaunchPad.
//Es indiferente usar el módulo de lectura de encoders y PWM izquierdo o derecho

Void Posicion(UArg arg0, UArg arg1)
{

    /*======================INICIALIZACIONES=========================*/

    Init_EPWM2 ();
    Init_Encoder_left ();

    for (;;) {
        Semaphore_pend(Posicion_Sem, BIOS_WAIT_FOREVER);



/*======================LECTURA DE ENCODERS=========================*/

        encoder_left=Get_Counter_left ();
        direncoder_left = EQep3Regs.QEPSTS.bit.QDF;

//==============CÁLCULO DEL MODELO CINEMÁTICO (modo coche)====================

        R=v_coche/w_coche;


        v_i=sqrt(((v_coche-(w_coche*(L/2.0)))*((v_coche-(w_coche*(L/2.0))))+((w_coche*S)*(w_co
        che*S))));

        v_d=sqrt(((v_coche+(w_coche*(L/2.0)))*((v_coche+(w_coche*(L/2.0))))+((w_coche*S)*(w_co
        che*S))));
        w_i=v_i/r;
        w_d=v_d/r;

        tita_coche=atan(S/R);


/*==============CONTROL DE POSICION DE LA DIRECCIÓN=================*/

       w_left=cal_velocidad(encoder_left, direncoder_left, &tita0_left, &tita_left);
       accion_left = P(tita_coche, tita0_left);
       left_action(accion_left);

    }
}

Void Posicion_Clock(UArg arg0)
```

```
{
    Semaphore_post(Posicion_Sem);
}


/*
 *  ======== Otras funciones ========
 */


//////////////////////////////////////////////////////////////////////////////////////////////////
//////////
//////   Cálculo de reguladores
//////////////////////////////////////////////////////////////////////////////////////////////////
/////////

float PI_Izda(float Wref, float W)
{
    E_Izda = Wref - W ;
    U_Izda = U_Izda + 0.216*E_Izda - 0.192*E_ant_Izda ;
    E_ant_Izda = E_Izda ;

    return U_Izda ;
}

float PI_Dcha(float Wref, float W)
{
    E_Dcha = Wref - W ;
    U_Dcha = U_Dcha + 0.216*E_Dcha - 0.192*E_ant_Dcha ;
    E_ant_Dcha = E_Dcha ;

    return U_Dcha ;
}


float P(float TitaRef, float Tita)
{
    E_Tita = TitaRef - Tita ;
    U_Tita = 225.625*E_Tita ;

    return U_Tita ;
}

//////////////////////////////////////////////////////////////////////////////////////////////////
//////////
//////   Cálculo de la velocidad real en rad/s
//////////////////////////////////////////////////////////////////////////////////////////////////
/////////
float cal_velocidad(float encoder,Uint_32 direccion, float *tita0, float *tita)
{
    float w, dtita=0;
    *tita0=0.0000732059799*encoder;

    if(direccion==1)
    {
        if(*tita0<*tita)
        {
            dtita=*tita0+dospi-*tita;

        }else
        {
            dtita=*tita0-*tita;
        }
    }
    else
    {

        if(*tita0<=*tita)
```

```
        {
            dtita=*tita0-*tita;
        }else
        {
            dtita=*tita0-(*tita+dospi);
        }
    }

    w = dtita/0.05;
    *tita=*tita0;
    return w;

}


//////////////////////////////////////////////////////////////////////////////////////////////////
//////////
//////   Accion PWM
//////////////////////////////////////////////////////////////////////////////////////////////////
////////
void PWM_Left (float U) {
    unsigned int PWM ;

    if (U > 5.0) U = 5.0 ;
    if (U < 0.0) U = 0.0 ;
    PWM = (819*U) ;  //819 para 5V, 1240.909 para 3,3V
    Set_Value_EPWM2 (PWM) ;
}

void PWM_Right (float U) {
    unsigned int PWM ;

    if (U > 5.0) U = 5.0 ;
    if (U < 0.0) U = 0.0 ;
    PWM = (819*U) ;  //819 para 5V, 1240.909 para 3,3V
    Set_Value_EPWM7 (PWM) ;
}

void Init_PeripheralClocks()
{
    EALLOW;
/*
    CpuSysRegs.PCLKCR0.bit.CLA1 = 0;
    CpuSysRegs.PCLKCR0.bit.DMA = 0;
    CpuSysRegs.PCLKCR0.bit.CPUTIMER0 = 1;
    CpuSysRegs.PCLKCR0.bit.CPUTIMER1 = 0;
    CpuSysRegs.PCLKCR0.bit.CPUTIMER2 = 0;
    CpuSysRegs.PCLKCR0.bit.HRPWM = 0;
  */  CpuSysRegs.PCLKCR0.bit.TBCLKSYNC = 1;

 /*   CpuSysRegs.PCLKCR1.bit.EMIF1 = 0;
    CpuSysRegs.PCLKCR1.bit.EMIF2 = 0;
*/
    CpuSysRegs.PCLKCR2.bit.EPWM1 = 0;
    CpuSysRegs.PCLKCR2.bit.EPWM2 = 1;
    CpuSysRegs.PCLKCR2.bit.EPWM3 = 0;
    CpuSysRegs.PCLKCR2.bit.EPWM4 = 0;
    CpuSysRegs.PCLKCR2.bit.EPWM5 = 0;
    CpuSysRegs.PCLKCR2.bit.EPWM6 = 0;
    CpuSysRegs.PCLKCR2.bit.EPWM7 = 1;
    CpuSysRegs.PCLKCR2.bit.EPWM8 = 0;
    CpuSysRegs.PCLKCR2.bit.EPWM9 = 0;
    CpuSysRegs.PCLKCR2.bit.EPWM10 = 0;
    CpuSysRegs.PCLKCR2.bit.EPWM11 = 0;
    CpuSysRegs.PCLKCR2.bit.EPWM12 = 0;
/*
    CpuSysRegs.PCLKCR3.bit.ECAP1 = 0;
    CpuSysRegs.PCLKCR3.bit.ECAP2 = 0;
```

```
    CpuSysRegs.PCLKCR3.bit.ECAP3 = 0;
    CpuSysRegs.PCLKCR3.bit.ECAP4 = 0;
    CpuSysRegs.PCLKCR3.bit.ECAP5 = 0;
    CpuSysRegs.PCLKCR3.bit.ECAP6 = 0;
*/
    CpuSysRegs.PCLKCR4.bit.EQEP1 = 1;
    CpuSysRegs.PCLKCR4.bit.EQEP2 = 0;
    CpuSysRegs.PCLKCR4.bit.EQEP3 = 1;
/*
    CpuSysRegs.PCLKCR6.bit.SD1 = 0;
    CpuSysRegs.PCLKCR6.bit.SD2 = 0;

    CpuSysRegs.PCLKCR7.bit.SCI_A = 0;
    CpuSysRegs.PCLKCR7.bit.SCI_B = 0;
    CpuSysRegs.PCLKCR7.bit.SCI_C = 0;
    CpuSysRegs.PCLKCR7.bit.SCI_D = 0;

    CpuSysRegs.PCLKCR8.bit.SPI_A = 0;
    CpuSysRegs.PCLKCR8.bit.SPI_B = 0;
    CpuSysRegs.PCLKCR8.bit.SPI_C = 0;

    CpuSysRegs.PCLKCR9.bit.I2C_A = 0;
    CpuSysRegs.PCLKCR9.bit.I2C_B = 0;

    CpuSysRegs.PCLKCR10.bit.CAN_A = 0;
    CpuSysRegs.PCLKCR10.bit.CAN_B = 0;

    CpuSysRegs.PCLKCR11.bit.McBSP_A = 0;
    CpuSysRegs.PCLKCR11.bit.McBSP_B = 0;
    CpuSysRegs.PCLKCR11.bit.USB_A = 0;

    CpuSysRegs.PCLKCR12.bit.uPP_A = 0;

    CpuSysRegs.PCLKCR13.bit.ADC_A = 1;
    CpuSysRegs.PCLKCR13.bit.ADC_B = 0;
    CpuSysRegs.PCLKCR13.bit.ADC_C = 0;
    CpuSysRegs.PCLKCR13.bit.ADC_D = 0;

    CpuSysRegs.PCLKCR14.bit.CMPSS1 = 0;
    CpuSysRegs.PCLKCR14.bit.CMPSS2 = 0;
    CpuSysRegs.PCLKCR14.bit.CMPSS3 = 0;
    CpuSysRegs.PCLKCR14.bit.CMPSS4 = 0;
    CpuSysRegs.PCLKCR14.bit.CMPSS5 = 0;
    CpuSysRegs.PCLKCR14.bit.CMPSS6 = 0;
    CpuSysRegs.PCLKCR14.bit.CMPSS7 = 0;
    CpuSysRegs.PCLKCR14.bit.CMPSS8 = 0;

    CpuSysRegs.PCLKCR16.bit.DAC_A = 0;
    CpuSysRegs.PCLKCR16.bit.DAC_B = 0;
    CpuSysRegs.PCLKCR16.bit.DAC_C = 0;
  */
    EDIS;
}

void Gpio_select(void) {
    EALLOW;

    GpioCtrlRegs.GPCDIR.bit.GPIO78 |= 1 ;
/*
    GpioCtrlRegs.GPAMUX1.bit.GPIO15 = 0;
    GpioCtrlRegs.GPADIR.bit.GPIO15 = 1;     // 1=OUTput,  0=INput
    GpioDataRegs.GPACLEAR.bit.GPIO15 = 1;   // Inicialmente low
*/

    GpioCtrlRegs.GPBMUX2.bit.GPIO58 = 0;
    GpioCtrlRegs.GPBDIR.bit.GPIO58 = 1;     // 1=OUTput,  0=INput
    GpioDataRegs.GPBCLEAR.bit.GPIO58 = 1;   // Inicialmente low
```

```
    EDIS;

    return ;
}
```

```
/**********************************************************************************/
/*                          Used modules                                          */
/**********************************************************************************/

#include "F2837xS_Device.h"
#include "encoder.h"


void Init_Encoder_left (void) {

   // EQEP3 = QEP_B

    EALLOW;

    GpioCtrlRegs.GPBPUD.bit.GPIO62 = 1;    // Disable pull-up on GPIO62 (EQEP3A)
    GpioCtrlRegs.GPBPUD.bit.GPIO63 = 1;    // Disable pull-up on GPIO63 (EQEP3B)

    GpioCtrlRegs.GPBQSEL2.bit.GPIO62 = 0;   // Sync GPIO62 to SYSCLK  (EQEP3A)
    GpioCtrlRegs.GPBQSEL2.bit.GPIO63 = 0;   // Sync GPIO63 to SYSCLK  (EQEP3B)

    GpioCtrlRegs.GPBGMUX2.bit.GPIO62 = 1;   // Configure GPIO62 as EQEP3A
    GpioCtrlRegs.GPBMUX2.bit.GPIO62 = 1;    // Configure GPIO62 as EQEP3A
    GpioCtrlRegs.GPBGMUX2.bit.GPIO63 = 1;   // Configure GPIO63 as EQEP3B
    GpioCtrlRegs.GPBMUX2.bit.GPIO63 = 1;    // Configure GPIO63 as EQEP3B

    EDIS;

    EQep3Regs.QDECCTL.bit.QSRC= 0 ;     // QEP quadrature count mode
    EQep3Regs.QDECCTL.bit.XCR= 0 ;      // 2x resolution: Count the rising/falling edge
    EQep3Regs.QPOSMAX = 85800;


    EQep3Regs.QEPCTL.bit.FREE_SOFT=2;   // Position counter is unaffected by emulation suspend
    EQep3Regs.QEPCTL.bit.PCRM=00;       // PCRM=00 mode - QPOSCNT reset on index event
    EQep3Regs.QEPCTL.bit.UTE=0;         // Unit Timeout Disable
    //EQep3Regs.QPOSMAX=0xffffffff;
    EQep3Regs.QEPCTL.bit.QPEN=1;        // QEP enable

}

void Init_Encoder_right (void) {

   // EQEP1 = QEP_A

    EALLOW;

    GpioCtrlRegs.GPAPUD.bit.GPIO10 = 1;    // Disable pull-up on GPIO10 (EQEP1A)
    GpioCtrlRegs.GPAPUD.bit.GPIO11 = 1;    // Disable pull-up on GPIO11 (EQEP1B)

    GpioCtrlRegs.GPAQSEL1.bit.GPIO10 = 0;   // Sync GPIO10 to SYSCLK  (EQEP1A)
    GpioCtrlRegs.GPAQSEL1.bit.GPIO11 = 0;   // Sync GPIO11 to SYSCLK  (EQEP1B)

    GpioCtrlRegs.GPAGMUX1.bit.GPIO10 = 1;   // Configure GPIO10 as EQEP1A
    GpioCtrlRegs.GPAMUX1.bit.GPIO10 = 1;    // Configure GPIO10 as EQEP1A
    GpioCtrlRegs.GPAGMUX1.bit.GPIO11 = 1;   // Configure GPIO11 as EQEP1B
    GpioCtrlRegs.GPAMUX1.bit.GPIO11 = 1;    // Configure GPIO11 as EQEP1B

    EDIS;

    EQep1Regs.QDECCTL.bit.QSRC= 0 ;     // QEP quadrature count mode
    EQep1Regs.QDECCTL.bit.XCR= 0 ;      // 2x resolution: Count the rising/falling edge
    EQep1Regs.QPOSMAX = 85800;

    EQep1Regs.QEPCTL.bit.FREE_SOFT=2;   // Position counter is unaffected by emulation suspend
    EQep1Regs.QEPCTL.bit.PCRM=00;       // PCRM=00 mode - QPOSCNT reset on index event
    EQep1Regs.QEPCTL.bit.UTE=0;         // Unit Timeout Disable
   // EQep1Regs.QPOSMAX=0xffffffff;
    EQep1Regs.QEPCTL.bit.QPEN=1;        // QEP enable
```

```
}

Uint_32 Get_Counter_left (void) {
   return EQep3Regs.QPOSCNT ;                // EQEP3 = QEP_B
}

Uint_32 Get_Counter_right (void) {
   return EQep1Regs.QPOSCNT ;                // EQEP1 = QEP_A
}
```

```
#include "F2837xS_Device.h"
#include "pwm.h"



void Init_EPWM2 (void) {

    EALLOW;
    GpioCtrlRegs.GPAPUD.bit.GPIO2 = 1;    // Disable pull-up on GPIO2 (EPWM2A)
    GpioCtrlRegs.GPAPUD.bit.GPIO3 = 1;    // Disable pull-up on GPIO3 (EPWM2B)

    GpioCtrlRegs.GPAMUX1.bit.GPIO2 = 1;   // Configure GPIO2 as EPWM2A
    GpioCtrlRegs.GPAMUX1.bit.GPIO3 = 1;   // Configure GPIO3 as EPWM2B
    EDIS;


    /* Configure EPWM2
       Assumes EPWM2 Clock is already enabled in Init_SysCtrl(); */

       // Setup TBCLK
       EPwm2Regs.TBCTL.bit.CTRMODE = 0 ;          // Up-count mode
       EPwm2Regs.TBPRD = 0x0FFF ;                 // 12 bits
       EPwm2Regs.TBCTL.bit.PHSEN = 0;             // Disable phase loading
       EPwm2Regs.TBPHS.bit.TBPHS = 0x0000;        // Phase is 0
       EPwm2Regs.TBCTR = 0x0000;                  // Clear counter
       EPwm2Regs.TBCTL.bit.HSPCLKDIV = 1 ;        // Divide by 2
       EPwm2Regs.TBCTL.bit.CLKDIV = 1 ;           // Divide by 2

       // Setup shadow register load on ZERO
       EPwm2Regs.CMPCTL.bit.SHDWAMODE = 0 ;       // Shadow mode, buffered pwm
       EPwm2Regs.CMPCTL.bit.SHDWBMODE = 0 ;       // Shadow mode, buffered pwm
       EPwm2Regs.CMPCTL.bit.LOADAMODE = 0 ;       // Load on counter = 0
       EPwm2Regs.CMPCTL.bit.LOADBMODE = 0 ;       // Load on counter = 0

       // Set Compare values
       EPwm2Regs.CMPA.bit.CMPA = 0x000F ;        // Set compare A value

       // Set actions
       EPwm2Regs.AQCTLA.bit.ZRO = 0 ;            // Do nothing
       EPwm2Regs.AQCTLA.bit.PRD = 2 ;            // Set: force EPWMxA output high
       EPwm2Regs.AQCTLA.bit.CAU = 3 ;            // Toggle EPWMxA output
       EPwm2Regs.AQCTLA.bit.CAD = 0 ;            // Do nothing

       EPwm2Regs.AQCTLB.bit.ZRO = 0 ;            // Do nothing
       EPwm2Regs.AQCTLB.bit.PRD = 1 ;            // Clear: force EPWMxB output low
       EPwm2Regs.AQCTLB.bit.CAU = 3 ;            // Toggle EPWMxB output
       EPwm2Regs.AQCTLB.bit.CAD = 0 ;            // Do nothing

}

void Init_EPWM7 (void) {

    EALLOW;
    GpioCtrlRegs.GPAPUD.bit.GPIO12 = 1;    // Disable pull-up on GPIO12 (EPWM7A)
    GpioCtrlRegs.GPAPUD.bit.GPIO13 = 1;    // Disable pull-up on GPIO13 (EPWM7B)

    GpioCtrlRegs.GPAMUX1.bit.GPIO12 = 1;   // Configure GPIO12 as EPWM7A
    GpioCtrlRegs.GPAMUX1.bit.GPIO13 = 1;   // Configure GPIO13 as EPWM7B
    EDIS;


    /* Configure EPWM7
       Assumes EPWM7 Clock is already enabled in Init_SysCtrl(); */

       // Setup TBCLK
       EPwm7Regs.TBCTL.bit.CTRMODE = 0 ;          // Up-count mode
       EPwm7Regs.TBPRD = 0x0FFF ;                 // 12 bits
```

```
        EPwm7Regs.TBCTL.bit.PHSEN = 0;              // Disable phase loading
        EPwm7Regs.TBPHS.bit.TBPHS = 0x0000;         // Phase is 0
        EPwm7Regs.TBCTR = 0x0000;                   // Clear counter
        EPwm7Regs.TBCTL.bit.HSPCLKDIV = 1 ;         // Divide by 2
        EPwm7Regs.TBCTL.bit.CLKDIV = 1 ;            // Divide by 2

        // Setup shadow register load on ZERO
        EPwm7Regs.CMPCTL.bit.SHDWAMODE = 0 ;        // Shadow mode, buffered pwm
        EPwm7Regs.CMPCTL.bit.SHDWBMODE = 0 ;        // Shadow mode, buffered pwm
        EPwm7Regs.CMPCTL.bit.LOADAMODE = 0 ;        // Load on counter = 0
        EPwm7Regs.CMPCTL.bit.LOADBMODE = 0 ;        // Load on counter = 0

        // Set Compare values
        EPwm7Regs.CMPA.bit.CMPA = 0x000F ;        // Set compare A value

        // Set actions
        EPwm7Regs.AQCTLA.bit.ZRO = 0 ;            // Do nothing
        EPwm7Regs.AQCTLA.bit.PRD = 2 ;            // Set: force EPWMxA output high
        EPwm7Regs.AQCTLA.bit.CAU = 3 ;            // Toggle EPWMxA output
        EPwm7Regs.AQCTLA.bit.CAD = 0 ;            // Do nothing

        EPwm7Regs.AQCTLB.bit.ZRO = 0 ;            // Do nothing
        EPwm7Regs.AQCTLB.bit.PRD = 1 ;            // Clear: force EPWMxB output low
        EPwm7Regs.AQCTLB.bit.CAU = 3 ;            // Toggle EPWMxB output
        EPwm7Regs.AQCTLB.bit.CAD = 0 ;            // Do nothing

}


void Set_Value_EPWM2 (unsigned int V) {
        EPwm2Regs.CMPA.bit.CMPA = 0x0FFF&V ;        // Set compare A value
        EPwm2Regs.CMPB.bit.CMPB = 0x0FFF&V ;        // Set compare B value
}

void Set_Value_EPWM7 (unsigned int V) {
        EPwm7Regs.CMPA.bit.CMPA = 0x0FFF&V ;        // Set compare A value
}
```