Proyecto Fin de Grado
Grado en Ingeniería de las Tecnologías Industriales

Desarrollo de un robot autoequilibrado basado en Arduino con motores paso a paso

Autor: José Antonio Borja Conde
Tutor: Ignacio Alvarado Aldea

Dep. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2018

Proyecto Fin de Grado
Grado en Ingeniería de las Tecnologías Industriales

Desarrollo de un robot autoequilibrado basado en Arduino con motores paso a paso

Autor:
José Antonio Borja Conde

Tutor:
Ignacio Alvarado Aldea
Profesor titular

Dep. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2018

Proyecto Fin de Grado: Desarrollo de un robot autoequilibrado basado en Arduino con motores paso a paso

Autor: José Antonio Borja Conde

Tutor: Ignacio Alvarado Aldea

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

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2018

El Secretario del Tribunal

A mis padres

A Paula

A mis hermanos

Agradecimientos

Este proyecto de fin de grado va dedicado a aquellas personas que me han ayudado y facilitado llegar hasta aquí, ya sea de forma directa o indirecta.

A mi madre, que sin importar las circunstancias siempre ha invertido su tiempo y esfuerzo para permitirme tener más horas de estudio o descanso. Día tras día, desde que en tercero de primaria tuve mi primer examen, con aquellos numerosos fines de semana dedicados completamente a ayudarme a estudiar. No hay suficientes palabras para describir tanto apoyo.

A mi padre, que, aunque su trabajo le impidiera ayudarme directamente, sin él no podría haber tenido los suficientes recursos y facilidades para conseguir este logro. Su alegría y forma de ser son la llama de nuestra familia, que nunca impide que un problema nos supere.

A Paula, que ha sido el único motivo por el que merecía la pena esforzarse. Ha sido quien me ha regalado la felicidad necesaria para poder continuar. Hace que un día de trabajo se convierta en un día perfecto por estar con ella cuando acabo. Que ir a estudiar juntos un fin de semana sea como ir de vacaciones por tener su compañía. Nunca podré devolverle todo lo que me ha dado.

A mis hermanos, que siempre me han ayudado en todo tipo de problemas que me han surgido, y siempre me han demostrado que les importo más que ellos mismo. Han hecho que crecer sea algo mágico.

A Ignacio, que me ha dado la oportunidad de realizar un proyecto con el que realmente he disfrutado. Por interesarse y esforzarse tanto en que los alumnos aprendan de una forma amena, que aumenta tanto la motivación que trabajar se convierte en diversión. Gracias a él me llevo el mejor sabor de boca posible de este grado.

Simplemente, gracias.

José Antonio Borja Conde

Sevilla, 2018

Resumen

Este proyecto tiene la principal finalidad de ser usado como equipo de prácticas para la docencia de nuestra Escuela. Inspirándose en los sistemas de aprendizajes basados en problemas (*Project-based learning*, *PBL*), pretende crear un sistema para que los alumnos, que poseen ciertos conocimientos previos, entren en contacto directo con problemas reales, que proporcionan un punto de partida en el proceso de aprendizaje y aumenta la motivación en el aprendizaje.

Para hacer viable esta idea, se ha disminuido todo lo posible el precio de los diferentes componentes y se ha usado una estructura obtenida con una impresora *3D* que permite la fabricación en serie.

Abstract

This Project have the main purpose of being used as a practice equipment for the teaching of our School. Inspired by Project-Based learning (*PBL*), it expects to create a system for students, who possess certain prior knowledge, come in contact with real problems, which provides a starting point in the learning process and increases motivation in learning.

To make this idea viable, the prico of the different components has been reduced as much as possible and a structure obtained with a *3D* printer that allows mass production has been used.

Índice

ÍNDICE DE TABLAS

INDICE DE FIGURAS

1 OBJETO DEL PROYECTO

Un péndulo tiene dos posiciones de equilibrio. Ambas se producen cuando la línea que une el punto de soporte y el centro de gravedad se encuentra en posición vertical, es decir, tiene la misma dirección que la de la gravedad. Cuando el centro de gravedad se encuentra por debajo del soporte, el equilibrio es estable: si se produce alguna perturbación, la física del sistema hace que se retorne a la posición de equilibro. En cambio, si el centro de gravedad se encuentra por encima del soporte, el equilibrio es inestable: cualquier perturbación hace que el sistema tienda a perder dicha posición.

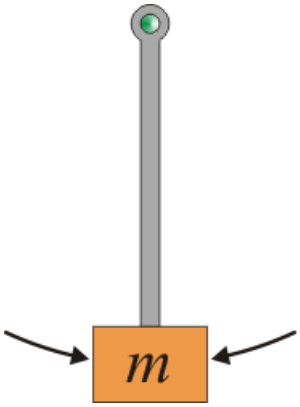

Figura 1-1. Péndulo en equilibrio estable

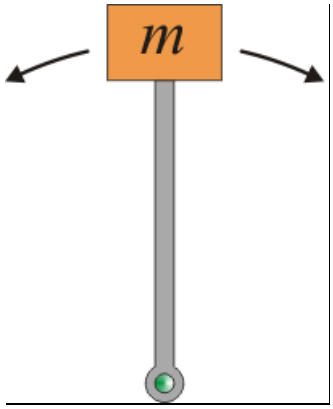

Figura 1-2. Péndulo en equilibrio inestable

Este proyecto se centra en el segundo caso. Por tanto, el objetivo es la creación y control de un de un vehículo tipo péndulo invertido autoequilibrado. Según la experiencia práctica, se sabe que siempre habrá alguna perturbación o ruido que desestabilice el sistema. El sistema está formado por dos motores paso a paso acoplados a dos ruedas que producen el movimiento (véase la siguiente figura). Este movimiento estará condicionado a órdenes de traslación y giro que reciba a través de un módulo *Bluetooth*, pero principalmente lo estará a la inclinación del vehículo, que se obtiene gracias a una unidad de medidas inerciales (*IMU*), para equilibrar el vehículo. De forma que la aceleración de los motores aumente o disminuya acercando la posición del vehículo al equilibrio inestable por el recorrido más corto. La siguiente figura muestra el sistema con las variables más relevantes para el control:

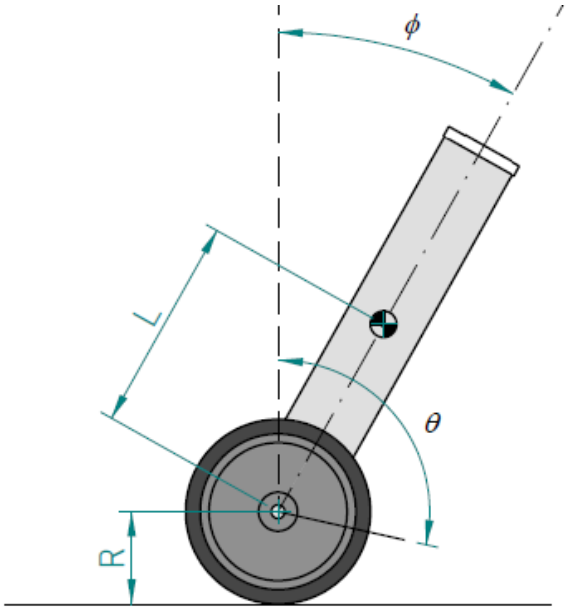

Figura 1-3. Diagrama del sistema con algunas variables

Para el cálculo del controlador se necesita el módelo físico-matemático del vehículo, el cual ha sido proporcionado, ya que se había desarrollado antes del inicio de este proyecto (aunque se le han incluido ciertas mejoras, como se puede ver en el capítulo 5). La implementación de este modelo en el entorno de trabajo *Simulink* permite realizar también simulaciones para realizar las pruebas necesarias a partir de los parámetros calculados. Éstos son:

Tabla 1-1. Parámetros para el modelo físico-mátematico del vehículo

Parámetro	**Notación**	**Valor**
Masa de una rueda	m_r	*0.044 kg*
Radio de una rueda	R	*0.053 m*
Momento de inercia de una rueda	J_r	$6.18 \cdot 10^{-5}$ *kg·m²*
Masa del vehículo sin ruedas	M	*0.745 kg*
Distancia entre el eje de ruedas y el centro de gravedad	L	*0.0805 m*
Momento de inercia del vehículo sin las ruedas	J_ϕ	$4.83 \cdot 10^{-3}$ *kg·m²*
Aceleración gravitacional	g	*9.81 m/s²*

Del desarrollo del modelo que se realiza en el capítulo "5. Modelo Físico-Matemático" (haciendo uso de las ecuaciones de *Lagrange*), se obtienen los siguientes resultados:

$$(2a + c \cdot \cos\phi) \cdot \ddot{\theta} + (c \cdot \cos\phi + 2b) \cdot \ddot{\phi} - c \cdot \dot{\phi}^2 \cdot \sin\phi - d \cdot \sin\phi = 0$$

$$a = \left(m_r + \frac{1}{2}M\right) \cdot R^2 + J_r$$
$$b = \frac{1}{2}(M \cdot L^2 + J_\phi)$$
$$c = R \cdot L \cdot M$$
$$d = -M \cdot g \cdot L$$
$$J_r = \frac{1}{2}m_r \cdot R^2$$
$$J_\phi = M \cdot L^2$$

Los valores de estos parámetros en este proyecto son:

a	0.0012317465 $kg{\cdot}m^2$
b	0.00482778625 $kg{\cdot}m^2$
c	0.0031785425 $kg{\cdot}m^2$
d	-0.588330225 $kg{\cdot}m^2/s^2$

Cuando se haya conseguido el anterior objetivo, se pretende diseñar componentes que faciliten la construcción del vehículo partiendo de los diferentes dispositivos electrónicos y la estructura impresa en *3D*. De forma que se puedan crear en serie equipos de prácticas, para asignaturas realiacionadas con el control y la electrónica.

Además, sería interesante utilizar este sistema para la implementación de distintas estrategias de control y sincronización entre varios vehículos.

2 PUNTO DE PARTIDA

Este proyecto surge a partir de otros proyectos realizados por Antonio Croche Navaroo y Cecilia González González, los cuales se citan en las referencias. Aunque tiene grandes diferencias tanto en software como en hardware, puede ser considerado como una posible mejora.

Ambos usaron el mismo equipo, que se puede ver en la siguiente figura:

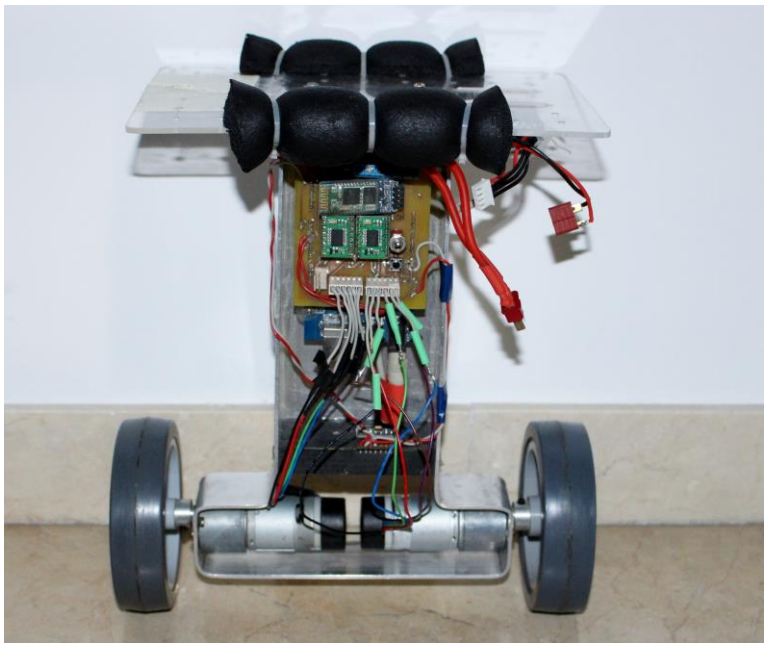

Figura 2-1. Imagen del vehículo del anterior proyecto

Los diferentes componentes con los que cuenta son descritos en "*Anexo 1: Componentes Proyecto Previo"*.

El primer aspecto a tener en cuenta es que la respuesta del controlador del vehículo para mantenerse estable es la aceleración de las ruedas. Por tanto, es necesario tener control sobre esta.

El principal problema que tenía el anterior vehículo era el mecanismo para mover las ruedas. Se usaban motores de corriente continua con reductora con realimentación en velocidad obtenidad a partir de la medida de dos encóders en cuadratura. Y esto tenía claras desventajas:

1. Holguras debido a la caja reductura: Los engranajes de la reductora no pueden tener un acoplamiento perfecto, lo que deriva en pequeños movimientos indeseados.
2. Zona muerta de los motores DC: Los motores no se mueven con voltajes inferiores a un nivel mínimo. Esto hace que haya un rango de voltaje, en torno a 0V, que no afecta al movimiento del motor. A este efecto se le denomina zona muerta.
3. Ruido que afecta a la medida de velocidad y aceleración de las ruedas: La obtención de la señal de encóder, discreta, produce señales de velocidad y aceleración muy ruidosas.

Por tanto, en este nuevo proyecto se han sustituido dichos motores por motores paso a paso que solucionan todos los problemas anteriores:

Al tener alta precisión (1600 paso por vuelta) y suficiente par, no es necesario disponer de reductora, por lo que no se tienen holguras.

Por el mismo funcionamiento de los motores (un pulso produce un paso), no hay zona muerta, pues siempre se

alimenta al mismo voltaje.

La velocidad y aceleración es conocida, ya que depende del tren de pulsos que se le envíe. Por lo que no se necesita encóder y se conoce con alta proximidad ambas componentes del movimiento de los motores.

Sin embargo, el uso de estos motores deriva en nuevos problemas de software:

En el proyecto anterior, para poder implementar aceleración se utilizaban dos controladores PID con los que se calculaba la aceleración, y se aplicaba el voltaje necesario al motor para conseguir la aceleración necesaria. Esto cambia drásticamente ahora. Ambos controladores *PID* son sustituidos por **dos algoritmos, uno que implemente velocidad a partir de un tren de pulsos, y otro que vaya modificando esta velocidad de forma que se aplique una aceleración deseada.**

Los materiales, por tanto, son prácticamente los mismo, sustituyendo los motores y sus respectivos drivers. También se usará un microprocesador con menos prestaciones, ya que los dos encóders necesitaban cuatro interrupciones externas y ahora no se necesita ninguna. Por tanto, se sustituye el *Arduino MEGA* (5 interrupciones externas disponibles) por el *Arduino UNO* (2 interrupciones). Estos se pueden ver en los siguientes apartados.

3 CAMBIOS DE HARDWARE

En esta sección se presentan los componentes electrónicos, así como la estructura y complementos en los que estos se soportan, que se comparten y no con el proyecto anteriormente citado. Sus principales características serán descritas. Aunque, en secciones posteriores, se entrará más en detalle, al explicar cómo se utilizan para obtener el funcionamento necesario.

El esquema final general se puede ver en la siguiente figura:

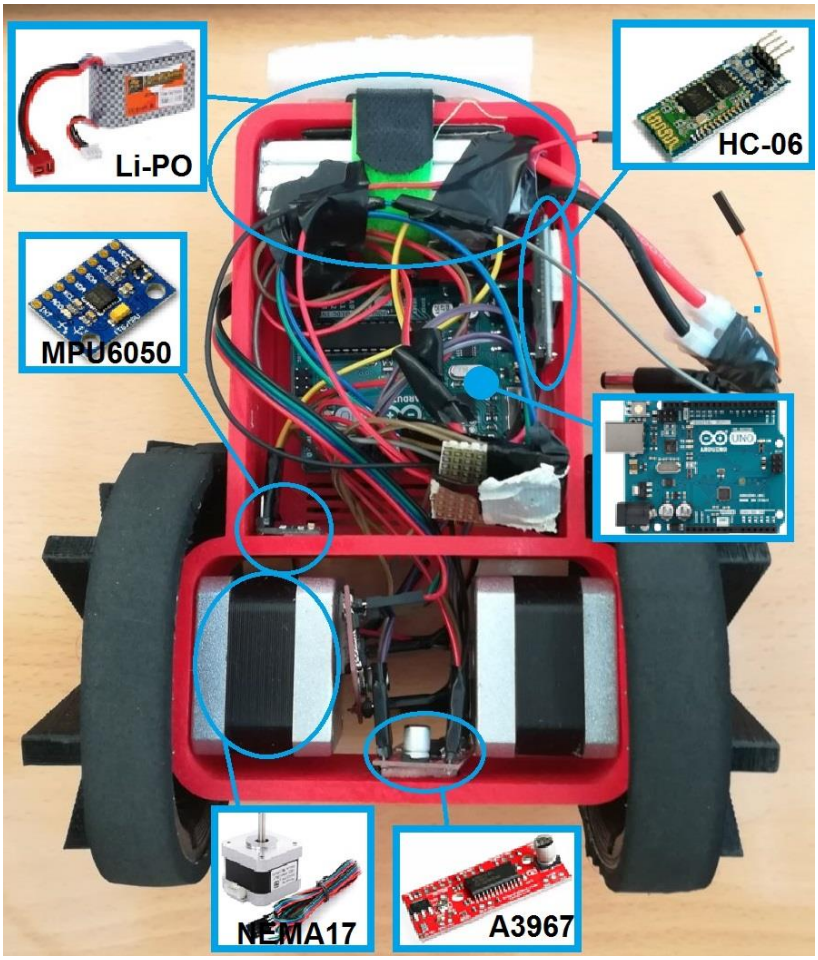

Figura 3-1. Esquema general del vehículo con sus principales componentes

3.1 Estructura impresa en 3D

Este componente sustituye al chases de aluminio y madera del anterior Proyecto.

El diseño de este modelo fue creado por Daniel Aparicio, quien lo puso a disposición del Departamento de Ingeniería de Sistemas y Automática con el fin de que se trabajase con él. El formato que se utilizó fue "STL". Consta de cuatro tipos: base, tapa, y llanta y portagomas de la rueda. De forma que, tras obtenerse a partir de una impresora 3D, se monten y se tenga la estructura en la que se pueden soportar todos los componentes electrónicos, ya que dispone de los encajes necesarios. Los diseños los podemos ver en las siguientes figuras:

Figura 3-2. Modelo 3D de la base para vehículo

Figura 3-3. Modelo 3D de la tapa para vehículo

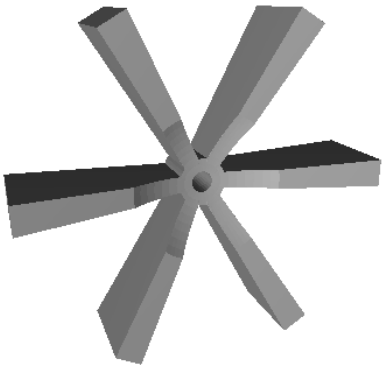

Figura 3-4. Modelo 3D de la llanta para rueda de vehículo

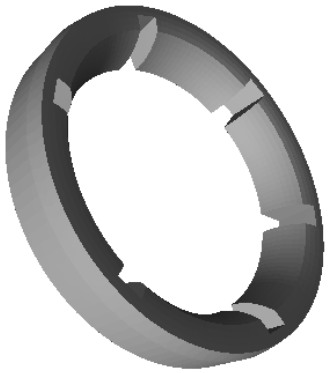

Figura 3-5. Modelo 3D del portagomas para rueda de vehículo

Como se puede ver, el portagomas y la llanta forman la rueda, junto a un material de alta fricción que se coloque como cubierta. Los motores serán atornillados en la parte inferior de la base, y sus ejes irán encajados en el hueco central de la llanta.

La tapa servirá de soporte para el *Arduino* y permitirá esconder el cableado.

El agujero central de la base permite conectar los motores con el microcontrolador, y la ranura lateral da la posibilidad de cargar programas en dicha placa sin necesidad de desmontar el vehículo.

La IMU debe ir lo más cerca posible del eje de giro para que las aceleraciones por el movimiento del sistema no interfieran demasiado en sus medidas, por lo que se coloca junto al agujero mencionado en el apartado

anterior.

Las características de impresión 3D usadas para este proyecto han sido:

- Material: ácido poliláctico (*PLA*)
- Altura de capa: 0.25
- Relleno: 25%

3.2 Motores Nema 17 17HD34008-22B

El uso de estos motores es la principal diferencia con respecto al proyecto anterior. En dicho proyecto se usaron motores DC. El tipo de motor con el que se trabaja ahora es paso a paso bipolar. Estos se caracterizan por disponer de dos bobinas que necesitan cambios en la dirección de las intensidades que circulan por ellas para poder generar pasos. Se tienen dos terminales para cada bobina. Energizándolas a través de estos, de forma ordenada, se consigue que el motor genere un paso o micropaso (que será la media, cuarta u octava parte de un paso completo). Por tanto, se necesita producir una secuencia repetitiva de energización de las bobinas. Para generarla será necesario incluir un *microstepping motor driver*, descrito en el siguiente apartado.

Figura 3-6. Motor Nema 17 17HD34008-22B

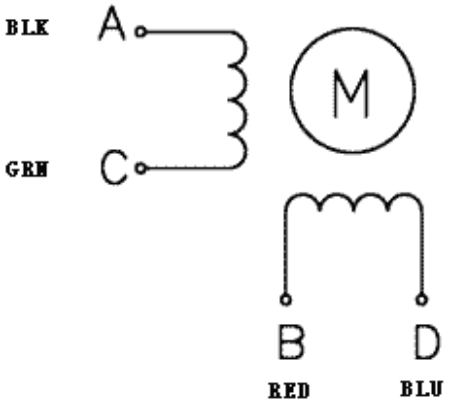

Figura 3-7. Esquema interno de un motor bipolar

Estos motores se caracterizan por tener alta precisión y potencia. Sus especificaciones podemos encontrarlas en la siguiente tabla:

Tabla 3-1. Características principales de los motores

Características	Medidas y unidades
Corriente nominal	1,2 A DC
Tensión de alimentación	12-36 V
Ángulo de avance	1.8º ± 5%
Par de fricción	12mN.m
Par de mantenimiento	≥300mN.m (I = 1.5 A)
Máx. Frecuencia de arranque sin carga	≥1500pps

Máx. Frecuencia de ejecución sin carga	≥8000pps
Peso del motor	0.23Kg
Tamaño (L * W * H)	42 x 42 x 34 mm

3.3 *Microstepping motor driver* A3967

Este dispositivo sería equivalente a la tarjeta controladora de motores del anterior proyecto.

El *A3967* es un *driver* para motores paso a paso bipolares. Se puede usar para modos de medio, cuarto y octavo paso, y para paso completo. Pudiéndose alternar entre estos en tiempo de ejecución. Para ello consta de un repetidor[1] que permite generar las secuencias necesarias en los devanados de las bobinas del motor, y además suministrar voltajes distintos al de sus entradas digitales (siempre y cuando se le esté proporcionando por los pines correspondientes el voltaje deseado).

Gracias al repetidor, el generar un paso/secuencia de pasos (o micropasos), se resume a enviar un pulso/tren de pulsos por el pin *STEP*.

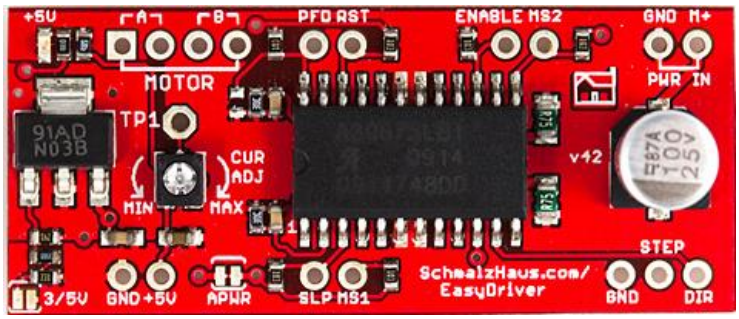

Figura 3-8. *Microstepping motor driver* A3967

3.3.1 Pines de entrada y salida

En este apartado, por simplicidad, solo se describen los pines que se utilizan en el proyecto. Para más información se puede recurrir al *datasheet*, incluido en la bibliografía.

- Entrada para reinicio (***RESET***): Cuando se active (a nivel bajo) el repetidor toma las condiciones iniciales y apaga todas las salidas. Las señales en el pin *STEP* son ignoradas hasta que se desactive *RESET*.
- Entrada de pasos (***STEP***): Un flanco de subida provoca un incremento unitario de paso (o micropaso). El movimiento se realizará en el sentido indicado por el pin *DIR*.
- Entradas de selección de micropasos (***MS1*** y ***MS2***): Permite seleccionar modo de medio, cuarto u octavo de paso, o paso completo, según la combinación detallada en la siguiente tabla dondo *L* y *H* denotan nivel bajo y alto, respectivamente:

[1] Según Wikipedia, una definición de "repetidor" es: "Un dispositivo digital que amplifica, conforma, retemporiza o lleva a cabo una combinación de cualquiera de estas funciones sobre una señal digital de entrada para su retransmisión".

Tabla 3-2.Tabla de verdad de resolución de micropasos

MS1	MS2	Resolución
L	L	Paso completo
H	L	Medio de paso
L	H	Cuarto de paso
H	H	Octavo de paso

- Entrada para determinar el sentido de giro (***DIR***): Si se encuentra a nivel alto, el movimiento del motor será positivo, de lo contrario será negativo.
- Salidas para los devanados de las bobinas del motor (***A+*** y ***A-***, ***B+*** y ***B-***): A diferencia de las entradas anteriores, son salidas analógicas. Entregan los voltajes correspondientes a cada bobina para energizarlas según la secuencia. Conocer los valores que toman no es relevante para la realización de este proyecto.
- Pines de alimentación (***GND*** y ***V+***): Reciben la alimentación tanto para alimentar tanto a los motores como a la propia placa.

3.3.2 Especificaciones principales

El rango de voltaje de salida para alimentar a los motores es de 4.75 a 30V, valores entre los que está comprendido el voltaje nominal (12V) de los escogidos.

El rango del voltaje de entradas lógicas para nivel bajo es de 0 a 1.155V, y el de nivel alto 1.98 a 5.5 V, por lo que es adecuado para las salidas del microcontrolador (*Arduino UNO R3*), que se encuentran a 0V a nivel bajo y a 5V a nivel alto, aproximadamente.

3.3.3 Restricciones temporales

La detección del cambio de estado en los pines de entrada tiene está condicionada a un mínimo de tiempo. Esto limita la frecuencia de cambio de cada pin. En la siguiente figura, obtenida del *datasheet*, se puede ver un esquema explicativo, el cual se complenta con la siguiente tabla.

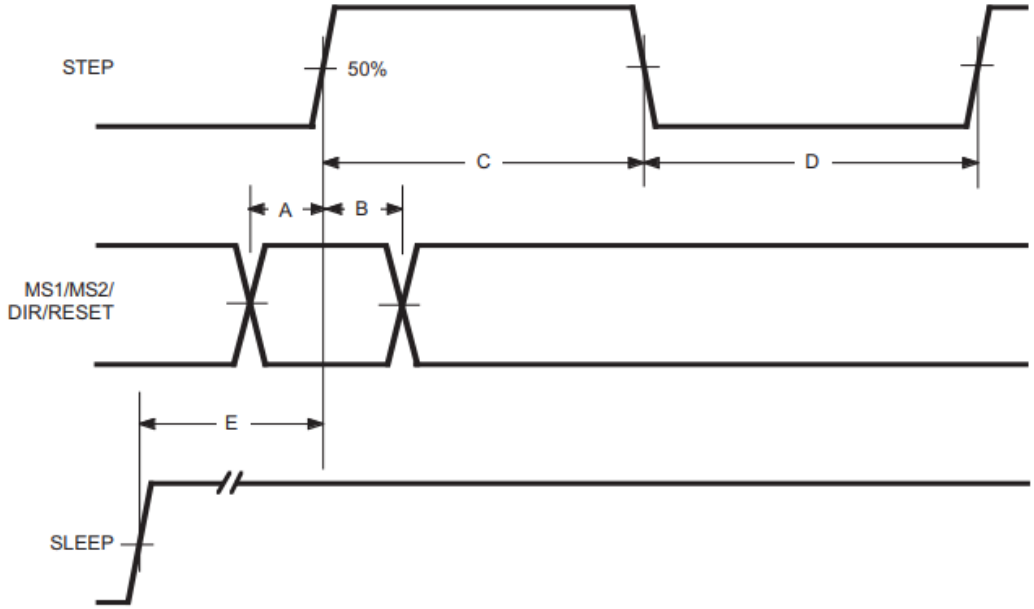

Figura 3-9. Restricciones temporales de *A3967*

Tabla 3-3. Restricciones temporales de *A3967*

Etiqueta	Descripción	Valor
A	Tiempo de configuración de datos	200 ns
B	Tiempo de retención de datos	200 ns
C	Mínimo ancho de pulso *STEP* (alto nivel)	1.0 µs
D	Mínimo ancho de pulso *STEP* (bajo nivel)	1.0 µs
E	Tiempo máximo de activación	1.0 ms

Debido a los mínimos ancho de pulso de *STEP*, la frecuencia máxima de pasos queda limitada a 500 kHz.

3.4 Batería *Li-Po*

Para la alimentación completa del sistema se ha usado una batería Li-Po 3S (11.1 V) con una capacidad de 1500 mAh.

Figura 3-10. Batería *Li-Po* 3S 1500 mAh

Esta batería consta de 3 celdas. Cada celda tiene un voltaje nominal de 3.7 V, pero cuando están cargadas al máximo proporcionan 4.2 V, por lo que se tiene 12.6 V en total.

Los motores elegidos necesitan ser alimentados a 12V. Por lo que los terminales de esta batería se conectan directamente a los pines de alimentación de los drivers (*V+* y *GND*), que transmitirán la potencia necesaria a los motores.

Para la alimentación del *Arduino*, se le ha puesto un adaptador a la pila que se conecta directamente al puerto *JACK* de la placa (el cual tiene un rango de voltaje de 7 a 12V). Esto permite alimentar a los demás componentes electrónicos desde los pines del *Arduino*.

3.5 Unidad de medidas inerciales *MPU-6050*

Esta *IMU* es la misma que se usó en el anterior vehículo.

Como se dijo anteriormente, en este proyecto, para el control del sistema, será necesario conocer el ángulo de inclinación del vehículo con respecto a al horizontal. Para ello se ha decidido usar la unidad de medidas inerciales *MPU6050*, perteneciente a la compañía Invensense.

Para facilitar su conexión con *Arduino*, se ha comprado ya integrada en el módulo GY-521. Este incluye los complementos necesarios para el correcto uso de la *IMU*, y facilita el acceso a sus pines.

La principal característica de la MPU6050 es que tiene seis grados de libertad, ya que tiene un acelerómetro y un giroscopio, ambos de 3 ejes. En este proyecto solo se usarán tres grados de libertad (dos ejes del acelerómetro y uno del giroscopio), lo que podría hacer pensar que podría encontrarse otra unidad de medidas inerciales más económica e igualmente válida. No obstante, el precio de la MPU6050 es muy reducido, ya que su uso está muy generalizado, por lo que no merece la pena buscar otras alternativas que solo proporcionen las prestaciones estrictamente necesarias.

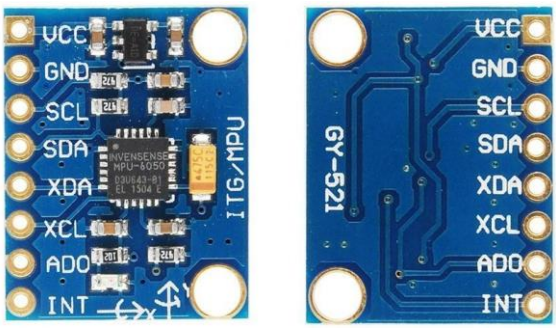

Figura 3-11. IMU MPU6050 integrada en GY-521

Como se ha indicado en el apartado anterior, se usarán los ejes del acelerómetro y un eje del giroscopio.

Ambos sistemas de medida proporcionan el valor de la misma variable, pero, los datos serán útiles:

- El acelerómetro proporciona la aceleración total del dispositivo proyecta en los ejes. Por lo que suponiendo que las aceleraciones que toma son despreciables con respecto a la gravedad (para ello se coloca lo más cerca posible del eje de giro, de forma que la aceleración de caída influya lo mínimo posible), se puede obtener el ángulo a partir del valor de la gravedad y relaciones trigonométricas. Por tanto, el ángulo obtenido es instantáneo (no depende de medidas anteriores). Sin embargo, su precisión no es demasiado buena.
- El giroscopio proporciona el valor de las velocidades angulares. En este proyecto solo es necesario obtener la velocidad angular (del vehículo completo) paralela a los ejes de los motores. Esta, junto al valor del incremento de tiempo entre medida y medida, permite calcular incrementos de ángulo, para actualizarlo a partir del valor anterior. La precisión del giroscopio es alta, pero, como se puede ver, este sistema va acumulando error medida tras medida.

La unión de ambas medidas usando un filtro complementario, con la siguiente estructura:

$$\phi_{comp} = x \cdot (\phi_{prev} + \phi_{giro}) + y \cdot \phi_{acel}$$
$$x + y = 1$$
$$x \gg y$$

La decisión tomada ha sido que el valor de *x* (peso del giroscopio) sea mucho mayor que el de *y* (peso del acelerómetro). De esta forma se tiene la precisión del giroscopio, y el error que este acumula es contrarrestado por el acelerómetro.

La MPU6050 consta de convertidores anológico-digital, tres para los ejes del acelerómetro y otros tres para los del giroscopio. Como se verá más adelante, es posible variar la precisión de estos.

También se dispone de una memoria *FIFO*, sin embargo, su uso es complicado y no aporta demasiada mejora al sistema, por lo que no se entrará en detalle.

La transmisión de datos entre el microcontrolador y la *IMU* se realizará a través del sistema de comunicación *I^2C*, con una frecuencia de 400 *kHz*.

La alimentación ha de ser entre 2.375 y 3.46 V, pero gracias al módulo GY-521 se puede usar directamente una tensión de 5 V.

3.7.1 Comunicación I²C

La comunicación I^2C fue desarrollada por *Philips*. Se caracteriza por necesitar únicamente dos cables para todo el conexionado. Uno de ellos transmite la señal de reloj y el otro los datos. Utiliza una estructura tipo Maestro-Esclavo, en la cual un dispositivo debe ser maestro y los demás esclavos. El maestro se encarga de iniciar la comunicación, y enviar y pedir datos a los esclavos. Aunque un esclavo puede convertirse en maestro (siempre que el anterior pase a ser esclavo), no es usual, debido a la complejidad.

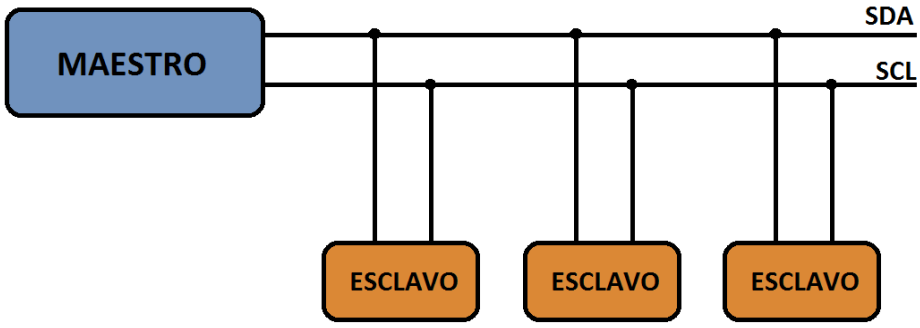

Figura 3-12. Esquema de comunicación I2C

La señal de reloj es generada únicamente por el maestro, lo que facilita la compatibilidad entre los dispositivos. Cada uno tiene una dirección asignada en el bus de datos, que permite el acceso de manera individual. Esta dirección consta de 7 bits, lo que permite tener 112 dispositivos (2^7 – 16 reservadas).

Para el inicio de la comunicación y la transmisión de datos se usa un sistema codificado de la misma forma que se explica en la siguiente figura.

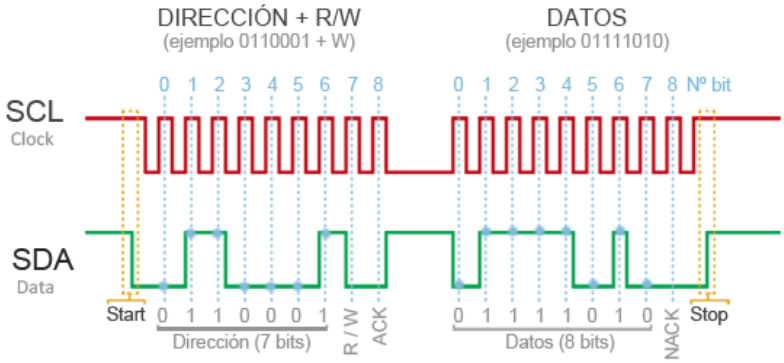

Figura 3-13. Proceso de comunicación I2C

Cuando no está iniciada la comunicación tanto la señal de reloj como la de datos se encuentra a uno. En ese momento, un flanco de bajada en *SDA* indica el inicio de la comunicación. Los siguientes siete bits determinan la dirección del esclavo, a continuación, se indica lectura o escritura y el siguiente flanco de subida finaliza con el proceso de inicio.

Una vez conectado, la señal de reloj se encontrará a cero y la de datos a uno. Nuevamente un flanco de bajada en *SDA* inicia el proceso de comunicación. Se envían o se reciben paquetes de 8 bits.

El fin de la comunicación se produce cuando hay un flanco de subida en *SDA* estando *SCL* activa (y sin estar en mitad del proceso de transmisión o recibimiento de datos).

3.7.1.1 I²C en Arduino

Arduino cuenta con su propio soporte I2C. En el caso del *Arduino UNO*, a *SDA* y *SCL* se puede acceder desde el pin *A4* y *A5*, respectivamente. Es común usar la librería "*Wire.h*", que simplifica de forma notable el proceso.

3.7.2 Características del giroscopio

Tabla 3-4. Características principales del giroscopio

Característica	Medida
3 convertidores Analógico/Digital	16 bits
Rango de salida programable	±250, ±500, ±1000, y ±2000°/s
Intensidad nominal	3.6 mA

La mayor sensibilidad se obtiene programando el rango a ±250º/s, el cual es suficiente para el sistema estudiado, por lo que se ha decidido usar este.

3.7.3 Características del acelerómetro

Tabla 3-5. Características principales del acelerómetro

Característica	Medida
3 convertidores Analógico/Digital	16 bits
Rango de salida programable	±2g, ±4g, ±8g y ±16g
Intensidad nominal	500 μA

La mayor sensibilidad se obetiene programando el rango a ±2gº/s, el cual es suficiente para el sistema estudiado, por lo que se ha decidido usar este.

Cuenta de interrupciones programables para avisar de que hay una medida disponible, o que se ha detectado altas aceleraciones, etc. Aunque en este proyecto no se ha hecho uso de ellas.

3.8 Módulo *Bluetooth HC-06*

El módulo bluetooth tampoco es diferente al que usaron Antonio Croche y Cecilia González.

Se adquiere un módulo Bluetooth HC-06, este dispositivo se comunica con la placa microcontroladora a través de un puerto serie y permite trabajar en un amplio rango de velocidades de transferencia que van desde 1200 baudios hasta 1382400 baudios.

Este módulo será usado para obtener datos del vehículo sin necesidad de conectar con un cable el ordenador y el segway, es decir, proporciona comunicación inalámbrica. Esto proporciona la opción de obtener los datos en tiempo real, siendo mucho más fácil y rápido el estudio del ajuste del funcionamiento del vehículo.

Este módulo Bluetooth se puede configurar, como se muestra en el código del apéndice, gracias a su memoria interna. El nombre que fue escogido para el dispositivo es SEGWAY y la contraseña 0000. La velocidad de transferencia a la que trabaja es 115000 baudios.

Por último, queda añadir que este dispositivo necesita 5 V de alimentación.

3.9 Arduino UNO

Las diferencias que interesan para la nueva mejora al usar el *Arduino UNO* en vez del *Arduino MEGA* como en el anterior proyecto son:

- Se tiene un precio inferior, lo cual es una gran ventaja en caso de querer realizar varios vehículos.
- Se reduce el número de pines digitales de 54 a 14, por lo que habrá que asegurarse de que no se necesitan más.
- Se disponde la mitad de timers, 3 en vez de 6. Que sean suficientes es el primer problema y de más envergadura que se presenta durante el proyecto.
- Hay 2 interrupciones externas disponibles, en vez de 5. Dejar de usar encóders (necesitarían 4 interrupciones) hace posible el uso de este microcontrolador.

Por lo demás, las diferencias de funcionamiento entre uno y otro son indiferentes para este proyecto.

3.10 Conexiones

La conexión entre todos los componentes sitados se realiza mediante cables únicamente.

Para entender el porqué de la posición de los cables se puede recurrir a la descripción de los componentes y la función de sus pines.

Como se puede ver, el *Arduino* es componente central, lo cual es lógico ya que es el microcontrolador del sistema. Por tanto, las conexiones pueden ser presentadas en función de las entradas y salidas de esta placa.

Tabla 3-6. Conexiones *Arduino-Componentes*

Arduino UNO – Driver A3967 del Motor *A*	
Pin digital 4	*STEP*
Pin digital 5	*DIR*
Pin digital 8	*MS1*
Pin digital 9	*MS2*
Arduino UNO – Driver A3967 del Motor *B*	
Pin digital 7	*STEP*
Pin digital 6	*DIR*
Pin digital 12	*MS1*
Pin digital 13	*MS2*
Arduino UNO – MPU6050	
3.3V	*Vcc*

GND	*GND*
Entrada Analógica *A4 (SDA)*	*SDA*
Entrada Analógica *A5 (SCL)*	*SCL*
Arduino UNO – Bluetooth HC-06	
5V	*Vcc*
GND	*GND*
Pin digital 0 (*RXD*)	*TXD*
Pin digital 1 (*TXD*)	*RXD*

Además de estas conexiones, debes ser alimentados el *Arduino*, a través del puesto *Jack*, y los drivers, a través de *Vcc y GND*, por la batería *Li-Po*.

En la siguiente figura se pueden ver estas conexiones de forma visual:

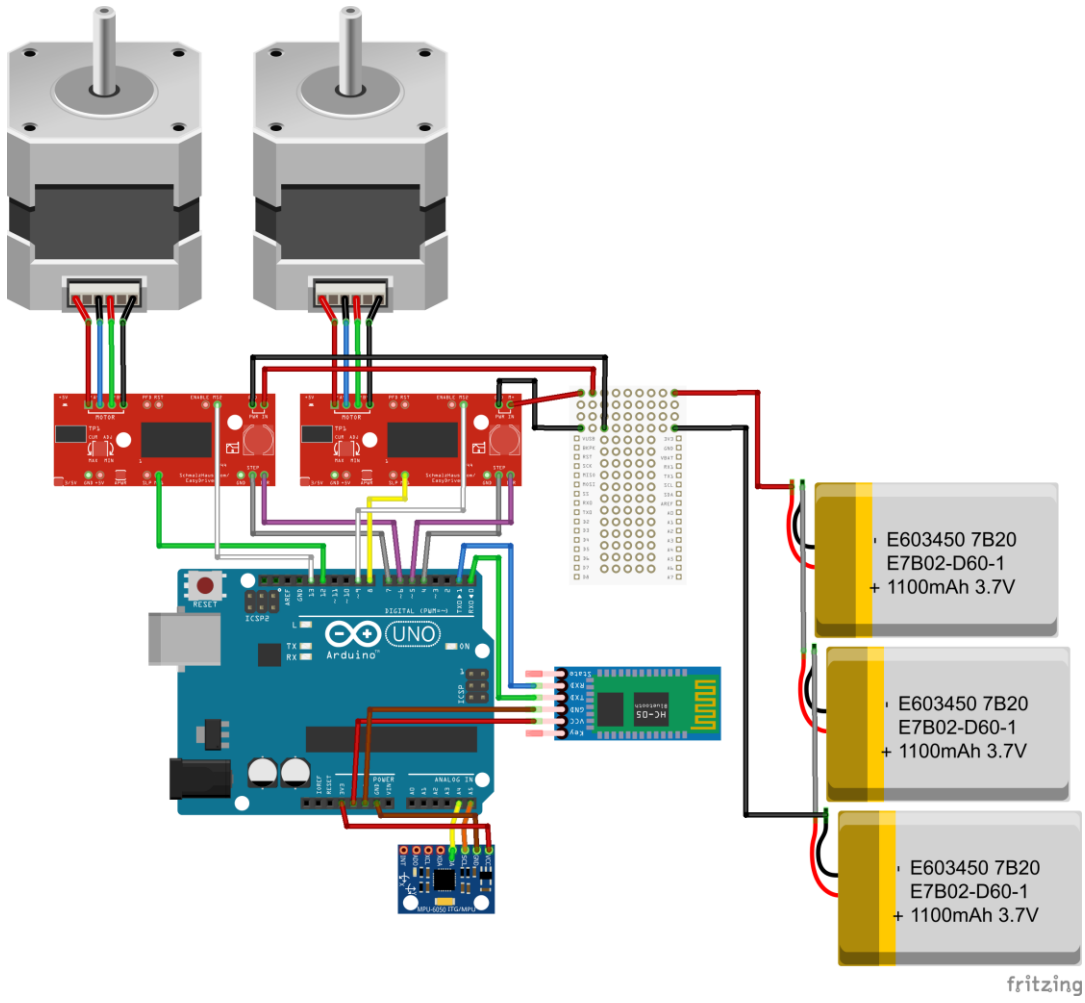

Figura 3-14. Esquema completo de conexiones

4 IMPLEMENTACIÓN ACELERACIÓN

Como ya se dijo anteriormente, la respues de nuestro sistema para controlar el vehículo es la aceleración de las ruedas. A partir de esta se consigue que el vehículo permanezca en su punto de equilibrio. Por lo que tener control sobre la aceleración es imprescindible en este proyecto.

Haciendo un análisis de los cambios en los componentes, se puede ver que la gran diferencia entre el anterior proyecto y este es la producción del movimiento de las ruedas. Las otras diferencias son simplemente de la estructura, por lo que no influyen en la programación.

El cambio del uso de motores DC a motores paso a paso, como ya se dijo anteriormente, se debe a que estos últimos tienen mucha más precisión que los DC al no necesitar realimentación de encóder para conocer las características del movimiento, se les puede aplicar velocidades prácticamente exactas a la referencia, no presentan zona muerta ni holguras, en el caso de no perder pasos los incrementos de posición pueden ser perfectamente conocidos con el contaje de estos, etc.

Como también se mencionó anteriormente, la sustitución de estos motores hace que haya que usar dos algoritmos, uno para implementar velocidad y otro para ir modificando esta y conseguir aceleración. Estos sustituyen los controladores PID que se usaban en el proyecto anterior para implementar aceleración a partir de la señal de encóder y el voltaje aplicado a los motores.

Para aplicar la velocidad, lo que hay que enviar es un tren de pulsos, con el periodo necesario, al pin *STEP* del driver de cada motor. Como se quiere dotar al vehículo de capacidad de giro, ambas velocidades deben poder ser distintas. Este problema parece tener fácil solución: Se crean 2 señales *PWM* en *Arduino* y directamente se tiene la velocidad que se desee. Esto sería posible, sin embargo, al usar un *Arduino UNO,* para este proyecto no lo es.

El *Arduino UNO*, como se explicará más adelante, consta de 3 timers, y uno de ellos prácticamente queda ocupado para operaciones propias. Por lo que, al generar dos señales *PWM,* se ocuparían los dos restantes. De forma que para el control y cálculo del ángulo no se dispondría de ningún timer accesible.

Por tanto, habrá que buscar un sistema en el que con un único timer se controlen los dos motores (y así dejar uno libre para las demás operaciones). Esto estará explicado en el apartado "4.3. Estrategia para implementar tren de pulsos para velocidad de los motores". Aunque, básicamente, consiste en usar las dos interrupciones de las que dispone un timer, y cada en ellas enviar un pulso al pin *STEP*.

Una vez se haya conseguido el anterior objetivo, hay que implementar un algoritmo capaz de modificar la velocidad de forma que se consiga un perfil de aceleración deseado. Esto se explica en el apartado "4.4 Estrategia para implementar aceleración en lo motores". Su funcionamiento básico consiste en ir variando el periodo de los trenes de pulsos que generan el movimiento del motor.

4.1 Interrupciones en Arduino

Como ya se ha indicado, las interrupciones serán necesarias para generar los trenes de pulsos. La explicación del motivo detallada se hará más adelente. De momento se introducen los conceptos básicos:

Cuando se está ejecutando un programa, suelen producirse ciertas condiciones o acontecimientos por las que hay que realizar una operación concreta. A estas circunstancias se les denomina evento.

La detección del evento es un problema común, que se resuelve normalmente de dos formas posibles.

La primera forma es la denominada "poll", que consiste en estar continuamente cada un tiempo fijo comprobando si se ha producido o no. Esta forma es más sencilla, pero tiene muchas desventajas, las cuales hacen que en numerosas ocasiones sea imposible de implementar:

1. Supone un alto coste de procesamiento. Ya que se está continuamente comprobando el estado, y en la mayoría de las ocasiones la respuesta no será útil (no se cumple el evento).
2. El hecho de tener que estar continuamente realizando la búsqueda, hace que otros procesos no puedan ser ejecutados de forma paralela. Al menos si el problema no es muy simple.
3. Si la respuesta al evento debe ser inmediata, este sistema no puede asegurarlo. De hecho, la probabilidad de que se responda al instante es prácticamente nula.
4. Si la duración del evento es inferior al tiempo de espera, puede que se produzca y desaparezca antes de ser detectado.

Por tanto, es evidente que este método solo se puede utilizar en tareas muy sencillas y con poca importancia. Por ello, los microprocesadores incorporan las "interrupciones".

Una interrupción es una parada temporal de la ejecución de un evento para ejecutar una subrutina que corresponde al evento que ha generado la interrupción. Cuando se ha ejecutado la subrutina, se continúa ejecutando el proceso interrumpido por el mismo punto en que lo dejó.

Por ejemplo, la activación de un sensor o un pulsador, que se produzca un error, la activación de un temporizador, etc., podrían ser considerados eventos para generar una interrupción.

La función correspondiente a una interrupción tiene la siguiente nomenclatura en Arduino:

```
ISR("Nombre del registro correspondiente")
{
// Proceso
}
```

En este proyecto, el tipo de interrupciones que se usarán fundamentalmente serán las interrupciones temporales, las cuales van asociadas al uso de temporizadores. Estos permiten producir eventos en instantes determinados de tiempo. Los hay de dos tipos:

- Temporizadores de un solo disparo: Avisan únicamente una vez desde que se activan hasta que se cumple el tiempo especificado. No tienen mucho interés para la programación en *Arduino*.
- Temporizadores periódicos: Serán muy útiles en este proyecto. Se caracterizan por permitir generar eventos con un periodo especificado.

A los temporizadores periódicos normalmente se les llama (y será el término utilizado en este proyecto) "timers".

El funcionamiento de un timer, muy simplificadamente, consiste en un contador (un registro de un número determinado de bits), que va incrementandose una unidad cada vez que se produce un flanco en la señal de reloj del microprocesador. La señal de reloj es siempre fija, y se debe conocer su frecuencia, lo que permite tener noción del tiempo. Cuando el contador alcance un valor determinado, se producirá el evento.

4.2 Timers en Arduino UNO

4.2.1 Introducción a los registros

Para poder explicar el funcionamiento de los timers, se hará continuamente referencia a sus registros, por tanto, aquí se hace una introducción a estos. No se entra muy detalladamente en para qué sirve cada uno. Se recomienda tener este apartado accesible en la lectura de las siguientes secciones, ya que clarifica el entendimiento.

Para empezar, un registro es un conjunto de datos estructurados. Cada dato tiene su propia dirección, a través de la cual se puede consultar o modificar su valor.

Para modificar su valor, en este proyecto, se usarán los llamados registros de desplazamiento. Esto se explicará a partir del siguiente ejemplo.

Ejemplo 4–1. *Supóngase que se tiene un registro llamado REGEJEM, que consta de 4 bits. Normalmenta, a cada posición se le asigna un identificador. Suponiendo que estas serían: POS0, POS1, POS2 y POS 3, y que sus valores iniciales son 0, 1, 1 y 0, queda el siguiente esquema final:*

REGEJEM	*POS3*	*POS2*	*POS1*	*POS0*
Valor	0	1	1	0

Para crear un registro de desplazamiento habría que escribir, por ejemplo, "1<<POS3", que es igual que escribir "1000". Haciendo uso de esto, se puden modificar los valores de la siguiente forma:

- Poner *POS3 = 1:*

  REGEJEMP |= 1<<POS3.

  *S*i nos fijamos es igual que hacer *REGEJEM = 0110 OR 1000 = 1110.*

- Poner *POS2 = 0:*

  REGEJEM &= ~(1<<POS2).

  *S*i nos fijamos es igual que hacer: *REGEJEM = 0110 AND NOT(0100) = 0110 AND 1011 = 0010*.

Una vez se tiene claro qué es y cómo modificar un registro de desplazamiento, se presentan los registros de los timers que se usarán en este proyecto, con una breve explicación que se puede complementar con los apartados posteriores, los cuales se indican. El carácter "x" se refiere al timer, que puede ser 0, 1 o 2:

- TCNTx: Valor del contador asociado al timer. Es un entero comprendido entre 0 y 255 para el timer 0 y 2, y entre 0 y 65535 para el timer 1. Cada flanco de reloj lo incrementa una unidad, de forma que este valor sirve de referencia para referencias temporales. Su valor vuelve a ser 0 cuando se alcanza el máximo o cuando se resetea. Más información en el apartado "4.2.2 Características principales".
- OCRxA y OCRxB: Valores para que salten las interrupciones A o B, respectivamente. Cuando TCNTx se iguale a alguna de estas variables, se activará la interrupción correspondiente. En modo normal, si están activadas, se ejecutará la función de interrupción. En modo CTC sirven para resetear el valor del contador cuando se iguala a cualquiera de las dos, en el PWM para activar y desactivar la señal de salida, etc. Más información en los apartados "4.2.3 Modos de temporización" y "4.2.4 Interrupciones Timers".

- TIMSKx: Se denomina máscara del timer. Sirve para activar o desactivar las interrupciones asociadas. Sus posiciones OCIExA y OCIExB activan y desactivan las interrupciones A y B del timer, las cuales se producen cuando TCNTx = OCIExA/B. La posición TOIEx activa y desactiva la interrupción por *OverFlow* del timer. Más información en el apartado "4.2.4 Interrupciones Timers".
- TCCRxA y TCCRxB: Permiten variar las características de los timers. Según las posiciones a las que se le varía el valor se modificará una característica u otra:
  - CSx0, CSx1 y CSx2: Posiciones de los registros TCCRxA/B para el prescalado del timer, de forma que TCNTx se incrementa cada tantos flancos de reloj como prescalado haya especificado. Más información en el apartado "4.2.2.1 Prescalado Timers".
  - WGMx3, WGMx2, WGMx1 y WGMx0: Posiciones de los registros TCCRxA/B para cambiar su modo de funcionamiento. Más información en el apartado "4.2.3 Modos de temporización".

4.2.2 Características principales

Para explicar los timer en Arduino, es necesario primero conocer algunas características básicas del Arduino.

El Arduino UNO tiene una frecuencia de reloj de 16Mhz. Esto significa que cada 62.5 ns se produce un flanco en la señal de reloj.

El Arduino UNO, consta de 3 timers programables:

- *Timer 0*: Tiene 8 bits de resolución, lo que quiere decir que el contador, *TCNT0*, puede alcanzar un valor máximo de (2^8-1) = 255 (de 0 a 255). Este temporizador es usado por funciones de tiempo muy comunes como *millis()*, *micros()*, *delay()*… Por lo que si se varía su modo de funcionamiento quedarám inhabilitadas. No obstante, es posible hacer uso de sus interrupciones, aunque el periodo por defecto asociado no pueda ser modificado. Tiene asociado los pines 5 y 6 para generar señales PWM.
- *Timer 1*: Tiene 16 bits de resolución, lo que quiere decir que el contador, *TCNT1*, puede alcanzar un valor máximo de (2^{16}-1) = 65335 (de 0 a 65335). Tiene los pines 9 y 10 asociados a la señal PWM.
- *Timer2*: Es equivalente al *Timer 0*, con la única diferencia de que no está asociado a las funciones de tiempo típicas y los pines asociados a la señal PWM son 3 y 11.

4.2.2.1 Prescalado Timers

Como se ha podido ver en apartados anteriores, el fin de los timers es generar señales con un periodo determinado.

Con la frecuencia de reloj y la resolución de los timers se pueden conocer los periodos:

- El reloj genera un pulso cada (1/16 MHz) = 62.5 nseg.
- El timer 1 permite un máximo de (2^{16}-1) = 65535 incrementos. El timer 0 y 2 permite 2555
- Periodo máximo con timer 1: 62.5nseg*65535 = 4.096 ms.
- Periodo máximo con timer 0 y 2: 62.5nseg*255 = 15.94 μs.

Los preescaladores permiten aumentar estos periodos de forma que en vez de incrementarse *TCNTx* cada pulso de reloj, lo haga cada un número de veces definido por el usuario.

Para ello habría que modificar los registros de la siguiente forma:

Tabla 4-1. Prescaladores Timers 0 y 1

CSx2	CSx1	CSx0	Descripción

0	0	1	Prescala = 1
0	1	1	Prescala = 64
0	1	0	Prescala = 256
1	0	1	Prescala = 1024

Tabla 4-2. Prescaladores Timer 2

CS22	CS21	CS20	Descripción
0	0	1	Prescala = 1
0	1	0	Prescala = 8
0	1	1	Prescala = 32
1	0	0	Prescala = 64
1	0	1	Prescala = 128
1	1	0	Prescala = 256
1	1	1	Prescala = 1024

Por tanto, se puden conseguir los siguientes periodos:

Tabla 4-3. Periodos máximos con timers prescalados

Prescala / Timer	1	8	32	64	128	256	1024
0	15.94 µs	127.52 µs	510.08 µs	1.021 ms	2.040 ms	4.086 ms	16.343 ms
1	4.096 ms	---	---	262.14 ms	---	1.0486 s	4.1942 s
2	15.94 µs	---	---	1.021 ms	---	4.086 ms	16.343 ms

4.2.3 Modos de temporización

Los temporizadores se pueden configurar en hasta dieciséis modos distintos. Como vemos en las siguientes tablas.

Tabla 4-4. Registros para modo de operación Timer 1

Modo	WGM13	WGM12	WGM11	WGM10	Modo de operación
0	0	0	0	0	Normal
1	0	0	0	1	PWM, Phase Correct, 8-bit
2	0	0	1	0	PWM, Phase Correct, 9-bit
3	0	0	1	1	PWM, Phase Correct, 10-bit
4	0	1	0	0	CTC
5	0	1	0	1	Fast PWM, 8-bit
6	0	1	1	0	Fast PWM, 9-bit
7	0	1	1	1	Fast PWM, 10-bit
8 … 15	…	…	…	…	…

Tabla 4-5. Registros para modo de operación Timer 0 y 2

Modo	WGMx3	WGMx2	WGMx1	WGMx0	Modo de operación
0	0	0	0	0	Normal
1	0	0	0	1	PWM, Phase Correct
2	0	0	1	0	CTC
3	0	0	1	1	Fast PWM
4	0	1	0	0	Reservado
5	0	1	0	1	PWM, Phase Correct
6	0	1	1	0	Reservado
7	0	1	1	1	Fast PWM
8 … 15	…	…	…	…	…

Básicamente destacan tres:

- Modo output compare (CTC): Este modo de operación consiste en fijar el valor máximo que debe alcanzar el contador del timer para resetarlo. Como se ha dicho anteriormente, se puede acceder al valor del contador a través del registro *TCNTx*. Para fijar la cota superior, se indica en los registros *OCRxA* y *OCRxB*. Cuando *TCNTx* sea igual a cualquiera de estos dos valores saltará la interrupción correspondiente al valor que se haya alcanzado, se reseteará el contador y se repetirá el proceso. Es decir, si *OCRxA* es menor que *OCRxB*, *TCNTx* se reseteará cuando se iguale al valor *OCRxA*. Si de lo

contrario *OCRxB* es menor que *OCRxA*, se reseteará cuando se iguale al valor *OCRxB*.

Para el timer 1 habría que poner: WGM1[3:0] = 0100, y para el timer 0 o 2: WGMx[3:0] = 0010.

Es importante tener en cuenta que el valor de OCRxA/B no puede sobrepasar el valor de resolución del timer (255 para el 0 y 2 y 65535 para el 1). Además, si su valor es nulo, no habrá interrupción.

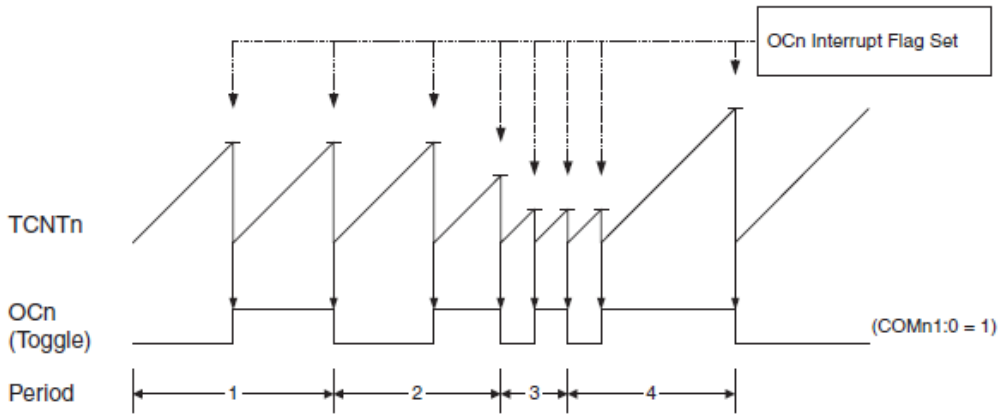

Figura 4-1. Funcionamiento del modo CTC

- Modo Fast PWM: Este modo de operación permite generar una señal por ancho de pulso. El valor *TCNTx* siempre llega al máximo. Hay que asignar a *OCRxA* o *OCRxB* el valor correspondiente para producir el ancho del pulso deseado según la siguiente fórmula:

$$OCRxy = Porcentaje\ PWM * Valor\ máximo\ TCNTx/100$$

De esta forma, mientras *TCNTx* sea inferior a *OCRxy*, la salida será positiva y en caso contrario nula.

No obstante, esto es válido cuando se tiene que la señal de salida no esté negada. En caso contrario, mientras *TCNTx* sea inferior a *OCRxy*, la salida será nula y en caso contrario positiva.

Para que la salida no esté negada hay que poner las posiciones del registro *COMx1 = 1* y *COMx2 = 0*, y para negarla *COMx1 = 1* y *COMx2 = 1*.

Según las tablas 4-3 y 4-2, para conseguir este modo en el timer 1 habría que poner: WGM1[3:0] = 0101, y para el timer 0 o 2: WGMx[3:0] = 0011.

Es importante tener en cuenta que el valor de OCRxA/B no pude sobrepasar el valor de resolución del timer (255 para el 0 y 2 y 65535 para el 1).

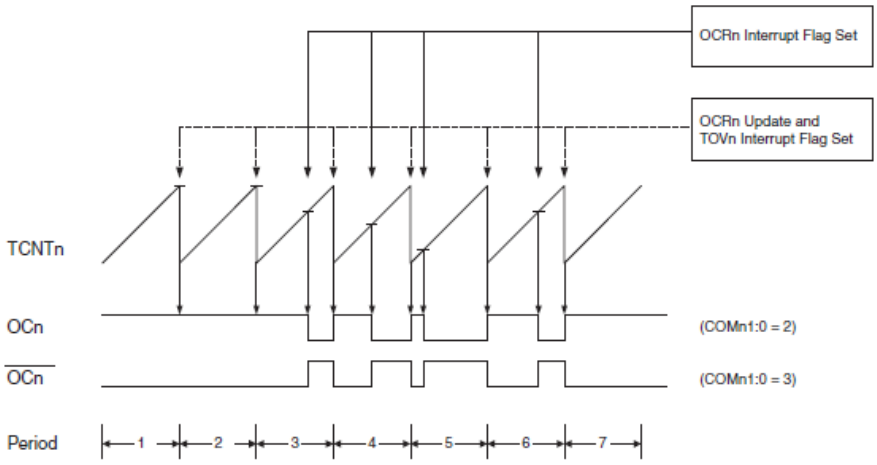

Figura 4-2. Funcionamiento del modo Fast PWM

Como se ve en la figura, mientras *TCNTx* es menor que *OCRxy*, la señal de salida no negada es positiva. Cuando *TCNTx* supera *OCRxy* la señal anterior se anula. En el caso de que la señal sea negada, el funcinamiento es al contrario.

- Modo normal: Es el modo más importante en este proyecto, ya que será el único que se use. Esto se debe a que, aunque es el más básico, es el que más libertad de programación permite, y con esta se pueden conseguir los demás modos de operación.

  La forma de utilizarlo es modificar el valor *OCRxA/B*. *TCNTx* irá incrementándose continumente del valor mínimo al máximo. Cuando su valor se iguale a alguno de los registros previamente indicados, saltará la interrupción correspondiente, y *TCNTx* seguirá aumentando.

  Habría que activar los avisos de estas interrupciones (modificando los registro de *TIMSKx*) y crear sus procesos correspondientes según la siguiente tabla:

Tabla 4-6. Interrupciones Timers

Registro	Función asociada	Interrupción
OCIExA	*ISR(TIMERx_COMPA_vect)*	*TCNTx = OCRxA*
OCIExB	*ISR(TIMERx_COMPB_vect)*	*TCNTx = OCRxB*
TOIEx	*ISR(TIMERx_OVF_vect)*	*TCNTx OverFlow*

4.2.4 Interrupciones Timers

Como se ha dicho anteriormente, cada timer consta de tres interrupciones. Las interrupciones A y B se producen cuando el valor *OCRxA/B* se iguala a *TCNTx*, y la interrupción por *OverFlow* se produce cuando *TCNTx* alcanza su máximo valor posible (255 o 65535, según el timer).

De esta forma, si está programado el modo normal y teniendo en cuenta la prescala y la resolución del timer,

se pueden generar interrupciones separadas un periodo deseado, con las siguientes ecuaciones:

$$T_A = OCRxA * \frac{Prescala}{16000000}$$

$$T_B = OCRxB * \frac{Prescala}{16000000}$$

$$T_{OVF} = Máx(TCNTx) * \frac{Prescala}{16000000}$$

Para activarlas hay que modificar la máscara de los timers de la siguiente forma (se hace para el caso del timer 1, pero es exactamente igual para los demás):

```
TIMSK1 = 0; //Limpiar la máscara del timer
TIMSK1 |= (1 << OCIE1A); //Activar interrupción A
TIMSK1 |= (1 << OCIE1B); //Activar interrupción B
TIMSK1 |= (1 << TOIE1); //Activar interrupción OverFlow
```

Por último, habría que crear la función que se ejecutará cuando se se produzca el evento:

```
ISR(TIMER1_OVF_vect)
{
  //Proceso cuando se genere OverFlow en el timer (TCNT1 = 65535)
}
ISR(TIMER1_COMPA_vect)
{
  //Proceso cuando se alcance el valor de A (TCNT1 = OCR1A)
}
ISR(TIMER1_COMPB_vect)
{
  //Proceso cuando se alcance el valor de B (TCNT1 = OCR1B)
}
```

Los procesos citados deben ser lo más breve posible, ya que si son densos habrá problemas con la ejecución de las distintas interrupciones que se produzcan y el programa dejará de funcionar.

En el caso de que sea muy necesario realizar un proceso largo dentro de una interrupción, puede usarse la orden *interrupts()*, que permite que las demás interrupciones se ejecuten sin haber acabado la que llamó a la función, y el programará seguirá funcionando.

4.3 Estrategia para implementar tren de pulsos para velocidad de los motores

Para tener control sobre la aceleración primero necesitamos tener sobre la velocidad.

En este proyecto se dispone de dos motores paso a paso acompañados de su driver. Como ya se dijo

anteriormente para generar un paso en uno de los motores hay que enviar un flanco positivo en el pin *STEP* correspondiente. Según las especificaciones de los drivers, la duración mínima del pin activo debía ser un microsegundo.

Los motores tendrán distintas velocidades, para poder permitir el giro del cuerpo, por lo que se necesitarán dos señales independientes, es decir, un tren de pulsos para cada motor.

El problema principal de esto era que el Arduino UNO dispone de tres timers, y el hecho de necesitar distintas señales para los motores hacía pensar en un principio que ocuparía el uso de dos de ellos. Además, la resolución del timer 1 es mucho mayor que la de los demás, por lo que se estaría desaprovechando su capacidad .

Tras varias pruebas, se descubrió que había una forma de utilizar el timer 1 para mover los dos motores sin necesidad de que siempre tuviesen la misma velocidad. Para ello hay que hacer uso de las interrupciones A y B del timer, una para cada señal.

La estrategia consiste en que el programa decide el número de incrementos del contador del timer que corresponde al periodo del tren de pulsos, teniendo en cuenta que el timer 1 tiene una resolución de 65536 y se prescala a 8 (32.77ms máximo periodo). De esta forma se tendría *npulsA* y *npulsB*, cada una correspondiente a uno de los motores. El timer estará programado en modo normal, por lo que siempre cuenta hasta su máximo valor. Seguidamente se realiza el siguiente proceso:

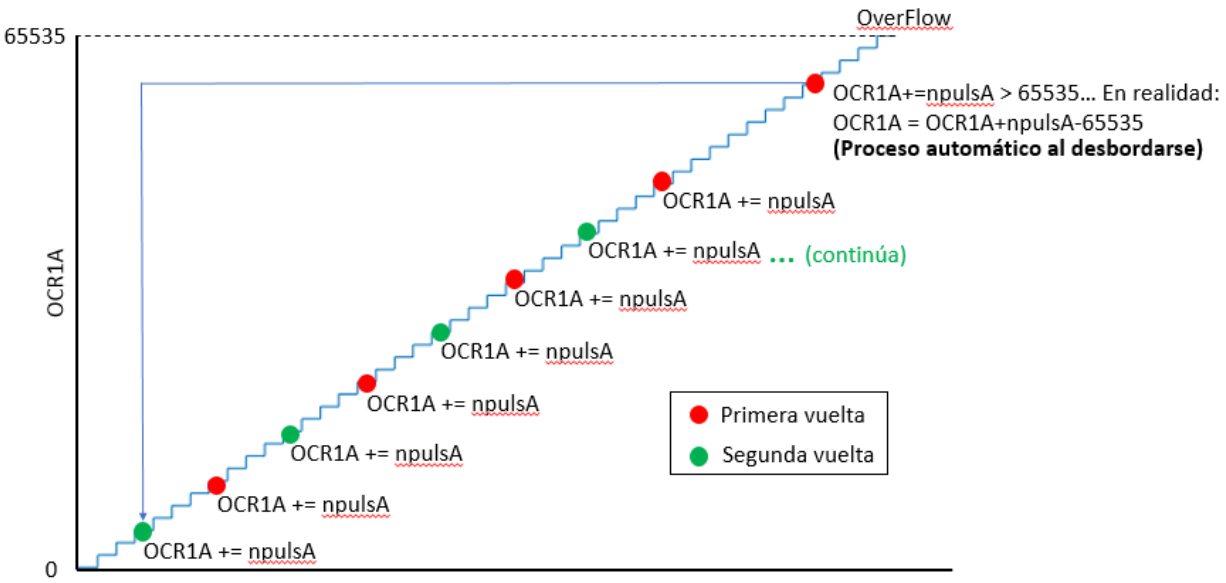

Figura 4-3. Estrategia tren de pulsos usando una interrupción y timer a OverFlow

Como se puede ver, en la figura solo está representada la interrupción A del timer 1, pero este cuenta con dos, por lo que la B realizaría el mismo proceso.

La función asociada la interrupción A, por ejemplo, es la siguiente:

```
ISR(TIMER1_COMPA_vect) //Generamos un pulso de aprox. 4µs
{
        PORTD|=(1<<step_pin_A);
        OCR1A=OCR1A+npuls_A;
        PORTD&=(~(1<<step_pin_A));
```

}

Resumiendo, el funcionamiento consiste en autoincrementar el valor con el que salta la interrupción cuando esta se produce. Así se pueden tener hasta 65535 interrupciones A o B desde que el contador del timer es nulo hasta que se realiza el OverFlow.

Esta estrategia permite controlar dos motores sin ningún problema, y teniendo toda la resolución del timer 1. No obstante, el código de estas interrupciones debe ser mínimo, ya que se van a ejecutar muchísimas más veces que las de otros procesos. Por tanto, se pensó que podría ser un problema el hecho de tener que tener en cuenta si el valor *OCR1A/B* más el de *npulA/B* sobrepasaba el máximo, 65535. Sin embargo, esta comprobación era imnecesaria, ya que el truncamiento lo hacía de forma automática como se puede ver en el siguiente ejemplo:

Ejemplo 4–1. *Proceso OCR1A para generar tren de pulsos*

- *Suponiento OCR1A = 60000 y NpulsA = 1500*
- *Cuando TCNT1 = 60000 -> Interrupción A genera un pulso y OCR1A = 61500*
- *Cuando TCNT1 = 61500 -> Interrupción A genera un pulso y OCR1A = 63000*
- *Cuando TCNT1 = 63000 -> Interrupción A genera un pulso y OCR1A = 64500*
- *Cuando TCNT1 = 64500 -> Interrupción A genera un pulso y OCR1A = 66000*
- *Cuando TCNT1 = 66000 -> Interrupción A genera un pulso y OCR1A debería valer 67500, pero como 67500>65535, sin necesidad de realizar ninguna comprobación, se realiza esta operación:*

  OCR1A = 66000+1500 = 1965, que realmente es igual a 67500 – 65535.

Esto soluciona notablemente el problema de generar dos trenes de pulsos distintos para el movimiento de los motores. La única consideración a tener en cuenta es que *NpulsA/B* no sea mayor que 65535, y que no sea menor que la velocidad que permitan los motores o drivers por sus características físicas.

4.4 Estrategia para implementar aceleración en los motores

Hasta el momento, se sabe que variando *npulsA* y *npulsB*, se varía el periodo de los trenes de pulso de los timer, lo que permite controlar con gran facilidad la velocidad de los motores. El siguiente paso es conseguir implementar la aceleración que recibamos del controlador del sistema.

La primera idea para solventar esto fue actualizar *npulsA* y *npulsB* cada vez que se producía un pulso. Es decir, se le envía un pulso al motor y se calcula en cuanto tiempo hay que enviar el siguiente.

Esta implementación sería ideal, no obstante, no fue posible de realizar.

Cuando se calcula se actualiza *npulsA/B*, hay que tener en cuenta muchas condiciones, y realizar varias operaciones:

- Calcular la nueva velocidad que se va a tener y comprobar que se encuentre en el margen de funcionamiento.
- Comprobar si es positiva o negativa y actuar sobre el pin correspondiente del driver, *DIR*.
- Comprobar, en qué rangos de micropasos puede generarse y activar los micropasos posibles que

proporcionen la mayor precisión.

Por tanto, no es posibler ejecutar todas estas líneas de códigos, que aproximadamente consumen 0.1ms, dentro de interrupciones que se producen, de media, cada 0.3ms, sin que acaben perturbándose entre ellas.

Debido a esto, hubo que buscar una alternativa. Esta fue ir actualizando *npulsA/B* usando el Timer 2.

Este timer está prescalado a 256, programado en modo normal, y con una resolución máxima de 225. Por lo que la interrupción por *OverFlow* se genera cada 4.08ms, periodo con el que se decidió actualizar la velocidad para implementar la aceleración. Este periodo está en relación el cálculo de la aceleración del controlador del sistema, como se puede ver en el siguiente apartado.

El problema de este método es que no se consigue un perfil exacto, si no aproximado, como se puede ver en la siguiente figura.

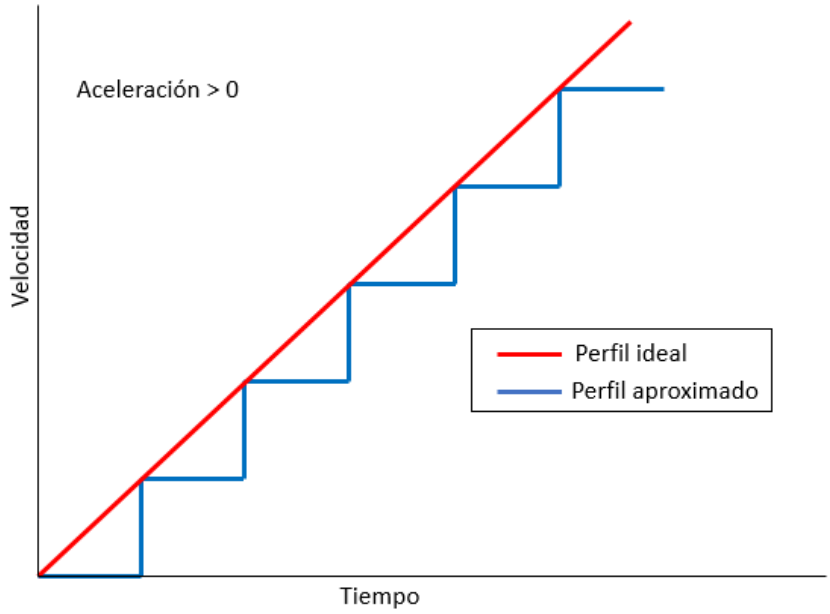

Figura 4-4. Perfil de velocidad ideal frente al aproximado

No obstante, si se actualiza con un periodo lo suficientemente pequeño en comparación con el cálculo de la aceleración, el resultado es suficientemente bueno.

4.5 Cálculo aceleración de los motores

Cuando ya tenemos control sobre la velocidad y aceleración de los motores, el siguiente paso es implementar un controlador que cada un periodo necesario devuelva la aceleración que deben aplicar las ruedas. Puede ser un controlador PID o LQR, según desee el usuario.

Para cumplir con el tiempo especificado, se hace uso de la misma interrupción que en el apartado anterior, la interrupción por *OverFlow* que produce el timer 2.

Como ya se dijo antes, este está prescalado a 256, y el valor máximo que puede alcanzar es 255, por lo que la interrupción se produce cada 4.08 ms. Sin embargo, este periodo no es el que se utiliza para ejecutar el controlador, si no 16.32 ms, que coincide con ejecutarlo cada cuatro interrupciones.

Para conseguir esto se hace uso del siguiente esquema:

ISR(TIMER2_OVF_vect)

```
{
 contLQR++;
        if (contLQR>=4)
        {
                contLQR = 0;
                //Proceso del controlador. Obtención de alfa
        }
 //Continuación con otras operaciones
 }
```

En realidad, primero se calculó que el periodo necesario para el cálculo del controlador era 16.32 ms, y después se decidió actualizar la velocidad para implementar la aceleración calculada con un periodo 4 veces inferior.

La verificación del periodo para el controlador se realizó de la siguiente forma:

- En el capítulo "5. Modelo Físico-Matemático" se llega a la siguiente función del sistema linealizado:

$$\dot{x} = (A - B \cdot K)x$$

Donde: $\dot{x} = \begin{bmatrix} \dot{\phi}_c \\ \ddot{\phi}_c \\ \ddot{\theta} \end{bmatrix}$; $A = \begin{bmatrix} 0 & 1 & 0 \\ -\frac{d}{f} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$; $B = \begin{bmatrix} 0 \\ -\frac{e}{f} \\ 1 \end{bmatrix}$; $K = [570\ 41.07\ 10]$; $x = \begin{bmatrix} \phi_c \\ \dot{\phi}_c \\ \theta \end{bmatrix}$;

Siendo: d = -0.588330225, e = 0.0056420355 y f = 0.012834115.

- Por lo que la matriz del sistema realimentado es:

$$(A - B \cdot K) = \begin{bmatrix} 0 & 1 & 0 \\ -181.43 & -33.59 & -4.40 \\ 516.98 & 76.41 & 10 \end{bmatrix}$$

- Cuyos autovalores son:

$$\begin{bmatrix} -10.27 \\ -6.660 + 0.532i \\ -6.660 - 0.532i \end{bmatrix}$$

- Por lo que se tiene un polo rápido, cuya constante de tiempo es:

$$\tau = \frac{1}{10.27} = 0.1s$$

- Y dos polos lentos con constantes de tiempo:

$$\tau = \frac{1}{6.66} = 0.15s$$

Siguiendo una de las condiciones más usadas en ingeniería de control para calcular el periodo mínimo de muestro: periodo de muestro menor que la sexta parte de tiempo característico del sistema:

$$T_s \leq \frac{T_p}{6}$$

Se puede comprobar que el tiempo de muestreo seleccionado, 0.01632 s, cumple dicha condición para todos los polos del sistema:

$$\mathbf{0.01632}\ \boldsymbol{s}\ \leq \frac{\mathbf{0.1}\ \boldsymbol{s}}{\mathbf{6}} = \mathbf{0.01667}\ \boldsymbol{s}$$

$$\mathbf{0.01632}\ \boldsymbol{s}\ \leq \frac{\mathbf{0.15}\ \boldsymbol{s}}{\mathbf{6}} = \mathbf{0.025}\ \boldsymbol{s}$$

En este apartado solo se explica como se introduce el cálculo usando los timers. El código del controlador se verá en detalle más adelante.

5 MODELO FÍSICO-MATEMÁTICO

Para el cálculo de las componentes del controlador *LQR* es necesario conocer el modelo físico-matemático del sistema a controlar. Para calcularlo se ha hecho uso de las ecuaciones de *Lagrange* partiendo de las varibles que se puden ver en la siguiente figura:

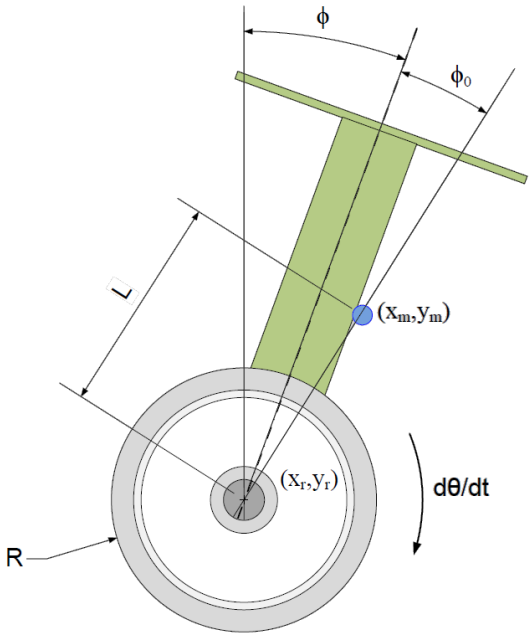

Figura 5-1. Modelo del vehículo con todas las variables

Tabla 5-1. Parámetros totales para el modelo físico-mátematico del vehículo

Parámetro	**Notación**	**Valor**
Masa de una rueda	m_r	*0.044 kg*
Radio de una rueda	R	*0.053 m*
Masa del vehículo sin ruedas	M	*0.745 kg*
Distancia entre el eje de ruedas y el centro de gravedad	L	*0.0805 m*
Aceleración gravitacional	g	*9.81 m/s²*
Ángulo de inclinación con respecto a la vertical	ϕ	
Ángulo de desviación del centro de gravedad	ϕ_0	
Ángulo de inclinación corregido	ϕ_c	

Ángulo de giro de la rueda	θ

El desarrollo para llegar al modelo es el siguiente:

$L = T - V = T_{tras} + T_{rot} - V$	(1) Ecuación de Lagrange
$T_{tras} = \frac{1}{2} m_r (\dot{x}_r^2 + \dot{y}_r^2) \cdot 2 + \frac{1}{2} M (\dot{x}_m^2 + \dot{y}_m^2)$	(2) Energía cinética de traslación
$x_r = R\theta; \quad \dot{x_r} = R\dot{\theta};$	(3) Posición y velocidad horizontal de las ruedas
$y_r = R; \quad \dot{y_r} = 0;$	(4) Posición y velocidad vertical de las ruedas
$x_m = R\theta + L\sin(\phi + \phi_0);$ $\dot{x_m} = R\dot{\theta} + L\dot{\phi}\cos(\phi + \phi_0);$	(5) Posción y velocidad horizontal del centro de gravedad del vehículo
$y_m = R + L\cos(\phi + \phi_0);$ $\dot{y_m} = -L \cdot \dot{\phi}\sin(\phi + \phi_0);$	(6) Posción y velocidad vertical del centro de gravedad del vehículo
$T_{tras} = \left(m_r + \frac{1}{2} M\right) R^2 \dot{\theta}^2 + \frac{1}{2} M L^2 \dot{\phi}^2 + MRL\dot{\theta}\dot{\phi}\cos(\phi + \phi_0)$	(7) Energía cinética de traslación
$T_{rot} = \frac{1}{2} m_r R^2 \dot{\theta}^2 + \frac{1}{2} M L^2 \dot{\phi}^2$	(8) Energía cinética de rotación
$V = MgLcos(\phi + \phi_0)$	(9) Energía potencial
$L = \left(\frac{3}{2} m_r + \frac{1}{2} M\right) R^2 \dot{\theta}^2 + ML^2 \dot{\phi}^2 + MRL\dot{\theta}\dot{\phi}\cos(\phi + \phi_0) - MgLcos(\phi + \phi_0)$	(10) Ecuación de Lagrange con sustituciones
$\frac{\partial L}{\partial \theta} = 0$	(11) Derivada de la ecuación de Lagrange con respecto al ángulo de la rueda
$\frac{\partial L}{\partial \dot{\theta}} = 2\dot{\theta}\left(\frac{3}{2} m_r + \frac{1}{2} M\right) R^2 + MRL\dot{\phi}\cos(\phi + \phi_0)$	(12) Derivada de la ecuación de Lagrange con respecto a la

	velocidad angular de la rueda
$\frac{d}{dt}\frac{\partial L}{\partial \dot{\theta}} = \ddot{\theta}(3m_r + M)R^2 + MRL\ddot{\phi}\cos(\phi + \phi_0) - MRL\dot{\phi}^2\cos(\phi + \phi_0)$	(13) Derivada de la ecuación de Lagrange con respecto a la velocidad angular de la rueda derivada con respecto al tiempo
$\frac{\partial L}{\partial \phi_c} = -MRL\dot{\phi}\dot{\theta}\,\text{sen}(\phi + \phi_0) + MgLsen(\phi + \phi_0)$	(14) Derivada de la ecuación de Lagrange con respecto al ángulo con la vertical
$\frac{\partial L}{\partial \dot{\phi}_c} = 2ML^2\dot{\phi} + \dot{\theta}MRL\cos(\phi + \phi_0)$	(15) Derivada de la ecuación de Lagrange con respecto a la velocidad angular con la vertical
$\frac{d}{dt}\frac{\partial L}{\partial \dot{\phi}_c} = 2ML^2\ddot{\phi} + \ddot{\theta}MRL\cos(\phi + \phi_0) - \dot{\theta}\dot{\phi}MRL\,\text{sen}(\phi + \phi_0)$	(16) Derivada de la ecuación de Lagrange con respecto a la velocidad angular con la vertical derivada con respecto al tiempo
$\frac{d}{dt}\frac{\partial L}{\partial \dot{\theta}} - \frac{\partial L}{\partial \theta} = \tau$ $\frac{d}{dt}\frac{\partial L}{\partial \dot{\phi}_c} - \frac{\partial L}{\partial \phi_c} = -\tau$	(17) Ecuaciones de Lagrange particuralizadas en las coordenadas generales basadas en el torque del motor.
$\frac{d}{dt}\frac{\partial L}{\partial \dot{\theta}} - \frac{\partial L}{\partial \theta} + \frac{d}{dt}\frac{\partial L}{\partial \dot{\phi}_c} - \frac{\partial L}{\partial \phi_c} = 0$	(18) Resultado de la ecuación anterior

De la sustitución las ecuaciones 11, 13, 14 y 16 en la 18, se obtiene el siguiente **modelo del sistema**:

$$(\boldsymbol{2a + c \cdot \cos(\phi + \phi_0)})\,\ddot{\boldsymbol{\theta}} + (\boldsymbol{c \cdot \cos(\phi + \phi_0) + 2b}) \cdot \ddot{\boldsymbol{\phi}} - \boldsymbol{c \cdot \dot{\phi}^2 \cdot \sin(\phi + \phi_0) - d \cdot \sin(\phi + \phi_0) = 0}$$

$$a = \left(\frac{3}{2}m_r + \frac{1}{2}M\right) \cdot R^2$$
$$b = M \cdot L^2$$
$$c = R \cdot L \cdot M$$
$$d = M \cdot g \cdot L$$

6 CONTROLADOR LQR

Partiendo del modelo calculado en el anterior apartado es posible calcular un controlador LQR (*Linear Quadratic-Regulator*) para la estabilización del sistema. Aunque, como primer paso, hay que obtener el sistema linealizado en torno al punto de equilibrio deseado:

Se recuerda que el modelo era:

$$\boldsymbol{F = (2a + c\cos(\phi + \phi_0))\,\ddot{\theta} + (c\cos(\phi + \phi_0) + 2b)\ddot{\phi} - c\dot{\phi}^2 \sin(\phi + \phi_0) - d\sin(\phi + \phi_0) = 0}$$

$$a = \left(\frac{3}{2}m_r + \frac{1}{2}M\right)R^2$$
$$b = ML^2$$
$$c = RLM$$
$$d = -MgL$$

Cuyos parámetros son:

Tabla 6-1. Parámetros totales para el modelo físico-mátematico del vehículo

Parámetro	**Notación**	**Valor**
Masa de una rueda	m_r	*0.044 kg*
Radio de una rueda	R	*0.053 m*
Masa del vehículo sin ruedas	M	*0.745 kg*
Distancia entre el eje de ruedas y el centro de gravedad	L	*0.0805 m*
Aceleración gravitacional	g	*9.81 m/s²*
Ángulo de inclinación con respecto a la vertical	ϕ	
Ángulo de desviación del centro de gravedad	ϕ_0	
Ángulo de inclinación corregido	ϕ_c	
Ángulo de giro de la rueda	θ	

Y se quiere linealizar en torno al punto de equilibrio cuyas condiciones son:

$$\phi_{c_{eq}} = 0;\ \dot{\phi_{c_{eq}}} = 0;\ \ddot{\phi_{c_{eq}}} = 0;\ \cos\left(\phi_{c_{eq}}\right) \approx 1;\ \text{sen}\left(\phi_{c_{eq}}\right) \approx \phi_{c_{eq}};$$

Para linealizar, se obtienen las derivadas de *F* con respecto a cada variable del sistema:

$\frac{\partial F}{\partial \ddot{\theta}_{eq}} = 2a + c$	(1) Derivada de *F* con respecto a la aceleración angular de la rueda
$\frac{\partial F}{\partial \phi_{c_{eq}}} = -d$	(2) Derivada de *F* con respecto al angulo con la verticual
$\frac{\partial F}{\partial \ddot{\theta}_{eq}} = 0$	(3) Derivada de *F* con respecto a la velocidad angular con la vertical
$\frac{\partial F}{\partial \ddot{\theta}_{eq}} = 2b + c$	(4) Derivada de *F* con respecto a la aceleración angular con la vertical

De las anteriores ecuaciones se obtiene el sistema linealizado:

$$F_{lin} = (2a + c)\ddot{\theta} + (2b + c)\ddot{\phi_c} - d\phi_c = 0$$

Si se toma:

$$e = (2a + c)$$
$$f = (2b + c)$$

Entonces:

$$\boldsymbol{F_{lin} = e\ddot{\theta} + f\ddot{\phi}_c - d\phi_c = 0}$$

Se puede obtener el espacio de estado del sistema:

$$\begin{bmatrix} \dot{\phi}_c \\ \ddot{\phi}_c \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ -\frac{d}{f} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \phi_c \\ \dot{\phi}_c \\ \theta \end{bmatrix} + \begin{bmatrix} 0 \\ -\frac{e}{f} \\ 1 \end{bmatrix} \ddot{\theta} \equiv \dot{x} = Ax + Bu$$

Como el *Arduino* trabaja en tiempo discreto, y el anterior sistema es continuo, habría que discretizarlo (el periodo de muestreo en este proyecto ha sido 16,32ms), obteniendo:

$$x_{k+1} = A_d x_k + B_k u_k$$

Las matrices de ponderación de estado, Q, y de la acción de control, R, elegidas son:

$$Q = \begin{bmatrix} 10 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 10 \end{bmatrix}; \; R = 0.1;$$

Siendo la ley de control resultante:

$$u_{k+1} = -Kx_k \equiv \ddot{\theta}_{k+1} = -[570\ 41.07\ 10]\begin{bmatrix}\phi_{c_k}\\ \dot{\phi}_{c_k}\\ \dot{\theta}_k\end{bmatrix}$$

A la que se le añade un término integral para corregir la diferencia entre el centro de gravedad del vehículo y el geométrico. Con esto se consigue el robot no tienda a moverse continuamente hacia el lado donde más peso tenga.

$$\boldsymbol{u_{k+1} = -Kx_k + K_e e_k \equiv \ddot{\theta}_{k+1} = -[570\ 41.07\ 10]\begin{bmatrix}\phi_{c_k}\\ \dot{\phi}_{c_k}\\ \dot{\theta}_k\end{bmatrix} + [0.3]\sum_{i=0}^{k}\dot{\phi}_{c_i}}$$

Donde $K_e = 0.3$ se ha obtenido experimentalmente.

El motivo para añadir este último término se encuentra en el artículo "Low cost two-wheels self-balancing robot for control education, C. Gonzalez, I. Alvarado, D. Muñoz La Peña". Citando textualmente:

"Finalmente se agregó un nuevo término en el control debido a la perturbación causada por la diferencia entre el centro de gravedad del robot y su centro geométrico. Si no son los mismo, el robot tiende a moverse donde pesa más.

Para evitar la influencia de esta perturbación, la ecuación está formulada en variables incrementales en tiempo discreto:

$$\Delta x_{k+1} = A \cdot \Delta x_k + B \cdot \Delta u_k + E \cdot \Delta w_k$$

La perturbación es constante; por lo tanto, su variación es cero.

$$E \cdot \Delta w_k = 0$$

Entonces la ecuación se simplifica:

$$\Delta x_{k+1} = A \cdot \Delta x_k + B \cdot \Delta u_k$$

El error en tiempo discreto y variables incrementales es:

$$\Delta e_{k+1} = A \cdot \Delta r_k - \Delta x_{k+1} = -\Delta x_{k+1}$$

En consecuencia, el sistema es:

$$\begin{bmatrix}\Delta x_{k+1}\\ e_k\end{bmatrix} = \begin{bmatrix}A & 0\\ -I & I\end{bmatrix} \cdot \begin{bmatrix}\Delta x_k\\ e_{k-1}\end{bmatrix} + \begin{bmatrix}B\\ 0\end{bmatrix} \cdot \Delta u_k$$

Suponiendo que la acción de control es una ganancia:

$$\Delta u_k = K_Z \cdot \begin{bmatrix}\Delta x_k\\ e_{k-1}\end{bmatrix} = K_x \cdot \Delta x_k + K_e \cdot \Delta e_{k-1}$$

Entonces se puede obtener que:

$$u_n = u_0 + K_x(x_n - x_0) + K_e \sum_{i=0}^{n-1} e_i$$

Donde Kx es la ganancia LQR calculada en (29). Ke está ajustado experimentalmente."

7 RESULTADOS

Para la verificación del correcto funcionamiento del sistema se ha realizado una serie de pruebas con las que poder analizar el comportamiento del vehículo. Además, se ha usado un simulador que proporciona los resultados esperados, suponiendo un sistema ideal. De esta forma, se pueden comparar datos reales y simulados.

7.1 Ensayo ante perturbaciones de tipo pulso

En esta prueba se realizan varios impulsos de magnitud indefinida en la parte superior del vehículo de forma que tienda a desestabilizarlo durante un corto periodo de tiempo.

En la siguiente gráfica se pueden ver las modificaciones que se producen en las variables más relevantes del sistema: inclinación con respecto a la vertical, velocidad angular del cuerpo, velocidad angular de las ruedas y aceleración angular de las ruedas. En este estudio, las referencias de todas estas variables son cero:

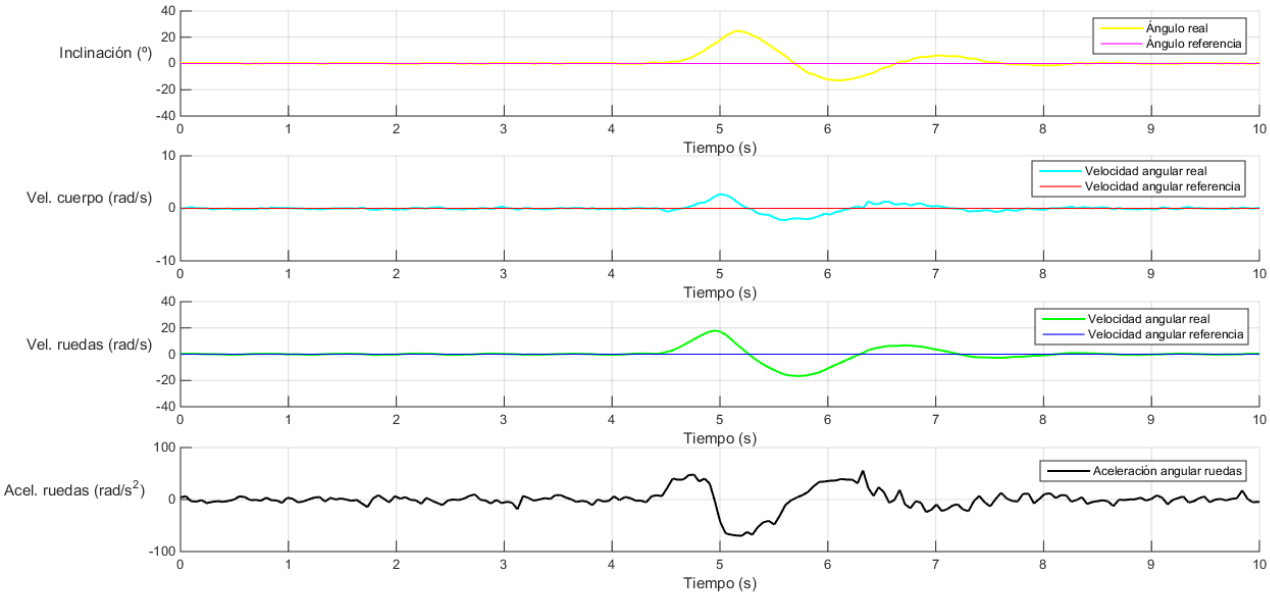

Figura 7-1. Gráfica: Variables frente a impulso

Como se puede ver, cuando el sistema está estabilizado en torno a su punto de equilibrio (cuatro primeros segundos), se realiza un impulso de forma que el ángulo (ϕ) y velocidad ($d\phi/dt$) del cuerpo con respecto a la vertical aumenten su valor en poco tiempo.

Instantáneamente, el controlador aumenta la aceleración de las ruedas para evitar la caída del sistema, por lo que la velocidad de las ruedas también aumenta. De esta forma consigue cambiar el sentido de la velocidad angular del cuerpo y disminuir su ángulo, momento en que la aceleración cambia de sentido para que la velocidad de las ruedas vaya disminuyendo.

Se puede ver que se producen algunas sobreosciolaciones, pero el sistema se estabiliza en poco tiempo.

Por mayor claridad, se presenta la misma gráfica, pero separada para cada varible:

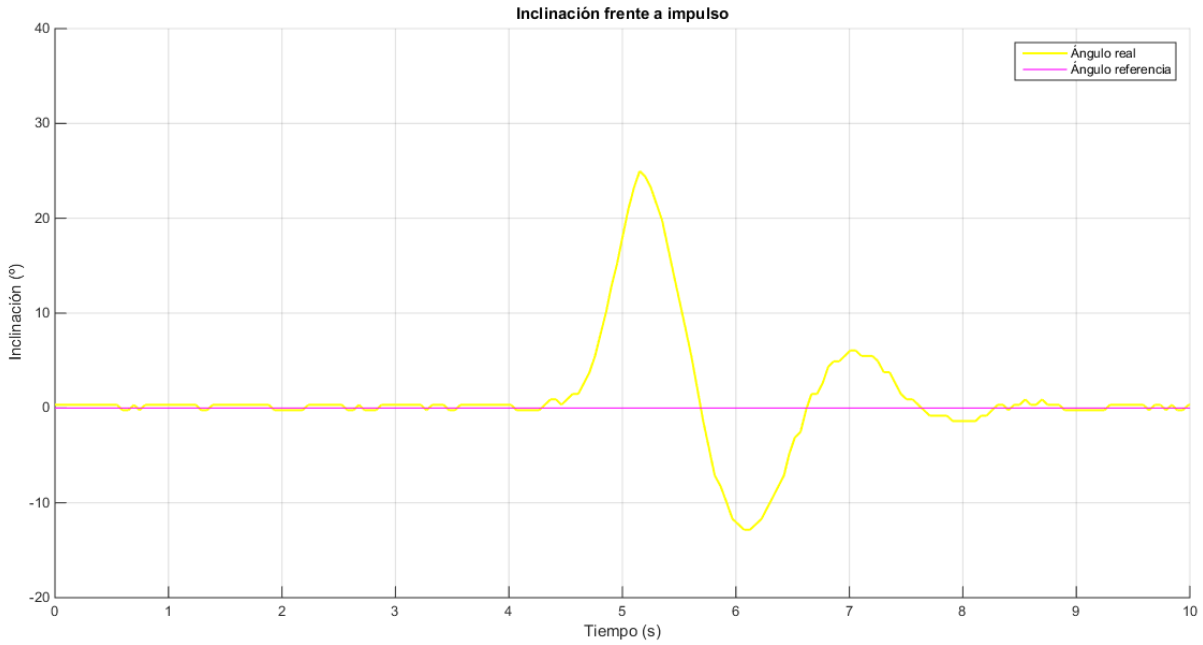

Figura 7-2. Gráfica: Inclinación frente a impulso

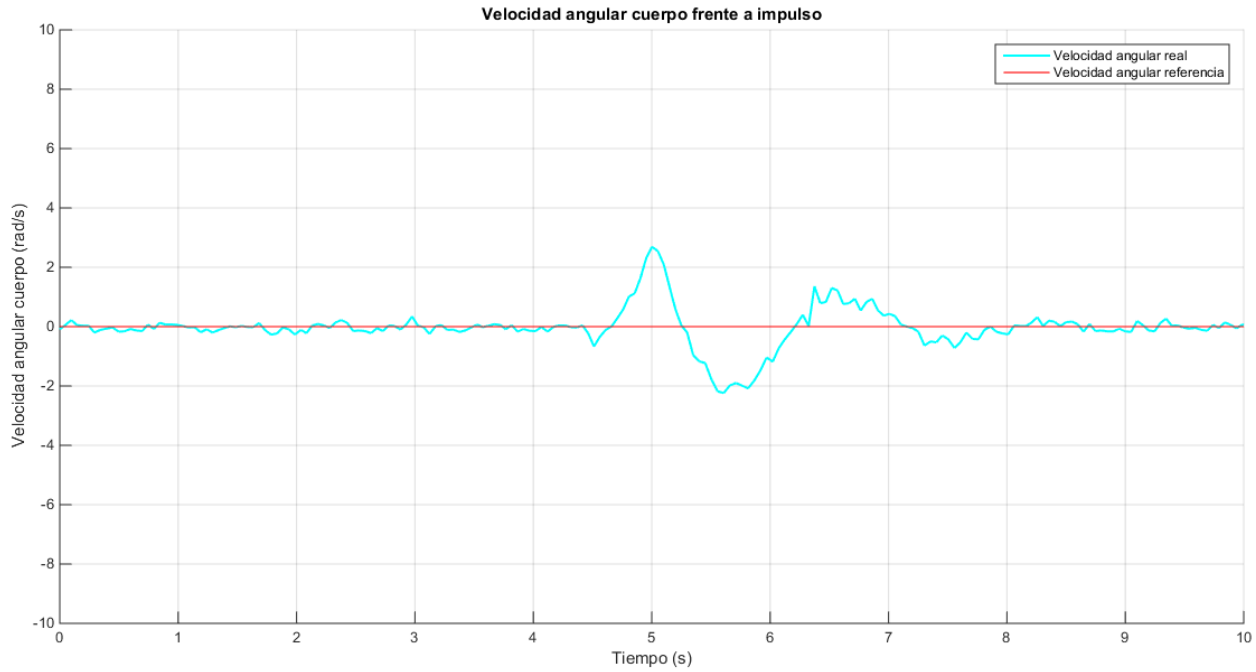

Figura 7-3. Gráfica: Velocidad angular cuerpo frente a impulso

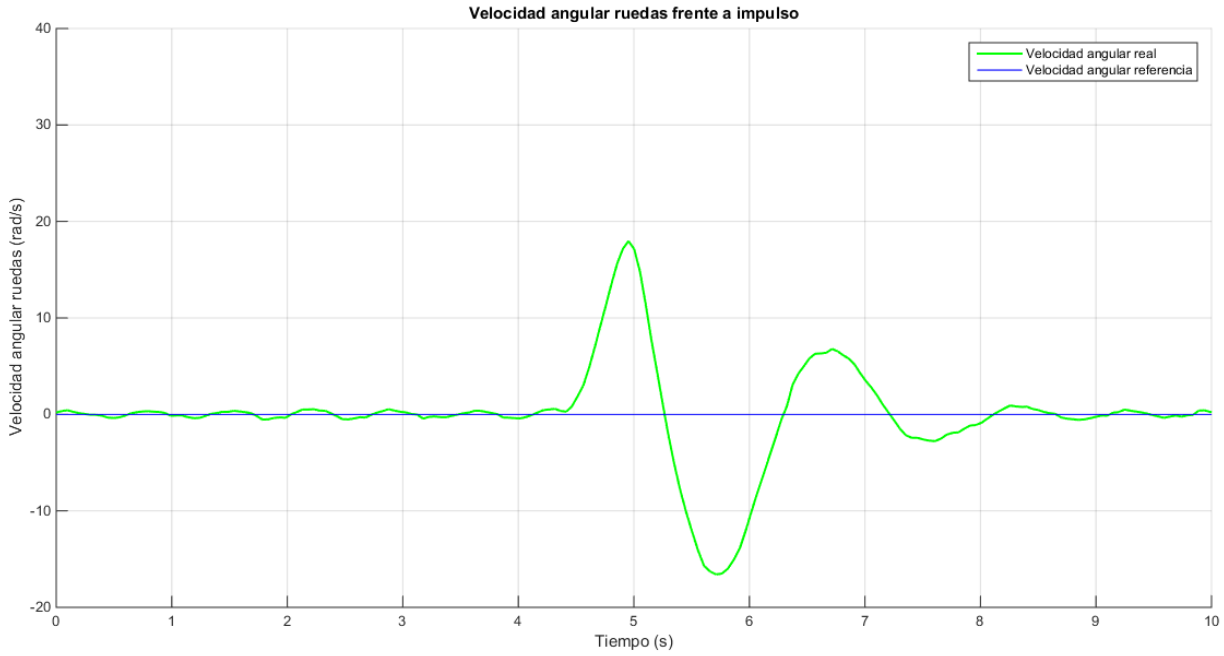

Figura 7-4. Gráfica: Velocidad angular ruedas frente a impulso

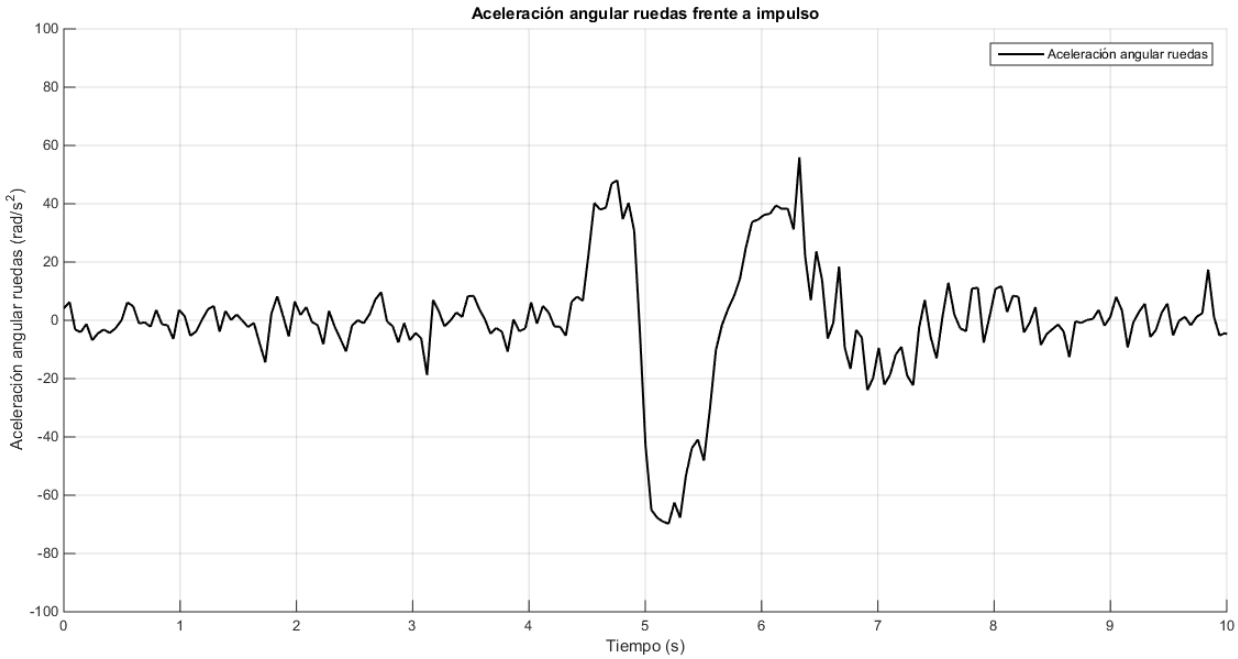

Figura 7-5. Gráfica: Aceleración angular ruedas (acción de control) frente a impulso

7.2 Ensayo ante fuerza creciente en el extremo superior

En esta prueba se empuja cada vez más la parte superior del para desestabilizarlo de forma continuada y creciente.

En la siguiente gráfica se pueden ver las modificaciones que se producen en las variables más relevantes del sistema: inclinación con respecto a la vertical, velocidad angular del cuerpo, velocidad angular de las ruedas y aceleración angular de las ruedas. En este estudio, las referencias de todas estas variables son cero:

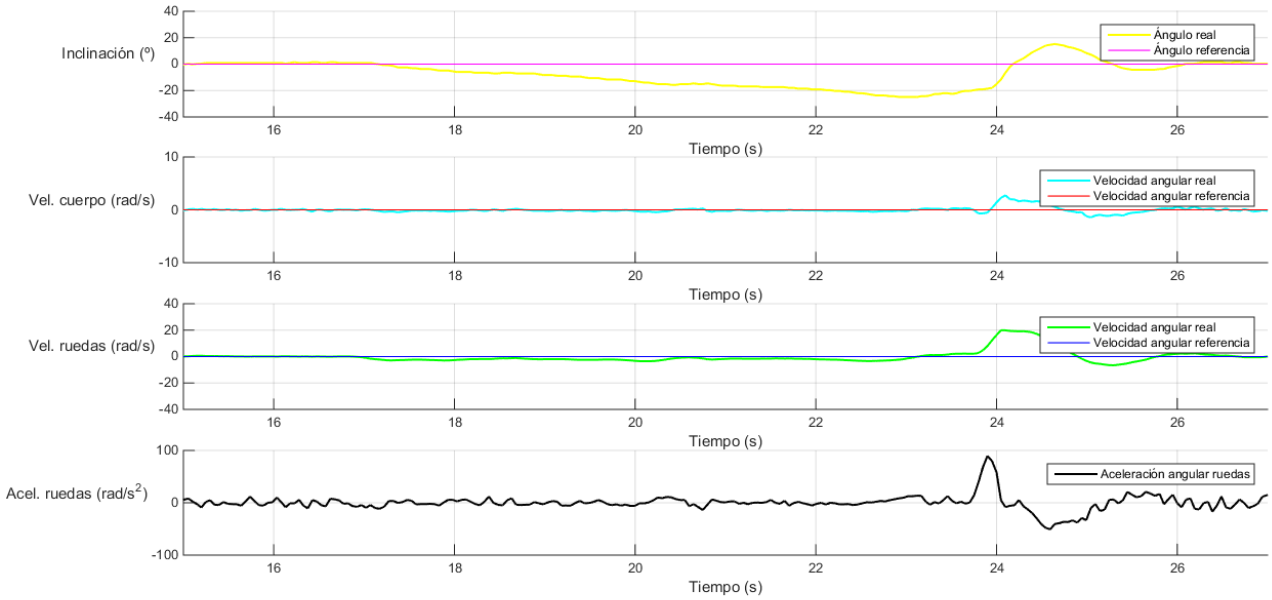

Figura 7-6. Gráfica: Variables frente a fuerza creciente

Como se puede ver, cuando el sistema está estabilizado en torno a su punto de equilibrio se empieza a empujar la parte superior del vehículo. El sistema de control inclina al vehículo en sentido contrario al origen de la fuerza de forma aproximadamente proporcional a esta. Así consigue contrarrestarla y continuar parado.

En el momento que se deja de aplicar la fuerza, el vehículo está altamente desequilabrado, por lo que el controlador aumenta la aceleración de las ruedas para evitar la caída del sistema, con esta lo hace la velocidad de las ruedas también, y el ángulo vuelve a tender a cero.

Se puede ver que se producen algunas sobreosciolaciones, pero el sistema se estabiliza en poco tiempo.

Por mayor claridad, se presenta la misma gráfica, pero separada para cada varible:

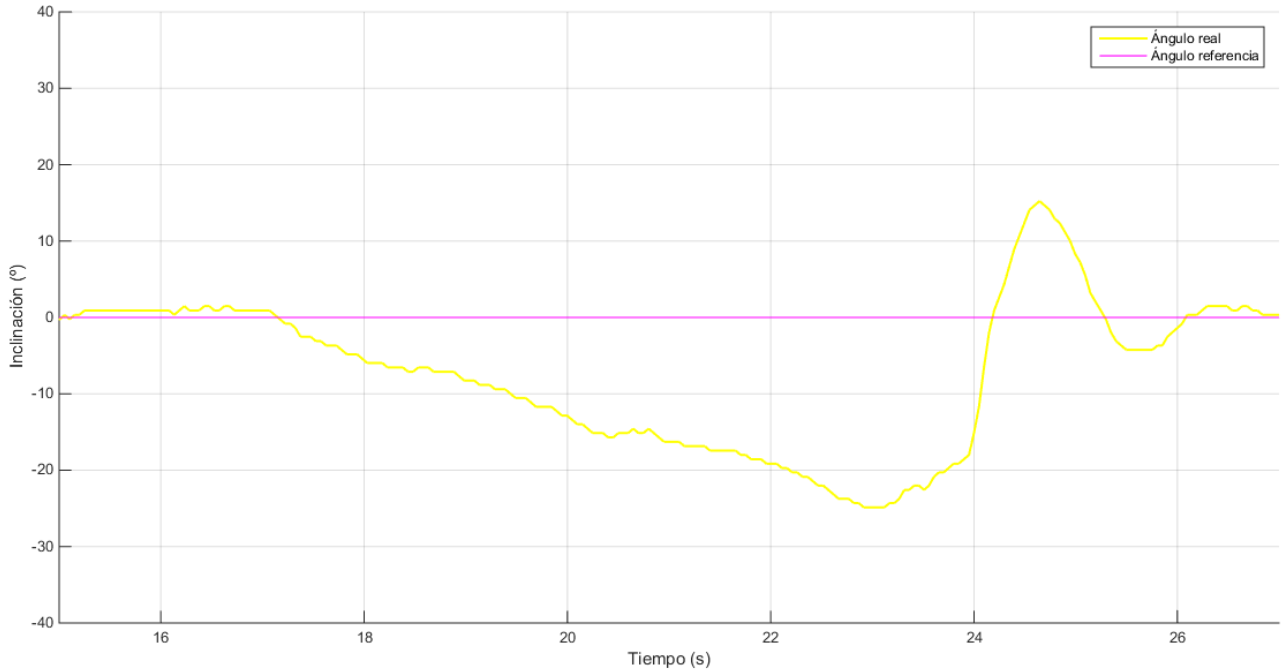

Figura 7-7. Gráfica: Inclinación frente a fuerza creciente

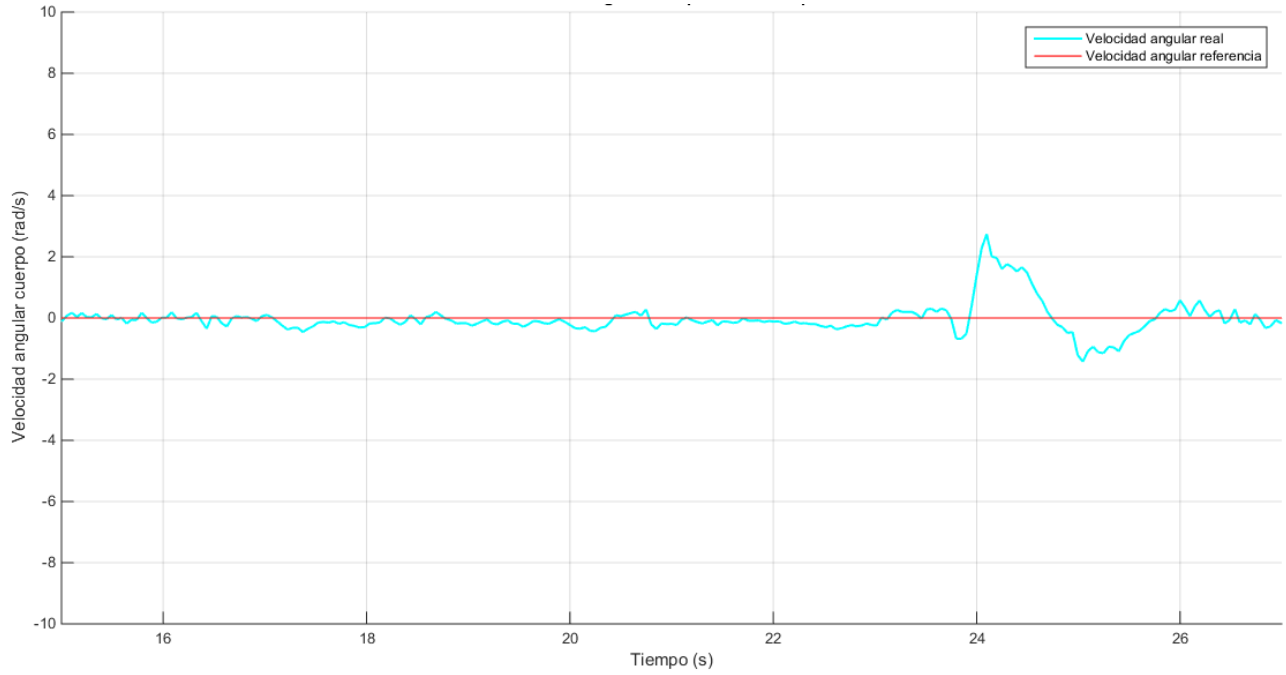

Figura 7-8. Gráfica: Velocidad angular cuerpo frente a fuerza creciente

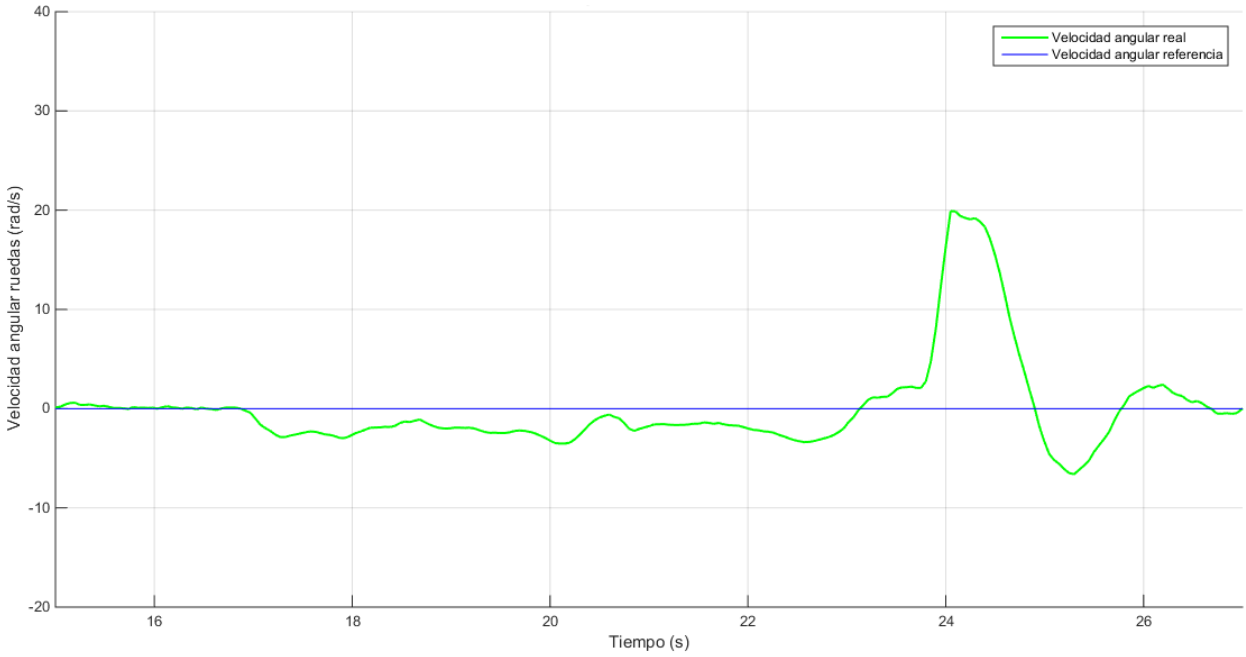

Figura 7-9. Gráfica: Velocidad angular ruedas frente a fuerza creciente

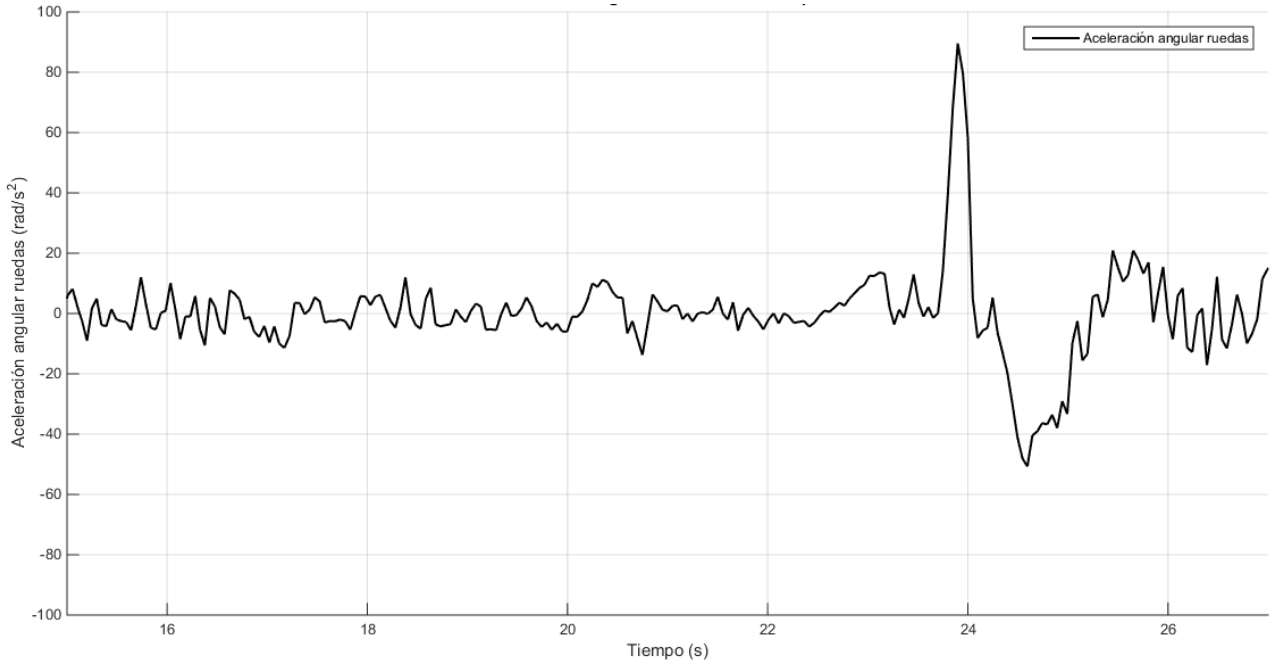

Figura 7-10. Gráfica: Aceleración angular ruedas (acción de control) frente a fuerza creciente

7.3 Ensayo ante carga descentrada

En esta prueba se coloca un objeto encima del vehículo de manera que varía su centro de gravedad.

En la siguiente gráfica se pueden ver las modificaciones que se producen en las variables más relevantes del sistema: inclinación con respecto a la vertical, velocidad angular del cuerpo, velocidad angular de las ruedas y aceleración angular de las ruedas. En este estudio, las referencias de todas estas variables son cero:

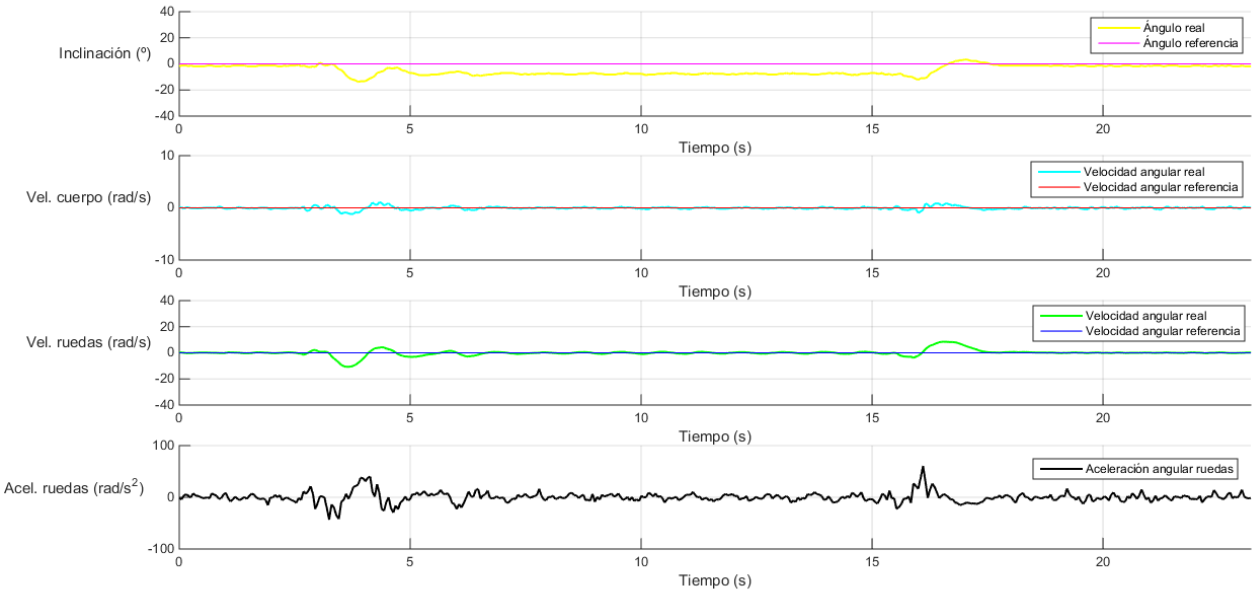

Figura 7-11. Gráfica: Variables frente a carga descentrada

Como se puede ver, cuando el sistema está estabilizado en torno a su punto de equilibrio se coloca una carga que varía el centro de gravedad. El sistema de control inclina al vehículo para contrarrestarla y continuar parado.

En el momento que se deja de aplicar la carga, el vehículo está altamente desequilabrado, por lo que el controlador aumenta la aceleración de las ruedas para evitar la caída del sistema, con esta lo hace la velocidad de las ruedas también, y el ángulo vuelve a tender a cero.

Se puede ver que se producen algunas sobreosciolaciones, pero el sistema se estabiliza en poco tiempo.

Por mayor claridad, se presenta la misma gráfica, pero separada para cada varible:

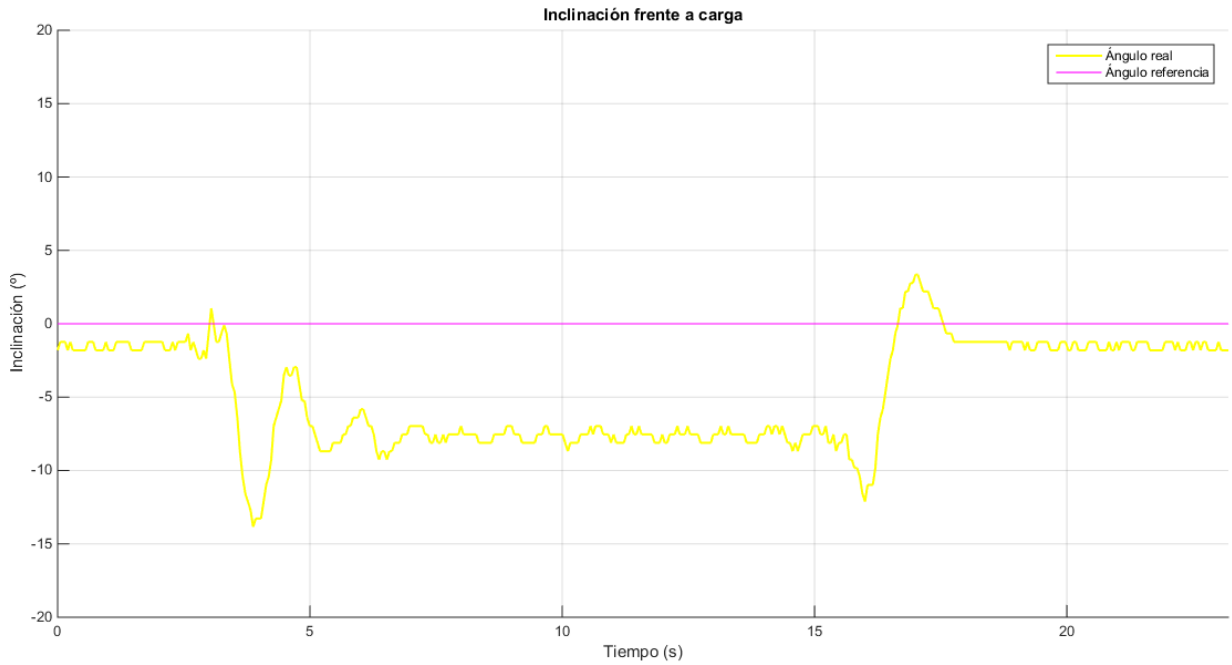

Figura 7-12. Gráfica: Inclinación frente a carga descentrada

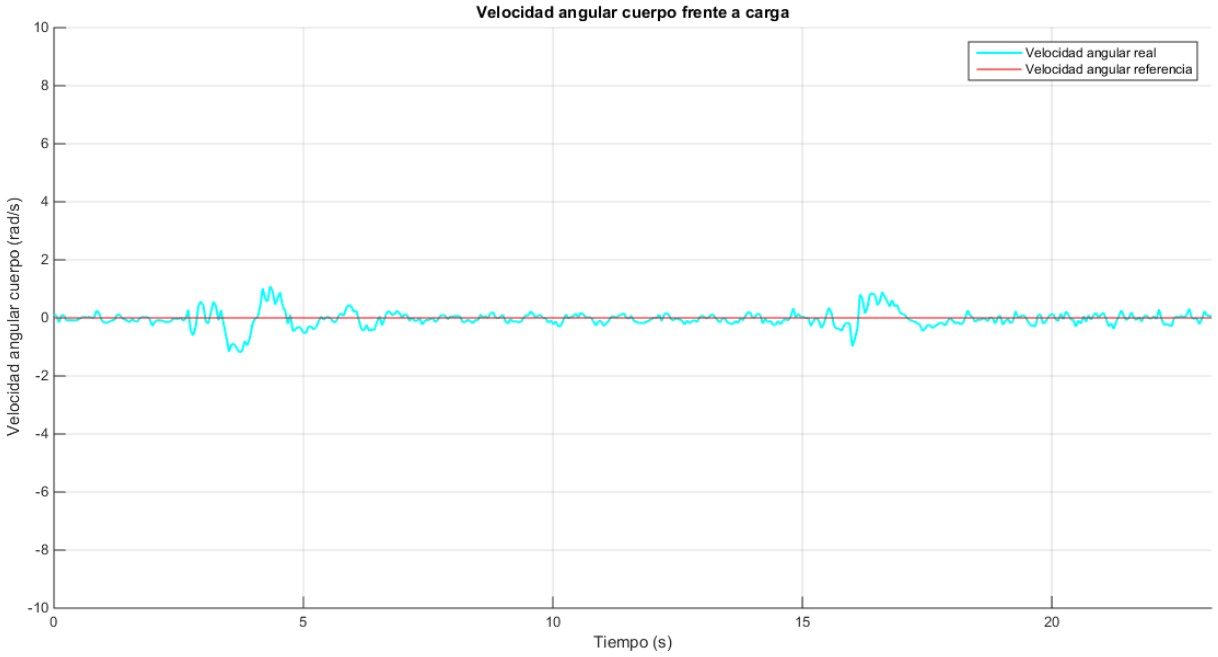

Figura 7-13. Gráfica: Velocidad angular cuerpo frente a carga descentrada

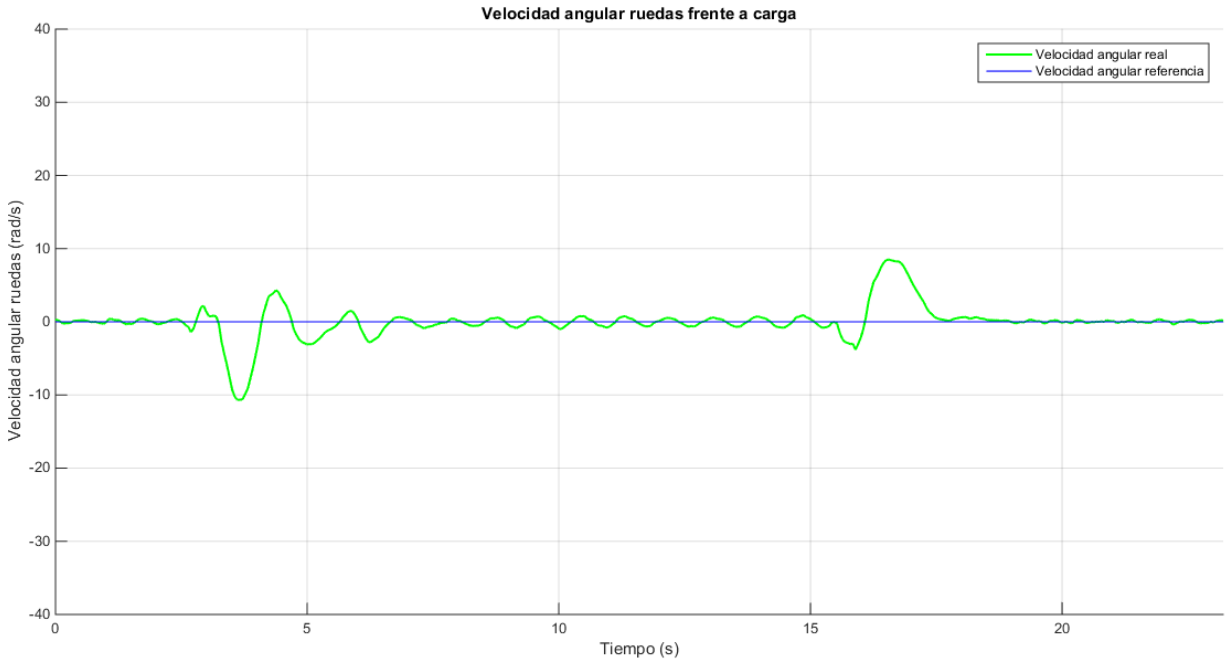

Figura 7-14. Gráfica: Velocidad angular ruedas frente a carga descentrada

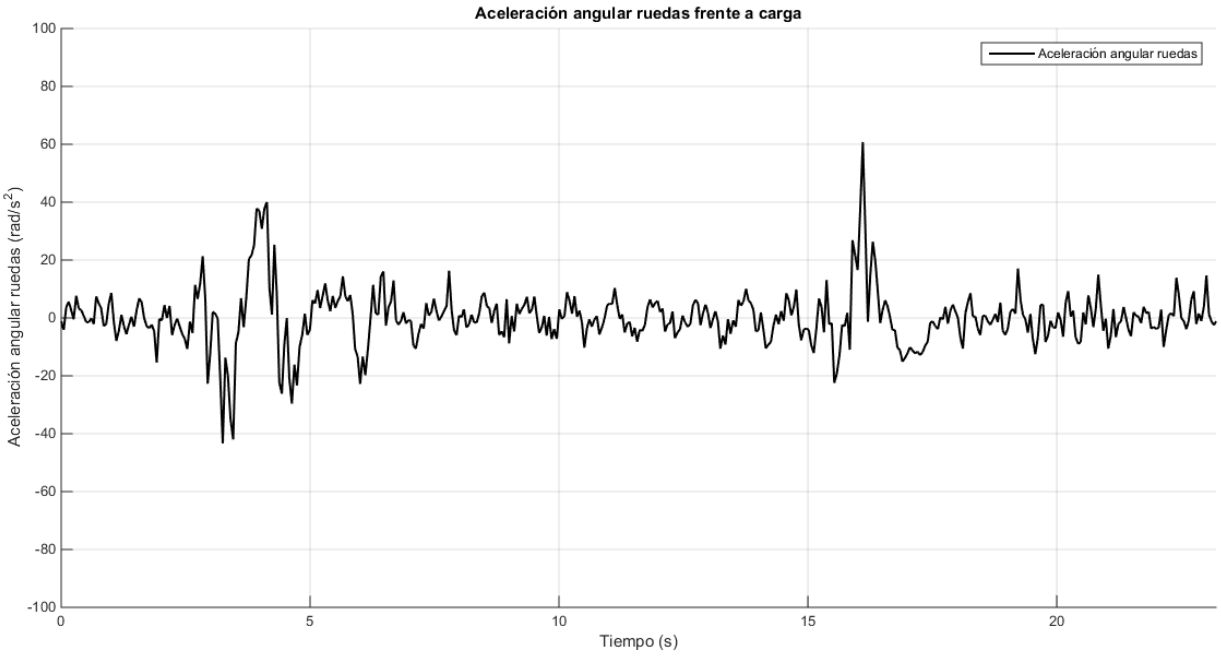

Figura 7-15. Gráfica: Aceleración angular ruedas (acción de control) frente a carga descentrada

7.4 Ensayo ante cambios de referencia en velocidad de las ruedas

Con el controlador *LQR* implementado se permite el desplazamiento del vehículo a una velocidad deseada. Se han realizado pruebas dándole referencias en forma de escalón a velocidades altas. Para ello se ha dejado que se estabilice durante cinco segundos, en los cinco siguientes se le ha indicado que se mueva a alta velocidad en sentido positivo, tras esto se le ha indicado reposo y finalmente se le ha establecido una velocidad alta de referencia, pero en sentido contrario.

Esta prueba se ha realizado también en el simulador, obteniendo resultados similares.

En las siguientes gráficas se pueden ver las modificaciones, tanto del sistema real como del simulado, que se producen en las variables más relevantes del sistema: inclinación con respecto a la vertical, velocidad angular del cuerpo, velocidad angular de las ruedas y aceleración angular de las ruedas.

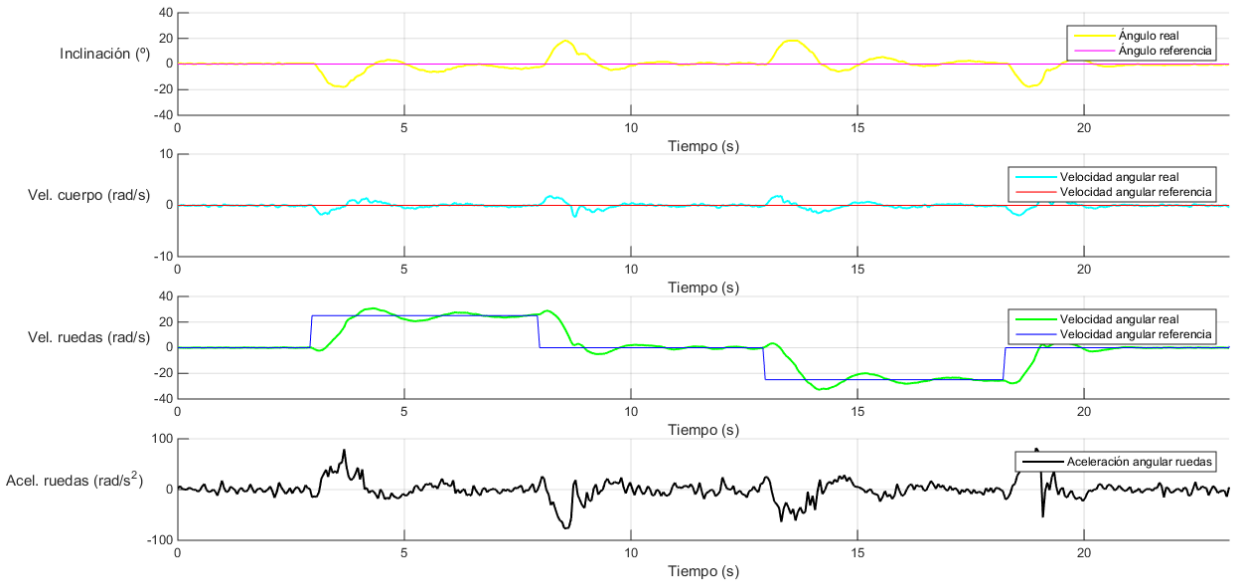

Figura 7-16. Gráfica: Variables frente a referencias en velocidad (reales)

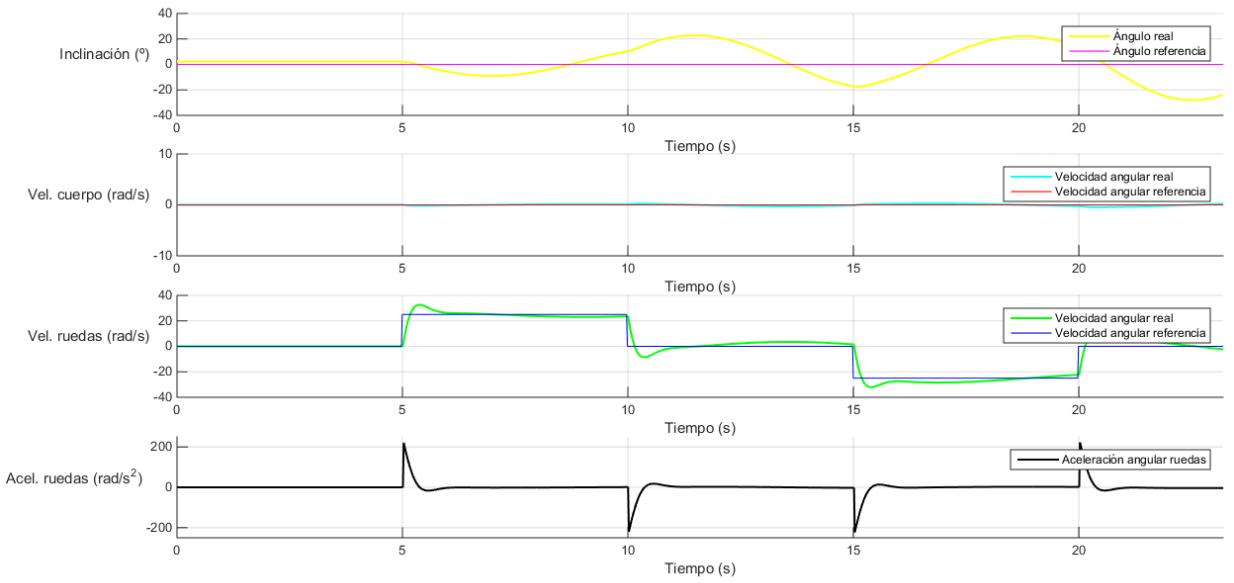

Figura 7-17. Gráfica: Variables frente a referencias en velocidad (simulación)

Como se puede ver, cuando hay un escalón en la referencia la aceleración aumenta o diminuye según corresponda para alcanzarla. Se suelen producir sobreosciolaciones, pero finalmente se estabiliza el sistema.

La principa diferencia entre la simulación y la realidad es las aceleraciones que se alcanza. Las del primer caso duplican a las reales. Esto se debe a que en la realidad la caída del vehículo (inclinación) es mucho mayor que en la simulación, debido a las condiciones no ideales, por lo que la aceleración se ve disminuida para evitar la desestabilización total. En caso contrario, la inclinación llegaría a un extremo en el que el sistema no puede recuperarse.

No obstante, las demás variables tienen un comportamiento bastante similar, tanto de magnitud como de forma.

Por mayor claridad, se presenta la misma gráfica, pero separada para cada varible:

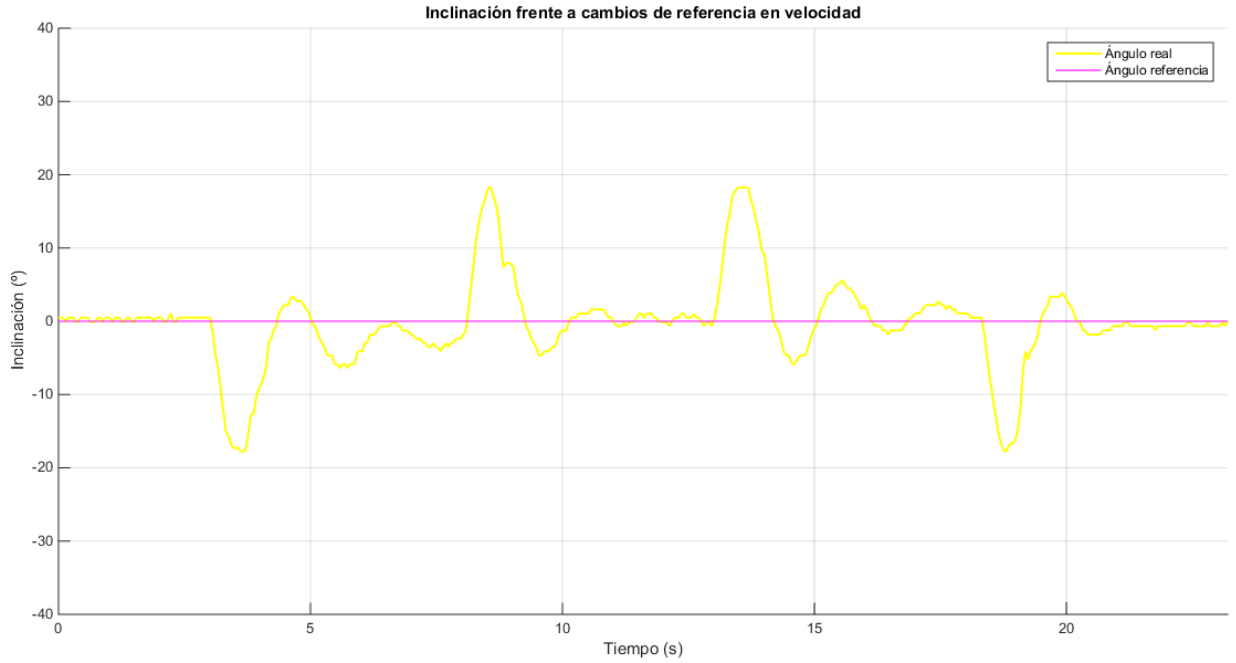

Figura 7-18 Gráfica: Inclinación frente a referencias en velocidad

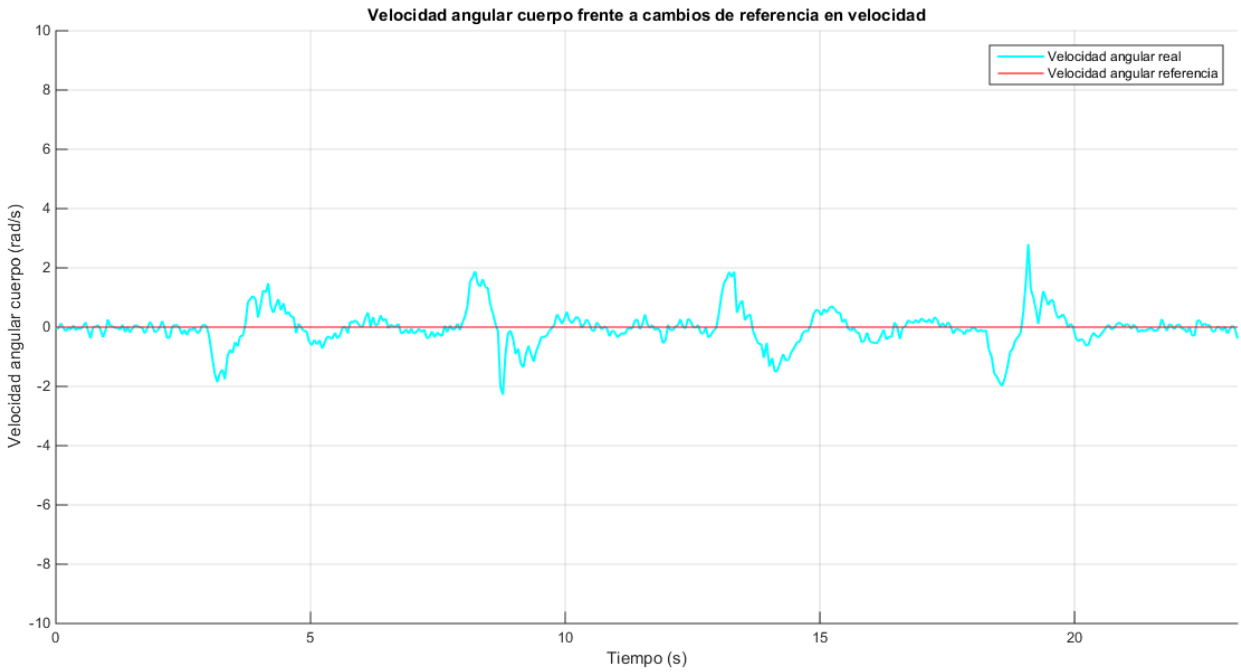

Figura 7-19. Gráfica: Velocidad angular cuerpo frente a referencias en velocidad

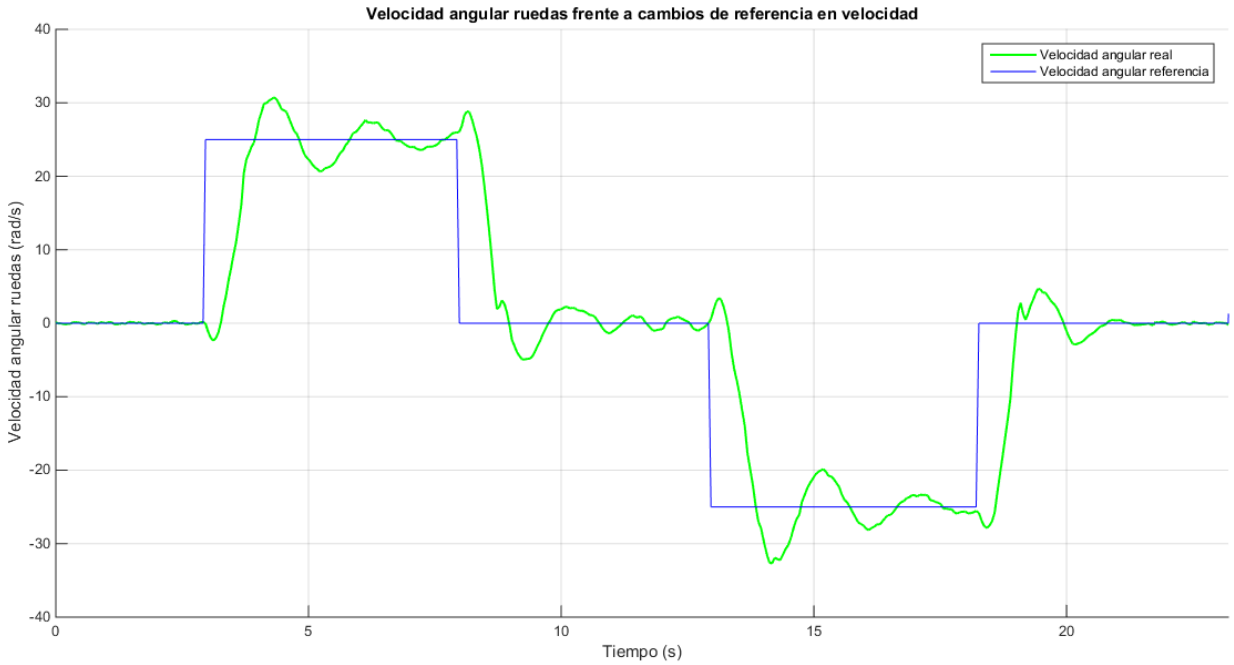

Figura 7-20. Gráfica: Velocidad angular ruedas frente a referencias

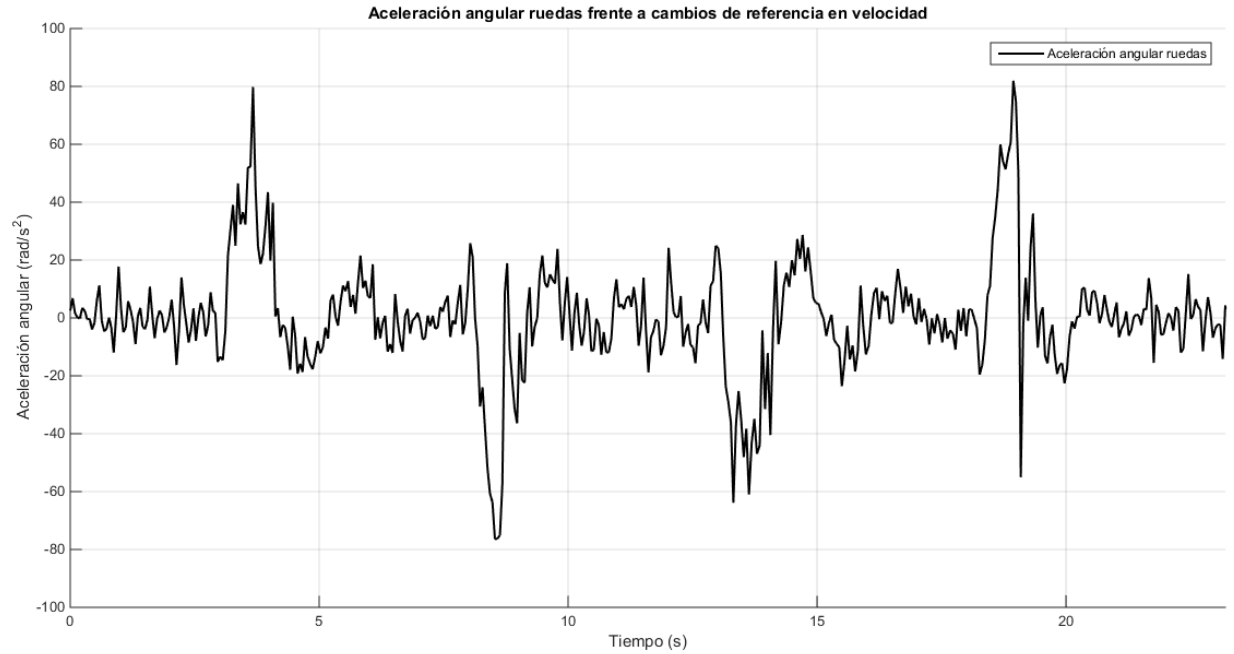

Figura 7-21. Gráfica: Aceleración angular ruedas (acción de control) frente a referencias en velocidad

7.5 Análisis de resultados

En los cuatro ensayos anteriores, se ha podido ver que tanto el ángulo y velocidad angular de inclinación del vehículo, como la velocidad angular de las ruedas, seguían un perfil similar al esperado. Es decir, similar al que proporciona la simulación del sistema en condiciones ideales.

Sin embargo, esto no ocurre con la acción de control (la aceleración de las ruedas). En esta variable se detecta un gran ruido.

El valor de esta variable es calculado por el controlador *LQR*, el cual utiliza las tres variables mencionadas en el primer apartado. Todas son multiplicadas, en el cálculo de la aceleración, por números mayores que uno, por lo que, si en alguna se produce ruido, este se mostrará amplificado en la salida del controlador.

Que el origen sea la velocidad angular de las ruedas puede ser descartado, ya que esta es una salida del sistema (que depende de la propia aceleración), y no se tiene su valor real (no hay realimentación de encóder).

Teniendo lo anterior en cuenta, se puede recurrir a las distintas gráficas de inclinación del cuerpo y la velocidad angular de este para estudiar de donde proviene.

Haciendo esto, se llega a la conclusión de que aproximadamente ambas afectan igual. Esto se debe a que el error del ángulo es diez veces menor que el de la velocidad angular, pero su ganancia es diez veces mayor, por lo que el peso en ambas es el mismo.

Por tanto, el ruido que se produce en la aceleración de las ruedas se debe a la *IMU*, ya que es el instrumento utilizado para calcular la inclinación y velocidad angular del cuerpo. No obstante, este no influye de forma notable al sistema, ya que, si se integra, el valor es aproximadamente nulo, por lo que la velocidad apenas se ve afectada, y el perfil que se desea en esta se consigue sin problemas.

Este último comentario puede ser verificado co el correcto funcionamiento del vehículo cuando está en funcionamiento.

8 POSIBLES MEJORAS

Los resultados de este proyecto han sido satisfatorios. El comportamiento del vehículo ha superado las expectativas. Sin embargo, hay procesos que pueden ser mejorados, principalmente el sistema de control.

El controlador utilizado es *LQR,* sin restricciones. Por lo que el vehículo responde sin tener en cuenta sus límites físicos. Por ejemplo, cuando recibe una fuerza creciente en su parte superior: a mayor sea esta, más se inclina para contrarrestarla sin importa si alcanza el ángulo limite para tener par suficiente con el recuperar su estado normal al desaparecer la fuerza, por lo que pierde el equilibrio.

Si se implementase un controlador con restricciones, el sistema sería mucho más robusto y flexible a agentes externos. No obstante, habría que hacer un estudio del tiempo de procesamiento de este, para ver si el microprocesador utilizado tiene las especificaciones necesarias. De todas formas, en caso negativo, podría utilizarse *Arduino-Intel Galileo*. De forma que el *Arduino UNO* siga implementando las operaciones para conseguir velocidad, aceleración, comunicación por bluetotth, obtención del ángulo, etc., y el procesador *Intel* (que tiene mucha más velocidad de procesamiento) implemente el control a alto nivel.

Otra de las mejoras que se podrían realizar sería crear un perfil de velocidad a partir de la aceleración más suave. Es decir, en vez de actualizar el valor de la velocidad cada 4 ms, hacerlo cada paso del motor.

Ya se ha comprobado que dicha operación necesita demasiado procesamiento como para implementarla cada tan poco tiempo. Pero se podría crear un algoritmo que tenga en cuenta ciertas condiciones para simplificarlo y reducir el tiempo de ejecución. Aunque, no parece que esto pueda suponer una mejora notable, ya que el resultado es bastante positivo.

Anexo 1: Componentes Proyecto Previo

Los materiales con los que dichos proyectos fueron realizados, incluyéndolos con su derecripción literalmente del propio trabajo de Cecilia González, son:

Chasis de aluminio y madera

Es la estructura sobre la que están montados todos los dispositivos. Consiste en un marco de aluminio con una tapa de aluminio que forma un compartimento para proteger la electrónica. En este compartimento se sitúan la placa microcontroladora, el módulo Bluetooth, las placas drivers de los motores y la unidad de medidas inerciales (IMU). En la parte inferior junto a las ruedas están instalados los motores en el interior del marco de aluminio. Para rigidizar la estructura se atornillan dos primas rectangulares de madera al chasis de aluminio. Así mismo para proteger de posibles caídas, el vehículo posee una placa de metacrilato transparente con bordes de gomaespuma (para amortiguar caídas). La batería queda dispuesta en esta placa.

Motores EMG-30 y encoders

El vehículo consta de dos motores de corriente continua dispuestos en el chasis de aluminio. Su característica principal es que tienen una tensión nominal de 12 V. Las características principales se muestran en la Tabla 2-1.

Tabla 2–8-1 Principales características del motor

Característica	Medida (unidades)
Tensión nominal	12 V
Par nominal	1.5 kg/cm
Velocidad nominal	170 rpm
Intensidad nominal	530 mA
Velocidad en vacío	216 rpm
Intensidad en vacío	150 mA
Intensidad de frenado	2.5 A
Potencia de salida nominal	4.22 W

Una característica imprescindible para el correcto uso del motor y del control del mismo es la zona muerta. Se considera zona muerta la región donde varía la entrada, es decir la tensión, sin obtener ningún cambio en la salida, que será la velocidad angular. Se han considerado dos tipos de fricción que provocan la existencia de esta zona, la fricción estática y la dinámica. Estas fricciones afectan a la relación entre la tensión aplicada al motor y la velocidad resultante.
Los ensayos realizados anteriormente no muestran los valores de tensión, sino el valor PWM. PWM son las siglas de Pulse Width Modulation, es decir modulación por anchura de pulsos. Consiste en una señal cuadrada de frecuencia constante, donde la anchura de pulso de esta señal definirá la tensión eficaz. Es necesario el

PWM porque un microcontrolador no puede transmitir una tensión variable pero sí puede emitir una señal PWM. Las diferentes de anchuras de pulso provocarán la diferencia de tensión que estará en un rango entre 0 y 5 V con un rango de variación de 0 a 255 para el PWM. Esta señal no va directamente al motor ya que necesita mayor voltaje. Esta señal servirá para la placa controladora.

En la Tabla 2-2 se observan los valores obtenidos en el ensayo de vacío, que es el usado para la implementación del código de control del sistema.

Tabla 2–8-2 Característica de la zona muerta

Motor	PWM límite positivo	PWM límite negativa
Motor izquierdo	17	18
Motor derecho	16	17

Se ha realizado experimentos donde se ha comprobado que los motores no son completamente lineales. Sin embargo, para simplificar el sistema se consideran lineales.

Los motores contienen una caja reductora de 30:1 y encoders de cuadratura (dos sensores separados 90°) con sensores de tipo Efecto Hall. Cada sensor magnético posee un canal de señal por lo que habrá dos canales, A y B, según el valor alto o bajo de los mismos se sabrá si el sentido es horario o antihorario. En la Tabla 2-3 se puede obsevar el sentido de giro según la entrada de estos dos canales.

Tabla 2–8-3 Tabla lógica del encoder

A:B actual \ A:B anterior	0:0	0:1	1:0	1:1
0:0		Antihorario	Horario	
0:1	Horario			Antihorario
1:0	Antihorario			Horario
1:1		Antihorario	Horario	

En el eje de abcisas de la Tabla 2-3 están los valores anteriores de los canales A y B y en el eje de ordenadas se encuentran los valores actuales, es decir para saber el sentido de giro se necesita tanto el valor anterior como el actual del sensor.

La alimentación se necesita para los motores y también para los enconders. Además son necesarios dos cables de salida para los valores de cada sensor, uno para cada canal.

Una de las ventajas de los sensores Hall es que no se ven afectados por la temperatura y tampoco necesitan un gran mantenimiento de limpieza.

Ruedas

Las ruedas utilizadas son Wheel100 tienen 100 mm de diámetro y una banda de rodadura de 26 mm de anchura de goma. La masa de la rueda es de 0.140 kg. La rueda se encuentra fijada al eje del motor.

Batería de litio

La batería que usa este vehículo es de la marca AH1-Tech de tres celdas, de capacidad 1300 mAh y tensión 12 V. Es una batería Li-Po, es decir, polímero de litio. Es imprescindible su uso ya que los motores necesitan una corriente para alimentarse mucho más alta que la ofrecida por la placa microcontroladora, además de que es esta batería la que concede la autonomía del vehículo por lo que los demás dispositivos no necesitan ser alimentados por la placa microcontroladora para funcionar.

Placa microcontroladora: Arduino Mega 2560

La placa de Arduino permite tomar información del entorno a través de entradas y calcular unas salidas, siendo capaz de controlar otros dispositivos. Este microcontrolador se programa mediante el lenguaje de programación Arduino (basado en Wiring) y el entorno de desarrollo Arduino (basado en Processing). Además el entorno de desarrollo es gratuito.

La placa microcontroladora se encarga de recibir datos de los sensores y de realizar los cálculos necesarios para controlar el sistema, por lo que es capaz de calcular las entradas a los dispositivos actuadores como es el motor. Se logra así conseguir el equilibrio del sistema además de que se podrá cumplir la trayectoria que exija el usuario.

La placa dispone de un conector USB con el que poder cargar el código del programa desde el ordenador a la placa. Esta unidad se puede alimentar tanto por el conector jack como por el conector USB.

La unidad controladora Arduino Mega 2560 está basada en el microcontrolador ATmega2560. Tiene 54 pines de entrada/salida digital, entre estos pines pueden utilizarse 15 para salidas PWM. Consta también de 16 entradas analógicas, 4 UARTs (puertos serie hardware), 5 pines con comunicación SPI, 2 pines con comunicación TWI (I2C), un cristal oscilador de 16 MHz (frecuencia del reloj interno de la placa, usado en interrupciones temporales), una conexión USB, un conector jack de alimentación, un conector ICSP y un botón de reset.

Tabla 2–8-4 Principales características Arduino Mega 2560

Característica	Medida (unidades)
Microcontrolador	ATmega2560
Tensión nominal	5 V
Tensión de entrada (uso normal)	7-12 V
Tensión de entrada (límites)	6-20 V
Pines de entrada/salida digital	54 pines (15 para PWM)
Pines de entrada analógica (incorporan convertidores ADC)	16 pines
Corriente continua por pin E/S	40 mA
Corriente continua en el Pin 3V3	50 mA
Memoria flash	256 KB
SRAM	8 KB
EEPROM	4 KB
Frecuencia de reloj	16 MHz
Rango de temperatura	-40°/85
UARTs	4 pines
SPI	5 pines
TWI (I2C)	2 pines

Temporizadores	6 Timers

Los pines digitales se alimentan a 5 V, distinguimos estos pines según su función que se detalla a continuación junto con el número del pin que tiene dicha función.

Pines serial: Serial 0: 0 (RX) y 1 (TX); Serial 1: 19 (RX) y 18 (TX); Serial 2: 17 (RX) y 16 (TX); Serial 3: 15 (RX) y 14 (TX). RX es un pin usado para recibir datos serie de tipo TTL y TX para transmitirlos.

Pines para interrupciones externas: Pin 2 (Interrupción 0), Pin 3 (Interrupción 1), 18 (Interrupción 5), Pin 19 (Interrupción 4) y Pin 21 (Interrupción 2). Estos pines disparan una interrupción en un valor bajo, en un flanco de subida o bajada, o en un cambio de valor.

Pines para PWM: con la función *analogWrite()* se obtiene una salida PWM de 8-bit. Los pines son del 2 al 13 y del 44 al 46.

Pines para comunicación SPI: son los pines 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS). Estos pines también están desviados a la cabecera ICSP.

LED: existe un LED en la placa conectado en el pin 13. Si el valor es alto el LED estará encendido por el contrario si el valor es bajo, el LED se apagará.

Pines para comunicación TWI: esta comunicación la tienen los pines 20 (SDA) y 21 (SCL).

Entre estos pines, en el proyecto se han utilizado los siguientes:

- Pines PWM (usados para los motores): 7 y 9.
- Pines de interrupciones externas (usados en los encoders): 2, 3, 18 y 19.
- Pines de comunicación I2C: 20 y 21.
- Pines de comunicación por puerto serie (usados para Bluetooth): 0 y 1.
- Pines de alimentación auxiliar: 3V3, 5V y GND.
- Pines de salidas digitales (para el motor): 40, 41, 42, 43.

Se usarán más pines ya que se implementa el uso de dos temporizadores. Su función se explicará detenidamente en el capítulo 3 que trata las mejoras del funcionamiento en el apartado de interrupciones temporales.

Unidad de medidas inerciales: IMU

Para la obtención de la inclinación y de la velocidad angular, se usa una IMU, capaz de integrar tres acelerómetros y tres giróscopos. Para este proyecto se utilizan únicamente dos acelerómetros y un giróscopo, ya que se puede considerar el sistema como un péndulo 2D. Esto quiere decir que la información que se necesita es el ángulo de inclinación de un único plano del espacio y la velocidad angular del vehículo (que tendrá dos componentes).

El dispositivo IMU del que consta el vehículo es la unidad MPU6050 del fabricante InvenSense. Este dispositivo irá conectado directamente a la placa de Arduino donde se procesarán los datos obtenidos por la IMU. Esta unidad dispone también de un puerto auxiliar I2C.

A continuación, en la Tabla 2-5 se muestran las principales características de los acelerómetros.

Tabla 2–8-5 Principales características de los acelerómetros

Característica	Medida (unidades)
Salida digital con rango programable	±2g, ±4g, ±8g, ±16g
Convertidores ADCs	16-bit
Intensidad nominal	500 μA
Intensidad en baja potencia en función de la frecuencia	10 μA a 1.25 Hz
	20 μA a 5 Hz
	60 μA a 20 Hz
	110 μA a 40 Hz

Entre otras de las características se encuentran la detección de orientación y señalamiente, la detección de golpeo, interrupciones programables por el usuario, interrupción de caída libre, de alta aceleración y de reposo. También contiene un Auto test para el usuario.

En este proyecto se necesita que la IMU trabaje con la mayor sensibilidad, es decir con una magnitud mínima ya se obtienen cambios en la salida. La mayor sensibilidad se consigue con el menor rango de aceleraciones posible, en este caso ±2g. Esto se configurará en el código.

Por otra parte, se tendrán en cuenta también las características de los giróscopos que aparecen en la Tabla 2-6.

Tabla 2–8-6 Principales características de los giróscopos

Característica	Medida (unidades)
Salida digital con rango programable	±250°/s, ±500°/s, ±1000°/s, ±2000°/s
Convertidores ADCs	16-bit
Intensidad nominal	3.6 mA
Intensidad de stanby	5 μA

Entre otras características se encuentran también una señal externa de tipo sync conectada al pin FSYNC que soporta la sincronización de imagen, vídeo y GPS. Además posee una actuación mejorada frente a ruidos de baja frecuencia, un filtro paso-bajo digitalmente programable y un Auto test para el usuario.

En los giróscopos sucede lo mismo que en los acelerómetros, se observa una mayor sensibilidad para el menor rango de velocidades angulares medibles. Una vez más para configurarlo, se deben configurar las direcciones del registro en el código según esta especificación.

Otra de las características importantes de este dispositivo es la alimentación que permite. La unidad MPU6050 trabaja con un rango 2.375-3.45 V, por lo tanto la alimentación será a través del Pin 3V3 de Arduino.

Dos de los pines más significativos son los de la comunicación I2C, el pin SCL y SDA conectados a la placa de Arduino en los pines 20 y 21 respectivamente.

Tarjeta controladora de Motores

La controladora usada ha sido TB6612FNG de Toshiba. El Driver de Motores está compuesto por transistores MOSFET dentro del circuito integrado que permiten su uso como amplificador. Esta función es fundamental para así poder alimentar a los motores con la tensión que necesitan, que la placa microcontroladora no es capaz de ofrecer. La disposición de los transistores y diodos tendrá un montaje de Puente H. Esta disposición permite

todas las combinaciones de movimiento del motor tanto aceleración como frenada además de cambio de sentido de giro y se utiliza una placa por cada motor.

La función de amplificación de señal permite aumentar de forma proporcional la señal PWM obtenida y adecuarla a la necesidad del motor. Además permite que el motor funcione a una velocidad variable. El rango de la salida de la controladora está entre 0 y 12 V, según el valor de entrada de la señal PWM. En la Tabla 2-7 aparecen las características principales de este dispositivo.

Tabla 2–8-7 Principales características Driver de Motores

Característica	Medida (unidades)
Canales de motores	2
Mínima tensión operativa	4.5 V
Máxima tensión operativa	13.5 V
Corriente continuada de salida por canal	1 A
Corriente pico de salida por canal	3 A
Corriente continuada de salida en paralelo	2 A
Frecuencia máxima PWM	100 kHz
Mínima tensión lógica	2.7 V
Máxima tensión lógica	5.5 V
Protección frente a tensión reversa	Sí

Módulo Bluetooth

Se adquiere un módulo Bluetooth HC-06, este dispositivo se comunica con la placa microcontroladora a través de un puerto serie y permite trabajar en un amplio rango de velocidades de transferencia que van desde 1200 baudios hasta 1382400 baudios.

Este módulo será usado para obtener datos del vehículo sin necesidad de conectar con un cable el ordenador y el segway, es decir, proporciona comunicación inalámbrica. Esto proporciona la opción de obtener los datos en tiempo real, siendo mucho más fácil y rápido el estudio del ajuste del funcionamiento del vehículo.

Este módulo Bluetooth se puede configurar, como se muestra en el código del apéndice, gracias a su memoria interna. El nombre que fue escogido para el dispositivo es MINISEGWAY y la contraseña 0000. La velocidad de transferencia a la que trabaja es 115000 baudios.
Por último, queda añadir que este dispositivo necesita 3 V de alimentación.

Placa de circuito impreso de tipo Shield

Esta placa de circuito impreso se usa para reducir el cableado. Se diseñó para las conexiones para la IMU, se incluyen también las controladoras de los motores y un zócalo p que conecta el módulo Bluetooth.
Esta placa es de tipo Shield, es decir, se conecta directamente sobre la placa microcontroladora de Arduino.

Se añade un interruptor para conectar o desconectar el módulo Bluetooth permientiendo así cargar los programas vía USB.

Anexo 2: Código

```
#include <SoftwareSerial.h>
#include <Wire.h>
#include "MPU6050.h"
#include "I2Cdev.h"


//////////////////////INICIO ACELERACIÓN DE MOTORES //////////////////////
#define pi 3.1415926535
int i;

//EN VECTORES: 0 = 1MP; 1 = 2MP; 2 = 4MP; 3 = 8MP;

//TIMERS
#define PRESCALA 8                                          //Prescalado del
timer de motores
//Periodos
uint16_t npuls_min[] = {700, 600, 600, 600};            //Menor número
posible del contador del timer para dar un paso. Nº mínimo de pulsos/paso. 1-
2-4-8 MP
uint16_t npuls_max[] = {65535, 65535, 65535, 65535};    //Timer 16 bits.
Mayor número posible del contador del timer para dar un paso. Nº máximo de
pulsos/paso. 1-2-4-8 MP
float T_PULS = 0.0000000625*PRESCALA;                   //Periodo de
incremento del contador del timer (Prescalado*1/16000000 segundos)

//Frecuencias mínima y máxima
float f_timer_min[] = {1/(npuls_max[0]*T_PULS),
                       1/(npuls_max[1]*T_PULS),
                       1/(npuls_max[2]*T_PULS),
                       1/(npuls_max[3]*T_PULS)}; //Nº mínimo de pasos/segundo
float f_timer_max[] = {1/(npuls_min[0]*T_PULS),
                       1/(npuls_min[1]*T_PULS),
                       1/(npuls_min[2]*T_PULS),
                       1/(npuls_min[3]*T_PULS)}; //Nº mínimo de pasos/segundo

//MOTORES
//Pasos para 1 vuelta
int pasosxvuelta[] = {200, 400, 800, 1600};      //nº de pasos para dar una
vuelta completa. 1,2,4,8 MP
// Frecuencias máxima y mínimas
float f_motor_max[] = {f_timer_max[0]/pasosxvuelta[0],
                       f_timer_max[1]/pasosxvuelta[1],
                       f_timer_max[2]/pasosxvuelta[2],
```

```
                        f_timer_max[3]/pasosxvuelta[3]};   //nº máximo de
vueltas/segundo. 1-2-4-8 MP
float f_motor_min[] = {f_timer_min[0]/pasosxvuelta[0],
                        f_timer_min[1]/pasosxvuelta[1],
                        f_timer_min[2]/pasosxvuelta[2],
                        f_timer_min[3]/pasosxvuelta[3]};   //nº mínimo de
vueltas/segundo. 1-2-4-8 MP

//Velocidades angulares. Micropasos 1/1; Micropasos 2/1; Micropasos 4/1;
Micropasos 8/1;
float omega_motor_max[] = {2*pi*f_motor_max[0],
                           2*pi*f_motor_max[1],
                           2*pi*f_motor_max[2],
                           2*pi*f_motor_max[3]};           //nº máximo de
radianes/segundo. 1-2-4-8 MP
float omega_motor_min[] = {2*pi*f_motor_min[0],
                           2*pi*f_motor_min[1],
                           2*pi*f_motor_min[2],
                           2*pi*f_motor_min[3]};           //nº máximo de
radianes/segundo. 1-2-4-8 MP

//
uint16_t npuls_A = npuls_max[3], npuls_B = npuls_max[3]; //Nº de pulsos del
timer que determinan la frecuencia.
float alfa_A = 0, alfa_B = 0;      //Aceleración necesaria que calculamos en
el control
float omega_A = 0, omega_B = 0;    //Velocidad angular de los motores
//Direcciones de pines de los drivers para los motores
int dir_pin_A = 5, dir_pin_B = 6, step_pin_A = 4, step_pin_B = 7 //Direccion
de pines DIR y STEP (PORTD: 5, 6, 4, 7)
int ms1_pin_A = 8 - 8 , ms2_pin_A = 9 - 8, ms1_pin_B = 12 - 8, ms2_pin_B = 13
- 8;  //Direccion de micropasos MS1 y MS2 (PORTB: 4, 5, 0, 1). Para dejarlo
en función de PORTB: "- 8"
//Direcciones de giro de los motores
int dir_A = 0, dir_B = 0; //Sentido de giro de los motores: "1"-> Positivo,
"0"-> Parado, "-1"->Negativo
//Constantes de conversión (Para ahorra cálculos en el proceso). Micropasos
1/1; Micropasos 2/1; Micropasos 4/1; Micropasos 8/1;
float omega2npuls[] =
{omega_motor_max[0]/(T_PULS*f_motor_max[0]*pasosxvuelta[0]),
 omega_motor_max[1]/(T_PULS*f_motor_max[1]*pasosxvuelta[1]),
 omega_motor_max[2]/(T_PULS*f_motor_max[2]*pasosxvuelta[2]),
 omega_motor_max[3]/(T_PULS*f_motor_max[3]*pasosxvuelta[3])}; //La usamos
para pasar de rad/s a pulsos de los timers.  1-2-4-8 MP
float giro = 0; //Variable donde se introduce la magnitud de giro del
vehículo

//
#define T_PULS_ACEL 0.004096 //Periodo para actualizar velocidad e
implementar aceleración. PS 256
#define T_PULS_LQR  0.01632 //Periodo para calcular aceleración con LQR. PS
256

//#define T_PULS_ACEL 0.00102 //Periodo para actualizar velocidad e
```

```
implmentar aceleración. PS 64
//#define T_PULS_LQR  0.00408 //Periodo para calcular aceleración con LQR PS
64

////////////// FIN ACELERACIÓN DE MOTORES ////////////////////////////////

//////////////////INICIO OBTENCIÓN ÁNGULO ////////////////////////////////
#define MPU 0x68
MPU6050 accelgyro;
int16_t ax, ay, az, temp, gx, gy, gz;      //Valores "crudos" que leemos de
la MPU6050
double ary, gsx;             //Valores anteriores que escalamos
float EscGiro = 1/131;               //Escala del giroscopio. Ponemos 1/131
en vez de 131 para poder multiplicar en vez de dividir.
float EscAcel = 180/3.1415926535;    //Para pasar de radianes a grados
multiplicando.
double angulo = 0; //

//////////////////////FIN OBTENCIÓN ÁNGULO ////////////////////////////////

////////////////////// INICIO OBTENCIÓN LQR //////////////////////
int contLQR = 0;
float K[] = {-570,  -41.0715,   10, -0.3}, c = 2, ivr = 0; //variables para
las constantes del LQR
float xg, xt, dir;
float u;
float phi_r,dphi_r,dtheta_r;
float phi, dphi,dtheta;
///////////////// FIN OBTENCIÓN LQR ////////////////////////////////////


////////////////////////// GRÁFICAS ////////////////////////////////////////7
  int tiempoInicial;
//////////////////// FIN GRÁFICAS ////////////////////////////////


void setup() {

  //INICIALIZACIÓN PINES MOTORES
  pinMode(dir_pin_A, OUTPUT);
  pinMode(step_pin_A, OUTPUT);
  pinMode(dir_pin_B, OUTPUT);
  pinMode(step_pin_B, OUTPUT);
  pinMode(ms1_pin_A + 8, OUTPUT); //Sumamos 8 para ponerlo en función de
valores generales, no PORTB
  pinMode(ms2_pin_A + 8, OUTPUT); //Sumamos 8 para ponerlo en función de
valores generales, no PORTB
  pinMode(ms1_pin_B + 8, OUTPUT); //Sumamos 8 para ponerlo en función de
valores generales, no PORTB
  pinMode(ms2_pin_B + 8, OUTPUT); //Sumamos 8 para ponerlo en función de
valores generales, no PORTB

  //INICIALIZACIÓN PUERTOS SERIES Y COMUNICACION I2C
  Serial.begin(9600); //Frecuencia del puerto serie
  //¡¡¡¡¡¡¡¡¡¡¡¡¡¡!!!!!!!!!!!!!!!!!!!!//
```

```
  //PROGRAMACIÓN DE LOS TIMERS
  //3 interrupciones. Siempre llega a OverFlow.
  //Timer 0. Lo usaremos para calcular el ángulo
  //Timer 1. Envía pasos a los motores
  //TImer 2. Actualiza la velocidad(vel_a,acel) cada OVF y ejecuta el LQR
cada 4 OVF
  TCCR1A = 0; //Inicializamos los registros de control. Modo Normal
  TCCR1B = 0; //Inicializamos los registros de control. Modo Normal
  switch(PRESCALA){
    case 1:
      TCCR1B |= (1 << CS10);            //Prescalado a 1
    break;
    case 8:
      TCCR1B |= (1 << CS11);            //Prescalado a 8
    break;
    case 64:
      TCCR1B |= (1 << CS11)|(1 << CS10); //Prescalado a 64
    break;
    case 256:
      TCCR1B |= (1 << CS12);            //Prescalado a 256
    break;
    case 1024:
      TCCR1B |= (1 << CS12)|(1 << CS10); //Prescalado a 1024
    break;
  };

  //Timer 2. Avisa para implementar nueva velocidad. Cada 4,096 ms
  TCCR2A = 0; //Inicializamos los registros de control. Modo Normal
  TCCR2B = 0; //Inicializamos los registros de control. Modo Normal
//  TCCR2B |= (1 << CS22);                        //Prescalado a 64
  TCCR2B |= (1 << CS21)|(1 << CS22);            //Prescalado a 256
//  TCCR2B |= (1 << CS20)|(1 << CS21)|(1 << CS22); //Prescalado a 1024

  Serial.println("Inicio correcto: Pines asignados y Timers programados");


////////////////////////// INICIO OBTENCIÓN ÁNGULO ///////////////////////
  // initialize device
  Serial.println("Initializing I2C devices...");
  accelgyro.initialize();

  // verify connection
  Serial.println("Testing device connections...");
  Serial.println(accelgyro.testConnection() ? "MPU6050 connection failed" :
"MPU6050 connection successful");

  // Introducción de los OFFSets
  accelgyro.setXAccelOffset(-2328);
  accelgyro.setYAccelOffset(447);
  accelgyro.setZAccelOffset(572);
  accelgyro.setXGyroOffset(77);
  accelgyro.setYGyroOffset(-4);
  accelgyro.setZGyroOffset(-1);
```

```
//////////// FIN OBTENCIÓN ÁNGULO ///////////////////////

  //Activar Interrupciones de los timers (habrá que cambiar el valor de OCR1A
y OCR2A)
  OCR1A = npuls_max[3]; //Valor para interrupción. NO EMPEZAR EN 0
  OCR1B = npuls_max[3]; //Valor para interrupción. NO EMPEZAR EN 0
  OCR0B = 255; //Para calcular el ángulo
  TIMSK1 = 0; TIMSK2 = 0;//Limpiar máscaras del timer
  TIMSK0 |= (1 << OCIE0B);
  TIMSK1 |= (1 << OCIE1A)|(1 << OCIE1B); //Interrupción A, B del timer 1.
  TIMSK2 |= (1 << TOIE2); //OverFlow del timer 2.

}

void loop() {

delay(5);
bluetooth();

Serial.print(micros());Serial.print(" ");Serial.print(phi);Serial.print("
");Serial.print(phi_r);Serial.print(" ");Serial.print(dphi);Serial.print("
");Serial.print(dphi_r);Serial.print(" ");Serial.print(dtheta);Serial.print("
");Serial.print(dtheta_r);Serial.print(" ");Serial.println(alfa_A);

}


ISR(TIMER1_COMPA_vect)
{
  PORTD|=(1<<step_pin_A);//IMPORTANTE QUE ESTO SEA LO PRIMERO DE LA
INTERRUPCIÓN PARA QUE EL DRIVER DETECTE EL FLANCO
  OCR1A=OCR1A+npuls_A;
  PORTD&=(~(1<<step_pin_A));
  intA++;
}
ISR(TIMER1_COMPB_vect)
{
  PORTD|=(1<<step_pin_B);//IMPORTANTE QUE ESTO SEA LO PRIMERO DE LA
INTERRUPCIÓN PARA QUE EL DRIVER DETECTE EL FLANCO
  OCR1B=OCR1B+npuls_B;
  PORTD&=(~(1<<step_pin_B));
  intB++;
}


ISR(TIMER2_OVF_vect)
{
  interrupts();
  contLQR++;
  if (contLQR>=4)
  {
    contLQR = 0;
    //Proceso LQR
    phi = angulo*(PI/180);
    dphi = gsx*(PI/180);
```

```
    dtheta = (omega_A - omega_B)/2.0;
    ivr=ivr+dtheta_r-dtheta;
    u = K[0]*(phi) + K[1]*(dphi) + K[2]*(dtheta) + K[3]*ivr;
    alfa_A = u;
    alfa_B = -u;

  }

  omega_A = constrain (omega_A+alfa_A*T_PULS_ACEL + giro*0.5, -
omega_motor_max[0], omega_motor_max[0]);    //Calculamos al velocidad
acelerada del motor A.
  omega_B = constrain (omega_B+alfa_B*T_PULS_ACEL + giro*0.5, -
omega_motor_max[0], omega_motor_max[0]);    //Calculamos al velocidad
acelerada del motor B.

  //Implementación velocidad motor A:
  if (omega_A >= 0)
  {
    if(dir_A != 1)  //Tal vez es más rápido reactivar el puerto que poner un
if?
    {
      PORTD|=(1<<dir_pin_A);     //Si la velocidad es positiva activamos el
puerto DIR
      dir_A == 1;
    }
    if (omega_A <= omega_motor_max[3] && omega_A >=omega_motor_min[3])
//¿Dentro rango sensibilidad x 8?
    {
      PORTB |= (1 << ms1_pin_A)|(1 << ms2_pin_A);            //Micropasos 8/1
      npuls_A = omega2npuls[3]/omega_A;                  //Nº de incrementos
del timer para la frecuencia deseada.
    }
    else if (omega_A <= omega_motor_max[2] && omega_A >=omega_motor_min[2])
//¿Dentro rango sensibilidad x 4?
    {
      PORTB &= ~(1 << ms1_pin_A); PORTB |= (1 << ms2_pin_A); //Micropasos 4/1
      npuls_A = omega2npuls[2]/omega_A;                  //Nº de incrementos
del timer para la frecuencia deseada.
    }
    else if (omega_A <= omega_motor_max[1] && omega_A >=omega_motor_min[1])
//¿Dentro rango sensibilidad x 2?
    {
      PORTB &= ~(1 << ms2_pin_A); PORTB |= (1 << ms1_pin_A); //Micropasos 2/1
      npuls_A = omega2npuls[1]/omega_A;                  //Nº de incrementos
del timer para la frecuencia deseada.
    }
    else if (omega_A <= omega_motor_max[0] && omega_A >=omega_motor_min[0])
//¿Dentro rango sensibilidad x 1?
    {
      PORTB &= ~((1 << ms1_pin_A)|(1 << ms2_pin_A));         //Micropasos 1/1
      npuls_A = omega2npuls[0]/omega_A;                  //Nº de incrementos
del timer para la frecuencia deseada.
    }
    else //¿Dentro de ningún rango? Tenemos puesto que haga la mínima posible
```

```
y punto
    {
      PORTB |= (1 << ms1_pin_A)|(1 << ms2_pin_A);            //Micropasos 8/1
      npuls_A = npuls_max[3];
    }
  }
  else
  {
    if(dir_A != -1)  //Tal vez es más rápido reactivar el puerto que poner un
if?
    {
    PORTD&=(~(1<<dir_pin_A));     //Si la velocidad es negativa desactivamos
el puerto DIR
    dir_A == -1;
    }
    if (-omega_A <= omega_motor_max[3] && -omega_A >=omega_motor_min[3])
//¿Dentro rango sensibilidad x 8?
    {
      PORTB |= (1 << ms1_pin_A)|(1 << ms2_pin_A);            //Micropasos 8/1
      npuls_A = -omega2npuls[3]/omega_A;                 //Nº de incrementos
del timer para la frecuencia deseada.
    }
    else if (-omega_A <= omega_motor_max[2] && -omega_A >=omega_motor_min[2])
//¿Dentro rango sensibilidad x 4?
    {
      PORTB &= ~(1 << ms1_pin_A); PORTB |= (1 << ms2_pin_A); //Micropasos 4/1
      npuls_A = -omega2npuls[2]/omega_A;                 //Nº de incrementos
del timer para la frecuencia deseada.
    }
    else if (-omega_A <= omega_motor_max[1] && -omega_A >=omega_motor_min[1])
//¿Dentro rango sensibilidad x 2?
    {
      PORTB &= ~(1 << ms2_pin_A); PORTB |= (1 << ms1_pin_A); //Micropasos 2/1
      npuls_A = -omega2npuls[1]/omega_A;                 //Nº de incrementos
del timer para la frecuencia deseada.
    }
    else if (-omega_A <= omega_motor_max[0] && -omega_A >=omega_motor_min[0])
//¿Dentro rango sensibilidad x 1?
    {
      PORTB &= ~((1 << ms1_pin_A)|(1 << ms2_pin_A));         //Micropasos 1/1
      npuls_A = -omega2npuls[0]/omega_A;                 //Nº de incrementos
del timer para la frecuencia deseada.
    }
    else //¿Dentro de ningún rango? Cómo paramos motores?
    {
      PORTB |= (1 << ms1_pin_A)|(1 << ms2_pin_A);            //Micropasos 8/1
      npuls_A = npuls_max[3];
    }
  }

  //Implementación velocidad motor B:
  if (omega_B >= 0)
  {
    if(dir_B != 1)  //Tal vez es más rápido reactivar el puerto que poner un
if?
```

```
    {
      PORTD|=(1<<dir_pin_B);     //Si la velocidad es positiva activamos el
puerto DIR
      dir_B == 1;
    }
    if (omega_B <= omega_motor_max[3] && omega_B >=omega_motor_min[3])
//¿Dentro rango sensibilidad x 8?
    {
      PORTB |= (1 << ms1_pin_B)|(1 << ms2_pin_B);            //Micropasos 8/1
      npuls_B = omega2npuls[3]/omega_B;                  //Nº de incrementos
del timer para la frecuencia deseada.
    }
    else if (omega_B <= omega_motor_max[2] && omega_B >=omega_motor_min[2])
//¿Dentro rango sensibilidad x 4?
    {
      PORTB &= ~(1 << ms1_pin_B); PORTB |= (1 << ms2_pin_B); //Micropasos 4/1
      npuls_B = omega2npuls[2]/omega_B;                  //Nº de incrementos
del timer para la frecuencia deseada.
    }
    else if (omega_B <= omega_motor_max[1] && omega_B >=omega_motor_min[1])
//¿Dentro rango sensibilidad x 2?
    {
      PORTB &= ~(1 << ms2_pin_B); PORTB |= (1 << ms1_pin_B); //Micropasos 2/1
      npuls_B = omega2npuls[1]/omega_B;                  //Nº de incrementos
del timer para la frecuencia deseada.
    }
    else if (omega_B <= omega_motor_max[0] && omega_B >=omega_motor_min[0])
//¿Dentro rango sensibilidad x 1?
    {
      PORTB &= ~((1 << ms1_pin_B)|(1 << ms2_pin_B));         //Micropasos 1/1
      npuls_B = omega2npuls[0]/omega_B;                  //Nº de incrementos
del timer para la frecuencia deseada.
    }
    else //¿Dentro de ningún rango? Tenemos puesto que haga la mínima posible
y punto
    {
      PORTB |= (1 << ms1_pin_B)|(1 << ms2_pin_B);            //Micropasos 8/1
      npuls_B = npuls_max[3];
    }
  }
  else
  {
    if(dir_B != -1)  //Tal vez es más rápido reactivar el puerto que poner un
if?
    {
    PORTD&=(~(1<<dir_pin_B));     //Si la velocidad es negativa desactivamos
el puerto DIR
    dir_B == -1;
    }
    if (-omega_B <= omega_motor_max[3] && -omega_B >=omega_motor_min[3])
//¿Dentro rango sensibilidad x 8?
    {
      PORTB |= (1 << ms1_pin_B)|(1 << ms2_pin_B);            //Micropasos 8/1
      npuls_B = -omega2npuls[3]/omega_B;                 //Nº de incrementos
```

```
del timer para la frecuencia deseada.
    }
    else if (-omega_B <= omega_motor_max[2] && -omega_B >=omega_motor_min[2])
//¿Dentro rango sensibilidad x 4?
    {
      PORTB &= ~(1 << ms1_pin_B); PORTB |= (1 << ms2_pin_B); //Micropasos 4/1
      npuls_B = -omega2npuls[2]/omega_B;                 //Nº de incrementos
del timer para la frecuencia deseada.
    }
    else if (-omega_B <= omega_motor_max[1] && -omega_B >=omega_motor_min[1])
//¿Dentro rango sensibilidad x 2?
    {
      PORTB &= ~(1 << ms2_pin_B); PORTB |= (1 << ms1_pin_B); //Micropasos 2/1
      npuls_B = -omega2npuls[1]/omega_B;                 //Nº de incrementos
del timer para la frecuencia deseada.
    }
    else if (-omega_B <= omega_motor_max[0] && -omega_B >=omega_motor_min[0])
//¿Dentro rango sensibilidad x 1?
    {
      PORTB &= ~((1 << ms1_pin_B)|(1 << ms2_pin_B));         //Micropasos 1/1
      npuls_B = -omega2npuls[0]/omega_B;                 //Nº de incrementos
del timer para la frecuencia deseada.
    }
    else //¿Dentro de ningún rango? Tenemos puesto que haga la mínima posible
y punto
    {
      PORTB |= (1 << ms1_pin_B)|(1 << ms2_pin_B);            //Micropasos 8/1
      npuls_B = npuls_max[3];
    }
  }

  omega_A -= giro*0.5;
  omega_B -= giro*0.5;

}


ISR(TIMER0_COMPB_vect)
{
  //Calcular ángulo:
interrupts();
  if (1){
    accelgyro.getMotion6(&ax, &ay, &az, &gx, &gy,
&gz);                          //lecturas del acelerometro y giroscopio
    intIMU = 0;
    gsx =
float(gx)/131;                                                    //escal
ando la lectora del giroscopio en eje X
    ary = EscAcel * atan(ay / sqrt(square(ax) +
square(az)));           //calculo de angulo eje Y con el acelerometro
    angulo = (0.01 * ary) + (0.99 * (angulo + (gsx *
4*0.000256)));         //juntando valores del giroscopio y acelerometro con
un filtro complementario
  cont0_borrar++;
  }
```

```
}

void bluetooth() {
  String cadena = "";
  if (Serial.available() > 0) comando =  Serial.read();
  if (comando == 'F' ) {
    //Serial.println("ADELANTE");
    cadena = "ADELANTE";
    if (dtheta_r >= 0)dtheta_r = dtheta_r + 15;
    else dtheta_r = 0;
  }
  if (comando == 'B' ) {
    //Serial.println("ATRAS");
    cadena = "ATRAS";
    if (dtheta_r <= 0) dtheta_r = dtheta_r - 15;
    else dtheta_r = 0;
  }
  if (comando == 'L' ) {
    //Serial.println("IZQUIERDA");
    cadena = "IZQUIERDA";
    giro = giro + 1;
  }
  if (comando == 'R' ) {
    //Serial.println("DERECHA");
    cadena = "DERECHA";
    giro = giro - 1;
  }
  if (comando == 'S' ) {
    //Serial.println("PARA");
    cadena = "PARA";
    dtheta_r = 0;
    giro = 0;
  }
  if (comando == 'I' ) {
    //Serial.println("REINCIO");
    cadena = "REINICIO";
    omega_A = 0;
    omega_B = 0;
    ivr = 0;
  }
  if (comando == '+' ) {
    archivo ++;
    Serial.println(" ");Serial.print("INICIO ARCHIVO:
");Serial.println(archivo);
  }
    if (comando == '-' ) {
    Serial.println(" ");Serial.print("FIN ARCHIVO:
");Serial.println(archivo);
  }

//  if (comando != '0') Serial.print(cadena);
  comando = '0';
}
```

REFERENCIAS

[1] Antonio Croche, Vehículo Autoequilibrado de Tipo Péndulo Invertido de Bajo Coste

[2] Cecilia González, Mejora del software de un vehículo autoequilibrado de tipo péndulo invertido de bajo coste

[3] http://laplace.us.es/wiki/index.php/Est%C3%A1tica_de_la_part%C3%ADcula_(GIE)

[4] https://www.amazon.co.uk/Printer-Stepper-Motor-Degree-17HD34008-22B/dp/B01N2IDHH9#detail_bullets_id

[5] https://www.allegromicro.com/es-ES/Products/Motor-Driver-And-Interface-ICs/Bipolar-Stepper-Motor-Drivers/A3967.aspx

[6] https://www.amazon.es/gp/product/B071V8FV2G/ref=ox_sc_sfl_title_1?ie=UTF8&psc=1&smid=A250KNM9XJL1Q

[7] https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf

[8] https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf

[9] https://www.luisllamas.es/arduino-orientacion-imu-mpu-6050/

[10] https://www.luisllamas.es/arduino-i2c/

[11] https://www.olimex.com/Products/Components/RF/BLUETOOTH-SERIAL-HC-06/resources/hc06.pdf

[12] https://www.luisllamas.es/que-son-y-como-usar-interrupciones-en-arduino/

[13] http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf

[14] https://2018.robotix.in/tutorial/avr/timers/

[15] Low cost two-wheels self-balancing robot for control education, C. Gonzalez, I. Alvarado, D. Muñoz La Peña