Proyecto Fin de Grado

Grado en Ingeniería de las Tecnologías Industriales

Mejora del software de un vehículo autoequilibrado de tipo péndulo invertido de bajo coste

Autor: Cecilia González González

Tutor: Ignacio Alvarado Aldea

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

Sevilla, 2016

Proyecto Fin de Grado
Grado en Ingeniería de las Tecnología Industriales

Mejora del software de un vehículo autoequilibrado de tipo péndulo invertido de bajo coste

Autor:
Cecilia González González

Tutor:
Ignacio Alvarado Aldea
Profesor titular

Dep. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2016

Proyecto Fin de Grado: Mejora del software de un vehículo autoequilibrado de tipo péndulo invertido de bajo coste

Autor: Cecilia González González

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, 2016

El Secretario del Tribunal

A mis padres

A mi hermana

Agradecimientos

Dedico este proyecto fin de grado a todos los que me han apoyado y me han ayudado a llegar hasta aquí. Aquellos que no han dejado que me rinda nunca. Aquellos que han hecho que sea quien soy.

A mis padres que han sufrido parte de este gran esfuerzo. A mi hermana que siempre me ha escuchado, mi mejor amiga. A mis amigos de batalla, con los que he superado todas las barreras y que consiguen hacerme reir incluso en los momentos más difíciles. Recorrer todo esto sola habría sido imposible y a ellos les dedico este proyecto que lo han hecho realidad, porque como dicen mis padres a veces hay que ponerse las orejeras y saber que lo único que hay que hacer es mirar hacia adelante. A mi tutor Ignacio, que siempre ha estado para ayudarme y sacar este proyecto.

Simplemente gracias.

Cecilia González González

Sevilla, 2016

Resumen

En nuestra Escuela se realizan Jornadas de Puertas Abiertas para futuros alumnos de ingeniería. Este proyecto funcionará como demostrador de un sistema de control automático en un vehículo para el alumnado que finaliza sus estudios de Bachillerato.

Por otra parte, este proyecto se usará como planta de ensayo de diferentes estrategias de control en dispositivos de bajo coste en el Departamento de Ingeniería de Sistemas y Automática.

Abstract

In our School, Open Days are made for future engineering students. This project will work as a demonstrator of an automatic control system in a vehicle for students who completed her high school studies.
Moreover, this project will be used as a test plant for different control strategies in low-cost devices in the Department of Systems Engineering and Automation .

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

$\approx$	Aproximado
etc.	Etcétera

1 OBJETO DEL PROYECTO

Uno de los problemas más estudiados en control de sistemas es el péndulo invertido, sistema muy interesante por ser no lineal, multivariable y que presenta un cero de fase no mínima.

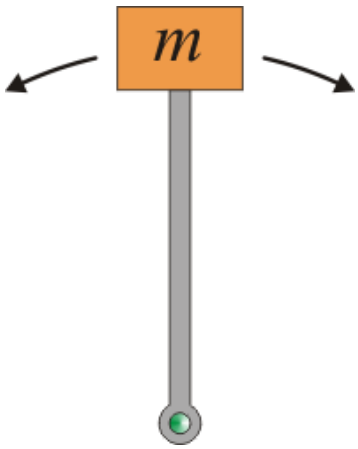

Figura 1-1. Péndulo invertido

Este proyecto trata de la mejora del control de un vehículo autoequilibrado de tipo péndulo invertido de bajo coste. El vehículo consta de una plataforma cuyo movimiento es realizado por dos ruedas accionadas por servomotores. Estos servomotores utilizarán los datos tomados de la unidad de medidas inerciales (IMU) que serán usados en el sistema de control que consigue el equilibrado del vehículo. Por lo que a partir de la entrada en tensión al motor esta provocará una velocidad angular, que será la salida del mismo, logrando con el resultado el control del autoequilibrado.

Se sabe que el sistema es de tipo péndulo invertido, un péndulo consta de dos posiciones de equilibrio, una estable y otra inestable, en este proyecto se busca el control de la posición de equilibrio inestable, es decir, la permanencia en la altura máxima a la que puede llegar el péndulo.

Se conoce que en la práctica, el equilibrio inestable en el péndulo invertido solo se alcanzará en determinados instantes de tiempo, ya que debido a perturbaciones o ruido en las señales de entrada es fácil que pierda la

posición de equilibrio. En consecuencia es imprescindible un sistema de control para obtener en los motores la salida deseada. Con esto se sabe que se debe calcular la velocidad angular que logra alcanzar la posición de equilibrio inestable o en su defecto la posición más cercana a este equilibrio. En la siguiente figura se pueden ver las variables usadas en el modelo.

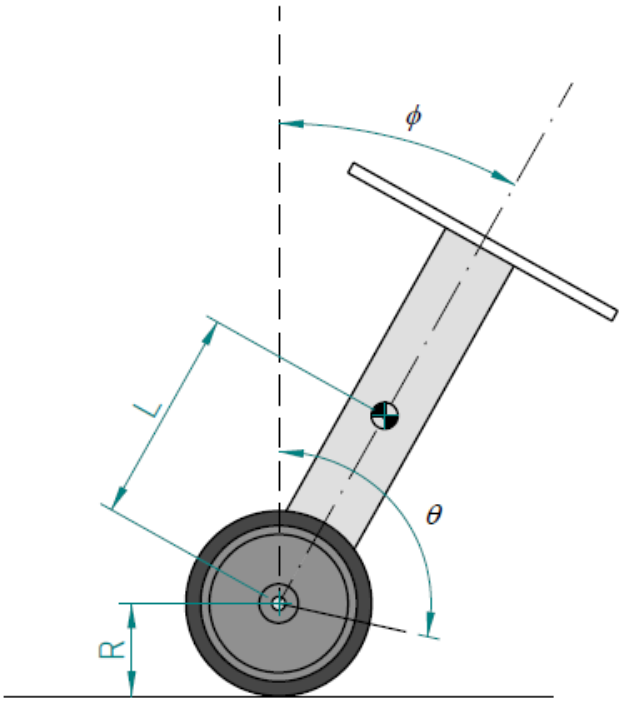

Figura 1-2. Diagrama del sistema con algunas variables

Se busca simular las estrategias de control en el prototipo del cual se tiene un modelo físico-matemático ya desarrollado anteriormente. Este modelo será usado también en las simulaciones en Simulink, donde se podrán evaluar sus parámetros numéricos. Los parámetros usados en la elaboración del modelo son:

- m_r: Masa de una rueda; su valor es 0.14 kg.
- R: Radio de las ruedas; cuyo valor es 0.05 m.
- J_r: Momento de inercia de una rueda; se obtiene $1.75 \cdot 10^{-4}$ kg·m^2.
- M: Masa del vehículo sin las ruedas; se calcula y se obtiene 0.82 kg.
- L: Distancia del eje de las ruedas al CG; el valor es 0.098 m.
- J_ϕ: Momento de inercia del vehículo sin las ruedas; se obtiene $7.875 \cdot 10^{-3}$ kg·m^2.
- g: Aceleración gravitatoria; se tomará el valor de 9.81 m/s^2.

Del estudio realizado del modelo se obtiene la siguiente ecuación:

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

$$a = \left(m_r + \frac{1}{2}M\right)R^2 + J_r$$

$$b = \frac{1}{2}(ML^2 + J_\phi)$$

$$c = RLM$$

$$d = MgL$$

$$J_r = \frac{1}{2}m_r R^2$$

$$J_\phi = ML^2$$

Otra de las características de este tipo de sistemas es que su centro de masas está por encima del eje sobre el

que rota el péndulo, en este proyecto el eje de las ruedas. Para conseguir alcanzar el punto de equilibrio, el centro de masas debe estar en la posición más elevada, por lo que el punto de equilibrio inestable buscado es la posición vertical tal y como se ha mencionado.

Una de las finalidades importantes de este proyecto es su uso como demostrador para futuros estudiantes de Ingeniería que asisten a la Feria del Estudiante para alumnado que finaliza su formación de Bachillerato y para las Jornadas de Puertas Abiertas en la Escuela de Ingenieros.

Otra de las finalidades de este proyecto es su uso como planta de ensayo de diferentes estrategias de control en dispositivos de bajos recursos en el ámbito del Departamento de Ingeniería de Sistemas y Automática.

2 PUNTO DE PARTIDA

En este capítulo se describen los dispositivos y características de los que consta el vehículo, además de los materiales del mismo. En la Figura 2.1 se muestra el vehículo al que se le van a realizar las mejoras en el software.

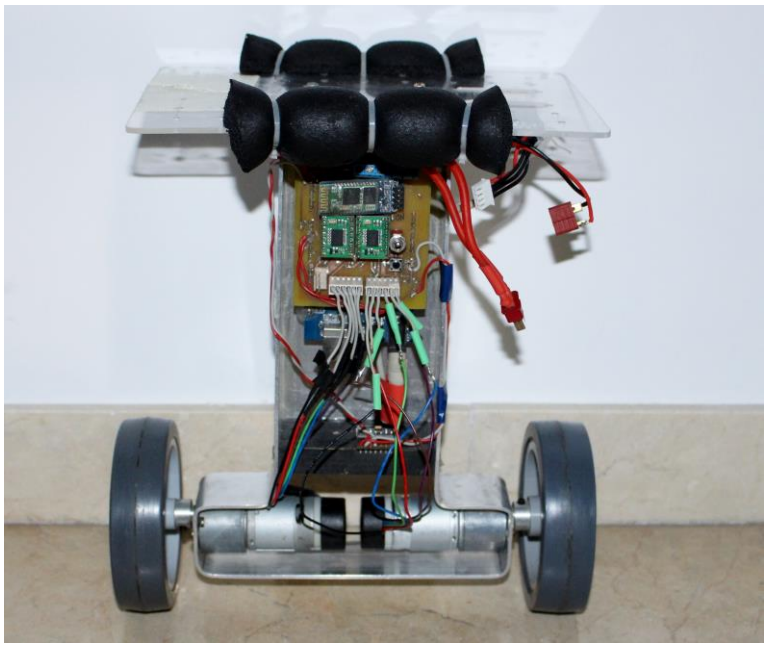

Figura 2-1. Imagen del vehículo

2.1. Materiales y sus características

El proyecto que se expone en esta memoria no parte de cero, sino de la continuación de un trabajo de final de carrera realizado por un alumno, el cual llegó a obtener resultados satisfactorios del sistema pero con posibles mejoras.

El material con el que se ha constado al inicio del proyecto ha sido el propio vehículo fabricado. Este está compuesto por:

2.1.1 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. Asimismo 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.

2.1.2 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 2–1 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 2–2 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 2–3 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.

2.1.3 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.

2.1.4 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.

2.1.5 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 2–4 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.

2.1.6 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 2–5 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 2–6 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.

2.1.7 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 2–7 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

2.1.8 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.

2.1.9 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.

3 MEJORAS EN EL FUNCIONAMIENTO

En este capítulo se presentan las distintas mejoras realizadas para el correcto funcionamiento del vehículo, ya que el Segway presentaba una serie de problemas.

Se van a realizar estas cuatro mejoras:

1. Mejora en el uso de la unidad de medidas inerciales.
2. Implementación del uso de temporizadores para garantizar los tiempos de cálculo de los controladores.
3. Mejora del funcionamiento del control PI del motor.
4. Rechazo de las perturbaciones originadas por el centro de gravedad.

Entre estos problemas se encuentra la dificultad que tenía el aparato para la correcta lectura de la IMU, lo que provocaba que dejara de controlar el sistema e irremediablemente el vehículo se caía. La solución al problema más urgente de resolver se encuentra explicada en el primer apartado, donde además se realiza un estudio del tipo de fallos que se dan en la unidad de medidas inerciales.

Por otra parte, una vez visto el código, se puede mejorar la precisión del intervalo de tiempo en el que comienza el control LQR y el control PI, esto quiere decir que se trabajará con interrupciones temporales. Se utilizarán los temporizadores que incorpora la placa de Arduino, sin tener que contabilizar así el tiempo que transcurre mientras se realizan los cálculos del control. Se consigue así un tiempo fijo en el que se logra realizar el cálculo de la acción de control, que con el anterior código no se podía asegurar.

Otro punto importante es la mejora del funcionamiento del control PI, para ello se usan los datos que se pasan desde Bluetooth a Matlab. En Matlab se podrá ver dónde está el mejor comportamiento del sistema usando el modelo como referencia de la actuación óptima.

Por último, otra de las mejoras es el rechazo de las perturbaciones originadas por el centro de gravedad. Existe esta perturbación ya que el centro de gravedad del aparato no se encuentra en su centro geométrico. Este problema se resolverá a través del software trabajando con el modelo en variables incrementales según se explica en el último apartado de este Capítulo.

3.1. Mejora en el uso de la unidad de medidas inerciales

Con el uso del vehículo se comprueba que el sistema no es capaz de controlarse en determinados momentos, provocando la caída del Segway. Por otra parte, no solo se cae sino que ya no vuelve a intentar controlar el sistema. Esto quiere decir que si se vuelve a colocar de manera correcta para que siga la trayectoria deseaada por el usuario, los motores del vehículo no actúan, este que queda completamente paralizado.

Para encontrar el error se utiliza el Monitor Serie, que muestra por pantalla los resultados que se van obteniendo a través del Bluetooth. Ya que el antiguo código muestra mensajes de error, se observa que aparece un error en la lectura de la unidad de medidas inerciales. El mensaje mostrado por el Monitor Serie es "IMURead timeout".

A partir del primer mensaje de error, el sistema lo mostrará continuamente por el Monitor Serie y el Segway dejará de estabilizarse. La unidad de medidas inerciales (IMU) se queda incapacitada para volver a intentar obtener los nuevos datos necesarios para el control de los motores.

3.1.1 Elección del tipo de solución

Inicialmente, parecía que el problema estaba en las posibles interferencias electromagnéticas que pueden darse en los cables que transmiten datos. Para solucionar el problema, una de las opciones sería cambiar algunas partes del hardware.

En la transmisión de datos de la IMU a Arduino se puede encontrar este posible fallo debido a interferencias electromagnéticas provocadas por las controladoras de motores. Existen soluciones al problema cambiando el hardware, entre las soluciones se encuentran cuatro posibilidades:

- Las interferencias dependen de la distancia a la que estén los cables que transmiten datos. Por lo tanto si físicamente se separan estos cables se consegue solventar el problema. No se puede variar la frecuencia, ni la longitud de los cables ni la potencia absorbida por la máquina, solo se puede variar la distancia de forma teórica. Sin embargo, en la práctica esto no es posible por la propia arquitectura del Segway por lo que queda descartada esta opción.
- Otra solución alternativa es trenzar el cable, cuya efectividad aumenta en campos magnéticos de baja frecuencia. Esto es posible realizarlo en una pequeña parte del mismo ya que estos cables se encuentran soldados a la placa dificultando su manejo. Esta opción tampoco logra arreglar el problema planteado.
- El apantallamiento es otra solución para la eliminación de las interferencias electromagnéticas. Esta opción requeriría poner un cable apantallado, por lo que aumentamos el valor del vehículo no siendo la mejor resolución ya que el apantallamiento no sería tan efectivo como el trenzado el cual tampoco consigue quitar el problema de interferencias electromagnéticas.
- Se puede blindar el cable de potencia con un conductor metálico, ya que los apantallamientos mediante materiales de baja permeabilidad suelen ser poco efectivos. Esta solución requiere un coste adicional, por lo tanto se descarta.

Ninguna de las soluciones Hardware son suficientemente efectivas por otra parte la estructura está ya cerrada y no es conveniente hacerle cambios, se deben realizar los cambios en el Software. Además, estudiando el antiguo código se llega a la conclusión de que el fallo no es del Hardware sino del propio código. El fallo se encuentra en la librería que se implementa para la uso de los datos de la IMU. Se descubre que al darse el problema de lectura de la unidad de medidas inerciales, el código de la librería que se usa permanece en alguno de sus bucles, provocando un bucle infinito.

El primer estudio del código para entender el bucle en el que permanece una vez aparece un error, se hará en la propia libería utilizada, la librería Wire. Se examina el código de la librería para entender su funcionamiento y encontrar el punto en el que permanece de forma infinita. Se puede comprobar que esta librería tiene programados algunos bucles que pueden ocasionar los problemas que aparecen en la lectura de la unidad de medidas inerciales.

Como ejemplo se muestra a continuación el siguiente código de la librería Wire:

```
while (something != value) {
  something = grabValue
}
```

Figura 3-1. Código de la librería Wire

Básicamente en el código si *something* nunca se iguala a *value* nunca se puede salir del bucle, este tipo de código provoca el problema encontrado en la lectura. La mejor solución es cambiar esta librería por la librería I2C, que no tiene este tipo de problemática. Con esta librería si se llega a este tipo de bucle hay un tiempo muerto y si *something* nunca llega a ser igual a *value* y ese tiempo se pasa a genera un código de error y vuelve el control del sistema, siendo la IMU capaz de seguir obteniendo datos. Por otra parte, las funciones que hay en esta librería son más fáciles de usar.

3.1.2 Configuración de la nueva librería

La implementación de la librería requiere un cambio de todas las funciones de la librería Wire a la librería I2C. El primer cambio que se realiza dentro del código es añadir con un include la nueva librería, es decir se debe añadir la siguiente línea de código tal y como se ve en la Figura 3-2:

```
#include <I2C.h>  // Esta librería permite  comunicar con dispositivos I2C  TWI.
```

Figura 3-2. Adición de la nueva librería

Para comenzar el cambio de librería se empieza por la inicialización, para ello existe una función que permite abrir la comunicación I2c, I2c.begin(), esta función es similar a la función de la librería Wire. A continuación se configura la frecuencia.Mirando el Datasheet del MPU6050, la velocidad máxima del bus es 400 kHz que será la frecuencia usada. Para configurarla se usa la función I2c.setSpeed(x), donde *x* puede ser 0 ó 1 según el valor al que se quiera configurar la frecuencia. Si se usa el 0, la frecuencia será de 100 kHz y si se usa el 1, la frecuencia será de 400 kHz. Como lo que interesa es una frecuencia de 400 kHz, *x* debe valer 1.

Seguidamente se configura la resistencia pull-up o pull-down. En este proyecto se activará la pull-up que está por defecto configurada en la librería Wire pero que en esta librería sí se debe especificar. La función que se necesita para la configuración de la resistencia es I2c.pullup(y), donde *y* puede valer 0 ó 1. Si *y* vale 1 la resistencia es de pull-up, por el contrario si es 0, la resistencia es de pull-down. Por tanto el valor de *y* es 1.

Para finalizar la inicialización, se configura el tiempo de espera. Si la función lo sobrepasa saltará algún error. Para este sistema el máximo tiempo de espera son 40 milisegundos. En la Figura 3-3, se puede ver la inicialización de la comunicación I2c. Es importante poner esta inicialización en el setup() del código de Arduino.

```
void setup() {
 I2c.begin();
```

```
        I2c.setSpeed(1); //Poner frecuencia a 400kHz
        I2c.pullup(1); //activamos la resistencia de pull-up
       I2c.timeOut(40);  //habilita el recibir los errores por el envío fallido y no se queda colgado ya que
//puede saber el tiepo de error y seguir funcionando
}
```

Figura 3-3. Inicialización de la comunicación I2c

La placa de Arduino especifica el registro que debe usarse para la comunicación I2c, se elige el registro de bajo voltaje que es el 0x68, en binario 1101000. Esta dirección será usada en las funciones programadas para la lectura de la unidad de medidas inerciales. Se configura según se muestra en la Figura 3-4.

```
const uint8_t IMUAddress = 0x68;
```

Figura 3-4.Variable para el registro de la comunicación I2c

Queda añadir que se necesita tanto una función que escriba en la IMU como otra que lea. Principalmente la más importante es la de lectura, pero se necesita la de escritura para inicializar el dispositivo correctamente.

3.1.2.1 Escritura en la IMU

Dentro de la inicialización, se necesitan configurar la tasa de muestreo, la sincronización y el filtrado. También se configuran los giróscopos y los acelerómetros, que se inicializan a la máxima sensibilidad y por tanto el mínimo rango de ángulo o de aceleraciones.

Todos estos datos se guardarán en la IMU con una función de escritura programada. Esta función de escritura se ha cambiado usando las funciones de la librería I2C. En el anterior código la transmisión de los datos se hacía paso a paso, sin embargo con la nueva librería no es necesario ya que una única función realiza todo el proceso de transmisión. En la librería Wire para realizar una escritura se necesita primero empezar la transmisión para ello se usa la dirección que comunicará la placa Arduino con la IMU, definiéndose así físicamente el lugar de la memoria donde irán los datos de la unidad de medidas inerciales. El siguiente paso será definir el registro de la IMU donde se escribirán los datos y por último queda escribir en el registro de la IMU las configuraciones antes mencionadas. Se finaliza esta comunicación enviando a ese registro el segmento de datos y su tamaño.

En la finalización de la comunicación se comprueba si la función devuelve un número distinto a cero, que implicaría que ha habido un error en la escritura. Los códigos de error se encuentran entre los valores 0 y 8, la función devolverá alguno de estos valores. Cada valor tiene un significado distinto como se puede comprobar:

- 0: No hay ningún informe de error. Si aparece esto es que el código se ha ejecutado de forma correcta. Este debe ser el valor normal que devuelva la función de escritura en la dirección de Arduino a través de la comunicación I2c.
- 1: El tiempo de espera ha sido mayor que el programado, es decir se decide un tiempo muerto, si este pasa la función devuelve un valor distinto de cero que es el único que no significa error. Si la función devuelve un 1 el código está a la espera de la confirmación de la condición de comienzo de transmisión de datos.

- 2: si la función devuelve un 2, el código está en espera por ACK o NACK después de transmitir la dirección y de escribir la variable bandera. ACK (acuse de recibo o asentimiento) es un mensaje que el destino de la comunicación envía al origen de esta para confirmar la recepción de un mensaje, además puede informar si se ha recibido todo de forma íntegra y si no ha sido así aparecerá el error. NACK (acuse de recibo negativo o asentimiento) es un mensaje del protocolo que se envía para informar de que en la recepción de una trama de datos ha habido un error. Esto quiere decir que aunque se haya mandado la dirección, esta no ha llegado correctamente al destino y salta la variable de error. Normalmente suele haber reenvío y se consigue de esta manera que entre la dirección.
- 3: este tipo de error se produce después de la transmisión de datos una vez transmitida la dirección y escrita la variable bandera. Este error se da ya que hay una espera porque ha habido error al transmitir el segmento de datos aunque la dirección sí llega correctamente.
- 4: el código permanece en una espera de la confirmación de la repetición de la condición de comienzo de transmisión de datos.
- 5: este error es parecido al 2 con la diferencia de que no es escritura de la variable bandera sino su lectura.
- 6: este otro tipo de error se parece al 3 con la diferencia que no es transmitir el segmento de datos sino recibirlo.
- 7: si la función devuelve un 7, el código se ha quedado a la espera de que se cumpla la condición de paro de la transmisión de datos.
- 8: aparece 0xFF, es decir hay una incompatibilidad en el tamaño del Buffer.

En los ensayos realizados para la búsqueda del fallo, se comprueba que este no es el error que provoca la caída del Segway, sin embargo ya que se ha cambiado la librería, deben cambiar también las funciones utilizadas en el código.

En la Figura 3-5 se muestra el nuevo código usado en la función programada para la escritura. En el nuevo código, cuando finaliza la escritura se ha añadido la liberación del bus dejando que el pin vuelva a ser un pin analógico estándar de Arduino. La liberación del bus se realiza con una función de la nueva librería ya que esta opción no es capaz de hacerla la función de escritura de la librería I2C al contrario que la librería Wire, cuya función de escritura liberaba el bus.

```
uint8_t IMUWrite(uint8_t registerAddress, uint8_t data, bool sendStop) {
 return IMUWrite(registerAddress, &data, 1, sendStop); // devuelve 0 si tiene éxito
}

uint8_t IMUWrite(uint8_t registerAddress, uint8_t *data, uint8_t length, bool sendStop) {
//registerAddress es el registro de la IMU
 uint8_t rcode=I2c.write(IMUAddress,registerAddress,*data);  //si devuelve 0 es que ha tenido éxito
 if (rcode) {
  Serial.print(F("IMUWrite failed: "));
  Serial.println(rcode);
 }
 if (sendStop=true){
  I2c.end();
```

```
    Serial.println(F("libera bus"));
  }
  return rcode;
}
```

Figura 3-5. Función de escritura

3.1.2.2 Lectura de la IMU

La función más usada sin duda será la lectura de la unidad de medidas inerciales. Para su buen funcionamiento se cambiarán las funciones de la librería Wire y se usará la librería I2C.

En el antiguo código para la lectura también se debe realizar paso a paso la conexión, por lo que primero se especifica el registro por el que se comunica la IMU y a continuación se escribe el registro del que se lee. Una vez conseguido esto simplemente se espera a que el segmento de datos haya llegado a su destino en la placa de Arduino. Se finaliza la transmisión comprobando el valor que devuelve la función que finaliza esa transmisión, ya que si el intento es fallido aparecerá un mensaje de error escrito en el propio código. Este mensaje no es el que aparece cuando el vehículo cae por lo que sabemos que la transmisión acaba correctamente. Toda esta conexión no es necesaria hacerla tan detalladamente con las funciones de la librería I2C ya que la propia función de lectura realiza todo esto.

A continuación, en el código con la librería Wire se procederá a la lectura del segmento de datos. En esta lectura se tiene en cuenta el tiempo que tarda en realizarla ya que si lo sobrepasa, los datos no tendrían ningún valor para el control del sistema. Se encuentra en esta parte del código el fallo, y el mensaje que aparece tras la caída es que se ha sobrepasado el tiempo en el que se puede leer. Este error provoca que el Segway ya no vuelva a ser capaz de leer y que el vehículo no consiga volver a equilibrarse.

La librería I2C es la encargada de arreglar este problema ya que el código no se queda congelado. La función devolverá un valor distinto de cero para indicar el tipo de error pero vuelve a intentar realizar la siguiente lectura. Se soluciona así el problema de la pérdida del control del vehículo.

En la Figura 3-6 se encuentra el código para la lectura de la IMU. Este código debe comprobar siempre que se ha recibido la información.

```
uint8_t IMURead(uint8_t registerAddress, uint8_t *data, uint8_t nbytes) {
  uint32_t timeOutTimer;
  I2c.read(IMUAddress,registerAddress,nbytes);
  for (uint8_t i = 0; i < nbytes; i++) {
    if (I2c.available())
      data[i] = I2c.receive();
    else {
      timeOutTimer = micros();
      while (((micros() - timeOutTimer) < 1000) && !I2c.available());
      if (I2c.available())
        data[i] = I2c.receive();
```

```
    else {
      Serial.println(F("IMURead timeout"));
      return 5;
    }
   }
  }
 return 0; //Si tiene éxito
}
```

Figura 3-6. Función de lectura

La conclusión al problema encontrado en la IMU es que la transmisión en la lectura tiene momentos en los que dura más de lo deseado por lo que los datos no son fiables y no se consideran. Además el microcontrolador en estos casos no recibe los datos. La forma de solucionarlo es volviendo a intentar tomar los datos nuevos y seguir funcionando a pesar del error pasado. Se consigue así de la forma más barata posible arreglar esta dificultad.

3.2. Interrupciones temporales

3.2.1. Descripción para el uso de los Timers

Un Timer (temporizador) es una parte del hardware situada en el controlador de la placa de Arduino. Podemos decir que se comporta como un reloj ya que es capaz de calcular las interrupciones temporales. Para el correcto funcionamiento del Timer se tienen que configurar algunos registros y en su inicialización se debe configurar al menos el preescalador y el modo de operación.

Las interrupciones temporales son funciones que se activan una vez el Timer haya dado la orden, es decir, el Timer es un contador y cuando el valor del registro que va aumentando a lo largo del tiempo desborda, se dispara para activar el código de la interrupción y esto lo hará cada cierto tiempo (tiempo fijo).

3.2.1.1. Tipos de Timers en la placa de Arduino

La placa de Arduino de la que disponemos tiene el microcontrolador Atmel ATMega 2560, este está compuesto por 6 tipos distintos de Timers entre ellos:

- Timer 0: es un temporizador de 8 bits, es decir, que registrará 2^8 valores que son 256. Este temporizador es usado en la funciones delay(), millis() y micros(), por lo que se debe tener en cuenta si se usa o no cuando se programa. Este Timer se puede configurar en modo contador (el valor del registro TMR0 se incrementa con cada ciclo de instrucción) o en modo temporizador (el valor del registro TMR0 se incrementa en cada flanco de reloj, que puede ser ascendente o descendente).
- Timer 1: es un temporizador de 16 bits, este puede registrar hasta 1024 valores. Este temporizador se usa en la librería servo pero dado que en este proyecto esta librería no se usa, se puede hacer uso de este temporizador para esta parte del código. El Timer 1 tiene dos registros de 8 bits que son para lectura y escritura. Este módulo cuenta desde 0x000 hasta 0xFFF y al desbordarse vuelve a 0x000, es

decir se resetea. A diferencia del Timer 0 este temporizador puede activar o detener la cuenta. El Timer 1 también se puede configurar en modo temporizador o en modo contador al igual que el temporizador 0.

- Timer 2: es un temporizador de 8 bits. Es similar al Timer 0 con la diferencia de que este es el que se usa para la función tone(). Este temporizador cuenta con un preescalador y un post-escalador. Este timer ya está en uso para el PWM de uno de los motores con el que se consigue configurar la frecuencia. Por consiguiente no es posible usarlo para otra tarea distinta como puede ser la realización en un intervalo de tiempo fijo de la acción de control.
- Timers 3, 4, 5: son temporizadores de 16 bits, estos están solo disponibles en este microcontrolador y su funcionamiento es muy similar al Timer 1. Los pines de los Timers en uso son tanto el del Timer 3 como del 4. El Timer 3 necesita los pines 2, 3 y 5 que ya son usados para interrupciones externas (Pin 2 y 3). El Timer 4 es igual que el 2, sirve para el PWM del otro motor, donde también se puede configurar el valor de la frecuencia. Estos dos Timers quedan descartados para su uso en la realización de la acción de control, quedando solo libre el Timer 5.

En conclusión, si se necesitan usar los Timers, los que quedan libres son el 1 y 5. Si se usaran quedarían todos los temporizadores en uso.

3.2.1.2. Tipos de registros de un Timer

Para cambiar el comportamiento del Timer tendremos que configurar los registros del mismo. Los más importantes son los siguientes:

- TCCRx: registro de control de la cuenta del temporizador (Timer/Counter Control Register). En este registro se configura el preescalador y el modo de operación. Se divide entre el A y el B en cada uno de los Timers.

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 3-7. Registro TCCR1A

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 3-8. Registro TCCR1B

En la Figura 3-7 y 3-8 se utiliza de ejemplo el Timer 1 cuyo registro es TCCR1, este registro tiene a su vez dos, TCCR1A y TCCR1B, según otras tablas que aparecen más adelante se puede definir como se ha mencionado antes el preescalador a través de los valores de CSxy. Para el modo de operación se usarán WGMxy cuyos registros se configuran según valores de otra tabla que aparece más adelante.

- TCNTx: registro contador (Timer/Counter Register). En este registro se almacena el valor actual del Timer.
- OCRx: registro de comparación a la salida (Output Compare Register), este es un modo de operación que se explicará más adelante.
- ICRx: registro de captura a la entrada (Input Capture Register), este solo es válido para Timers de 16 bits. Es un registro usado para uno de los tipos de los modos de operación al igual que el registro OCRx.
- TIMSKx: registro de la máscara de las interrupciones del contador. Este activa o desactiva las interrupciones del Timer.
- TIFRx: registro de la bandera de las interrupciones del contador. Este indica las interrupciones pendientes del Timer.

Como ya se ha mencionado para las interrupciones temporales se usarán Timers que estén libres. Para el control LQR se usará uno de los Timers libres, entre ellos el 1 o el 5. Lo mismo sucede con el control PI. Sin embargo, queda por decidir cómo repartir el uso de los temporizadores para los distintos controles algo que dependerá de la prioridad de cada Timer y del controlador. El estudio de las prioridades se resuelve en el apartado 3.2.2.

3.1.2.3 Configuración del preescalador y del modo de operación

Los Timers dependen de una fuente de reloj del propio microcontrolador, por lo tanto la unidad más pequeña medible es el periodo que ofrece el reloj del microcontrolador. Para saber dicho periodo se sabe que este reloj es de 16 MHz, por lo que calculando la inversa se obtiene el periodo.

La fórmula para el cálculo del periodo con el dato de la frecuencia es la siguiente:

$$T = \frac{1}{f} = \frac{1}{16MHz} = 62.5ns$$

Buscamos que las interrupciones sean cada 10 ms la del control PI y cada 40 ms el control LQR. Por lo tanto el tiempo fijo para que se active la interrupción es mucho mayor que el periodo calculado del reloj del microcontrolador. Esto quiere decir que los Timers se deben configurar con los requisitos especificados de tiempo y no con los configurados por defecto (los del reloj interno).

Inicialmente se configura el preescalador según los valores CSx0, CSx1 y CSx2. El preescalador es un divisor de frecuencia configurable antes de cada incremento. Su finalidad es hacer que el temporizador sea más lento comparado con el reloj interno de la placa microcontroladora.

El cálculo del preescalador aparece en el apartado 3.2.3 de cálculo de los valores de los registros para la inicialización de los Timers. Se observa en la Figura 3-9 qué valores tiene cada registro según el preescalador requerido, se utiliza en esta tabla el Timer 1 como ejemplo.

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/1 (No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

Figura 3-9. Selección del preescalador

A continuación se explican los diferentes modos de operación que se pueden usar, existen cuatro modos:

- Timer overflow (desbordar): cuando el temporizador alcanza un valor límite, el valor del registro TOVx se establece en el registro de la interrupción por bandera que es el registro TIFRx, se llamará a la rutina de interrupción (interrupt service routine) ISR(TIMERx_OVF_vect).
- Output Compare Match (activa la interrupción por comparación de valores): cuando salta este tipo de interrupción, la bandera OCFxy del registro OCTx se establece en el registro de la interrupción por bandera, TIFRx. A continuación se establece el output compare match interrupt service en el registro TIMSKx a través de la activación de OCIExy, que permite que la interrupción esté activada. Esta rutina se llama ISR(TIMERx_COMPy_vect).
- Timer Input Capture (captura el tiempo especificado por el programador para activar la interrupción): si salta este tipo de interrupción se activará TIFRx ya que se activa ICFx y TIMSKx hará que comience la rutina de interrupción ISR(TIMERx_CAPT_vect).
- PWM (modulación por ancho de pulsos): este modo modifica el ciclo de trabajo de una señal periódica.

Para elegir el modo de operación se configuran unos valores determinados en los registros, estos se pueden ver en la Figura 3-10 donde también se usa el Timer 1 como ejemplo.

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1x at	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICR1	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCR1A	BOTTOM	TOP

Figura 3-10. Configuración del modo de operación

El modo de operación que se va a utilizar en ambos temporizadores es el Output Compare, modo CTC, mirando la Figura 3-10 se comprueba que este modo es el cuarto, por lo que en la inicialización tendremos que activar o desactivar WGMx0, WGMx1, WGMx2 y WGMx3. Por otra parte según esta tabla se sabe que en el registro OCR1A u OCR5A (según el Timer que se necesite usar) hay que guardar el valor máximo hasta el que se tiene que contar para llegar al tiempo especificado, en nuestro caso a 10 ms o a 40 ms según el Timer del que se trate.

3.2.2. Elección de la tarea de cada Timer según su prioridad

Como se ha comentado anteriormente, se pueden usar los Timers 1 y 5, ya que se necesitan dos interrupciones temporales, una por cada tipo de control (control LQR y el PI de los motores), se utilizan por tanto estos dos Timers. Para su uso se tendrá en cuenta también la prioridad de los Timers y la prioridad del controlador que se necesite.

Cuando se produce el evento que desencadena la interrupción, el microcontrolador procederá a interrumpir la ejecución del programa principal y atender la interrupción. Para realizarlo de la forma correcta, se debe saber con anterioridad cómo tratar dicha interrupción. Para ello, se programa la rutina de atención a la interrupción.

Los microcontroladores tienen una zona conocida de memoria donde se almacenan las direcciones de las distintas rutinas de tratamiento para las diferentes interrupciones. La tabla de vectores de interrupción es esta zona de memoria. Si el microcontrolador reconoce una interrupción, este se encarga de buscar en la tabla de vectores de interrupción la dirección de memoria donde debe saltar para realizar la rutina correspondiente, donde estará guardado el código de la interrupción.

Cuando el tratamiento de la interrupción termina, el microcontrolador hará que se siga con la misma tarea que estaba realizando antes de la interrupción. Para asegurar esto, el microcontrolador guarda el estado en el que se encuentra y una vez acabada la rutina de interrupción se restaura el estado y se continúa con la ejecución del programa principal.

El acceso a la tabla de vectores de interrupción está organizado mediante un índice “n” que permite acceder a la dirección de la correspondiente rutina de interrupción. Además este índice permite organizar la tabla de interrupciones para asignar la prioridad de ejecución cuando se producen dos interrupciones de manera simultánea.

Las interrupciones se organizan por tanto en vectores de interrupción con prioridad según se comprueba en la Figura 3-11 y Figura 3-12. La dirección más baja del vector de interrupciones tiene prioridad sobre la más alta.

La rutina de interrupción de servicio (ISR), que es la que se implementa en las interrupciones temporales, no devuelve ningún parámetro ni recibe nada.

Si una ISR está en funcionamiento, no pueden funcionar otras interrupciones, esto puede ser un problema ya que las interrupciones de los encoders son más prioritarias que las interrupciones internas. Sin embargo se asegura, según la tabla donde aparecen los vectores de interrupción, que las interrupciones externas son prioritarias a las internas provocadas por temporizadores.

Las interrupciones externas, es decir la de los encoders, se atienden antes que las del control PI y el control LQR (interrupciones internas con temporizadores) ya que por defecto son prioritarias según la configuración de la propia placa microcontroladora. Por otra parte, se usarán como temporizadores el Timer 1 y el 5 ya que sus pines no están en uso. Y se tendrá en cuenta que el Timer 1 es prioritario al Timer 5, usando el Timer 1 para el control PI y el Timer 5 para el control LQR. Se decide esta prioridad ya que el control PI se realiza cuatro veces más que el control LQR. Dándole así más exactitud al intervalo de tiempo en el que se realiza el cálculo del control PI.

Queda añadir que todas las variables que deban ser modificadas por la rutina ISR deben ser declaradas como *volatile*, esto nos asegura que las variables usadas dentro de una ISR son actualizadas correctamente.

A continuación se muestran en la Figura 3-11 y la 3-12 el vector de interrupción según prioridad de esta.

Vector No.	Program Address[2]	Source	Interrupt Definition
1	$0000[1]	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$0002	INT0	External Interrupt Request 0
3	$0004	INT1	External Interrupt Request 1
4	$0006	INT2	External Interrupt Request 2
5	$0008	INT3	External Interrupt Request 3
6	$000A	INT4	External Interrupt Request 4
7	$000C	INT5	External Interrupt Request 5
8	$000E	INT6	External Interrupt Request 6
9	$0010	INT7	External Interrupt Request 7
10	$0012	PCINT0	Pin Change Interrupt Request 0
11	$0014	PCINT1	Pin Change Interrupt Request 1
12	$0016[3]	PCINT2	Pin Change Interrupt Request 2
13	$0018	WDT	Watchdog Time-out Interrupt
14	$001A	TIMER2 COMPA	Timer/Counter2 Compare Match A
15	$001C	TIMER2 COMPB	Timer/Counter2 Compare Match B
16	$001E	TIMER2 OVF	Timer/Counter2 Overflow
17	$0020	TIMER1 CAPT	Timer/Counter1 Capture Event
18	$0022	TIMER1 COMPA	Timer/Counter1 Compare Match A
19	$0024	TIMER1 COMPB	Timer/Counter1 Compare Match B
20	$0026	TIMER1 COMPC	Timer/Counter1 Compare Match C
21	$0028	TIMER1 OVF	Timer/Counter1 Overflow
22	$002A	TIMER0 COMPA	Timer/Counter0 Compare Match A
23	$002C	TIMER0 COMPB	Timer/Counter0 Compare match B
24	$002E	TIMER0 OVF	Timer/Counter0 Overflow
25	$0030	SPI, STC	SPI Serial Transfer Complete
26	$0032	USART0 RX	USART0 Rx Complete
27	$0034	USART0 UDRE	USART0 Data Register Empty
28	$0036	USART0 TX	USART0 Tx Complete
29	$0038	ANALOG COMP	Analog Comparator

Figura 3-11. Tabla de vectores de interrupción según prioridad

Vector No.	Program Address[2]	Source	Interrupt Definition
30	$003A	ADC	ADC Conversion Complete
31	$003C	EE READY	EEPROM Ready
32	$003E	TIMER3 CAPT	Timer/Counter3 Capture Event
33	$0040	TIMER3 COMPA	Timer/Counter3 Compare Match A
34	$0042	TIMER3 COMPB	Timer/Counter3 Compare Match B
35	$0044	TIMER3 COMPC	Timer/Counter3 Compare Match C
36	$0046	TIMER3 OVF	Timer/Counter3 Overflow
37	$0048	USART1 RX	USART1 Rx Complete
38	$004A	USART1 UDRE	USART1 Data Register Empty
39	$004C	USART1 TX	USART1 Tx Complete
40	$004E	TWI	2-wire Serial Interface
41	$0050	SPM READY	Store Program Memory Ready
42	$0052[3]	TIMER4 CAPT	Timer/Counter4 Capture Event
43	$0054	TIMER4 COMPA	Timer/Counter4 Compare Match A
44	$0056	TIMER4 COMPB	Timer/Counter4 Compare Match B
45	$0058	TIMER4 COMPC	Timer/Counter4 Compare Match C
46	$005A	TIMER4 OVF	Timer/Counter4 Overflow
47	$005C[3]	TIMER5 CAPT	Timer/Counter5 Capture Event
48	$005E	TIMER5 COMPA	Timer/Counter5 Compare Match A
49	$0060	TIMER5 COMPB	Timer/Counter5 Compare Match B
50	$0062	TIMER5 COMPC	Timer/Counter5 Compare Match C
51	$0064	TIMER5 OVF	Timer/Counter5 Overflow
52	$0066[3]	USART2 RX	USART2 Rx Complete
53	$0068[3]	USART2 UDRE	USART2 Data Register Empty
54	$006A[3]	USART2 TX	USART2 Tx Complete
55	$006C[3]	USART3 RX	USART3 Rx Complete
56	$006E[3])	USART3 UDRE	USART3 Data Register Empty
57	$0070[3]	USART3 TX	USART3 Tx Complete

Figura 3-12. Tabla de vectores de interrupción según prioridad

3.2.3. Cálculo de la inicialización de los Timers

Para el uso de los Timers, lo primero que debe programarse es su inicialización. En este apartado se explica la configuración de ambos temporizadores, tanto del Timer 1 como del 5.

3.2.3.1. Inicialización Timer 1

Lo primero que se debe hacer es configurar el Timer sabiendo que la interrupción debe producirse cada 10 ms (control PI). Es conocido que la frecuencia de la fuente de reloj es de 16 MHz. Teniendo en cuenta que el periodo que se busca es 10 ms, es decir una frecuencia de 100 Hz se sabe que habrá que modificar los valores por defecto del reloj ya existente en la placa de Arduino.

Primeramente se comienza cambiando el preescalador para intentar conseguir los 10 ms de periodo. Si no se encontraran se necesita hacer uso del valor CTC (valor máximo que se debe contar del modo de operación Output Compare, para configurar el periodo deseado), si se llega a este valor se activa la interrupción.

A continuación se calcula el divisor de frecuencia (el preescalador) teniendo en cuenta que el Timer 1 es de 16 bits. Se sabe que este puede contar hasta el valor 65536, por lo que el valor de CTC que se calcula para que pasen los 10 ms debe ser menor que el valor máximo que puede contar este Timer. Para que esto se cumpla se elige el preescalador de 8 cuyo periodo será el siguiente:

$$T = \frac{1}{16MHz/_8} = 5 \cdot 10^{-4} ms$$

Para configurar este preescalador se hace uso de la tabla mostrada en la Figura 3-9 donde se comprueba que CS11 debe valer uno y CS10 y CS12 valdrán cero.

Seguidamente se configura el modo de operación donde se debe definir el valor hasta el que se debe contar para conseguir que la interrupción se active cada 10 ms. Antes de esta configuración se observa que la interrupción se activaría cada 33 ms solo con el preescalador tal y como se comprueba en la ecuación:

$$T = (2^{16} - 1) \cdot (5 \cdot 10^{-4} ms) = 32.7675\ ms \approx 33\ ms$$

Por lo tanto se calcula el valor CTC para corregir esto ya que el tiempo deseado no son 33 ms sino 10 ms:

$$Valor\ CTC = \frac{tiempo\ deseado}{resolución\ Timer} - 1 = \frac{10ms}{5 \cdot 10^{-4} ms} - 1 = 19999$$

Para configurar este valor se almacenará el valor de CTC en el registro OCR1A. Para este Timer CTC será 19999.

Se debe tener en cuenta que esta inicialización se hará en el setup() del código, ya que es donde se inicializan las variables, registros, librerías, etc. Esta función solo se ejecutará una vez, bien porque se ha encendido la placa o bien porque se resetea y vuelve a inicializarse. En la Figura 3-13 se muestra el código de inicialización del Timer 1 que estará situado en el setup().

```
void setup() {
noInterrupts();          //desactiva todas las interrupciones para inicializar la interrupción temporal

 TCCR1A = 0;             //se inicializan los registros de control del Timer 1
 TCCR1B = 0;
 TCNT1  = 0;

 OCR1A =19999;           // valor CTC que se guarda en el registro OCR1A
 TCCR1B |= (1 << WGM12);   //configuración del modo output compare
 TCCR1B |= (1 << CS11);    //configuración del preescalador de 8
 TIMSK1 |= (1 << OCIE1A);  // activa la interrupción por comparación temporal

 interrupts();           // activa todas las interrupciones
}
```

Figura 3-13. Código de inicialización del Timer 1

3.2.3.2. Inicialización Timer 5

Este temporizador será usado para el código del control LQR. En este temporizador el intervalo de tiempo en el que salta es distinto, en este caso es cada 40 ms. La frecuencia de este temporizador es cuatro veces menor por lo que será 25 Hz. Se usa como referencia el reloj interno de la placa de Arduino que por defecto está configurado con una frecuencia distinta. Al igual que el temporizador anterior se debe configurar este para ajustar los 40 ms.

La inicialización empieza de la misma forma que en el temporizador anterior. Lo primero que se debe calcular es el periodo después de haber decidido cúal es el mejor preescalador. El periodo debe ser mayor que el requerido. Al igual que en el caso anterior este temporizador puede contar hasta 65536, por lo que el preescalador que podemos usar más pequeño es el de 64. Se obtiene el periodo calculado en la siguiente ecuación.

$$T = \frac{1}{16MHz/64} = 4 \cdot 10^{-3} ms$$

Si se eligiera un preescalador más pequeño se sobrepasaría el valor 65536 y saltaría el controlador antes de que pasaran lo 40 ms.

Para configurar este preescalador se hace uso de la tabla mostrada en la Figura 3-9 donde se comprueba que CS50 y CS51 deben valer uno y CS52 valdrá cero.

Seguidamente se configura el modo de operación donde se debe definir el valor hasta el que se debe contar para conseguir que la interrupción se active cada 40 ms. Antes de esta configuración se observa que la interrupción se activaría cada 262 ms solo con el preescalador tal y como se comprueba en la ecuación mostrada:

$$T = (2^{16} - 1) \cdot (4 \cdot 10^{-3} ms) = 262.14\, ms \approx 262\, ms$$

Por lo tanto se calcula el valor CTC para corregir esto ya que el tiempo deseado no son 262 ms sino 40 ms:

$$Valor\ CTC = \frac{tiempo\ deseado}{resolución\ Timer} - 1 = \frac{40ms}{4 \cdot 10^{-3} ms} - 1 = 9999$$

Para configurar este valor se almacenará el valor de CTC en el registro OCR5A. Para este Timer CTC será 9999.

En la Figura 3-14 se muestra el código de inicialización del Timer 5 cuya rutina de interrupción constará del control LQR.

```
void setup() {
noInterrupts();          //desactiva todas las interrupciones para inicializar la interrupción temporal

 TCCR5A = 0;             //se inicializan los registros de control del Timer 5
 TCCR5B = 0;
 TCNT5  = 0;

 OCR5A = 9999;           // valor CTC que se guarda en el registro OCR5A
 TCCR5B |= (1 << WGM52);  //configuración del modo output compare
 TCCR5B |= (1 << CS50);   //configuración del preescalador de 64
 TCCR5B |= (1 << CS51);   //configuración del preescalador de 64
 TIMSK5 |= (1 << OCIE5A); // activa la interrupción por comparación temporal

 interrupts();           // activa todas las interrupciones
}
```

Figura 3-14. Código de inicialización del Timer 5

3.2.4. Rutina de interrupción de los Timers

Como se ha mencionado antes, la prioridad es del control PI y por tanto al tener prioridad el Timer 1, este será el que realice dicho código, que será necesario cada 10 ms. El Timer 5 se encargará del control LQR que lo realizará cada 40 ms.

3.2.4.1. Rutina de interrupción del Timer 1

En esta rutina se pogramará el código correspondiente al cálculo del control PI para los motores. En la Figura 3-15 se encuentra el código de la rutina de interrupción.

```
ISR(TIMER1_COMPA_vect)        // rutina de interrupción con Output Compare
{
    e=r-acelf;
    ITerm += Ki*e;
    if((abs(ITerm))>255.0){
     if (ITerm>0) ITerm=255.0;
     else ITerm=-255.0;}

    u_m=Kp*e+ITerm;
}
```

Figura 3-15. Rutina de interrupción del Timer 1

3.2.4.2. Rutina de interrupción del Timer 5

En esta rutina se pogramará el código correspondiente al cálculo del control LQR. En la acción de control ya se ha añadido el término correspondiente al rechazo a perturbaciones y su sumatorio que se explicará en el último apartado del Capítulo 3. En la Figura 3-16 se encuentra el código de la rutina de interrupción.

En la acción de control se encuentran cuatro sumando, el primero hace referencia al ángulo ϕ, el segundo es su derivada. El tercer término hace referencia a la velocidad, consecuencia del ángulo θ y el último sumando es la perturbación. Esta ley de control será explicada en el último apartado de este capítulo donde se añade el término de la perturbación.

```
ISR(TIMER5_COMPA_vect)        // rutina de interrupción con Output Compare
{
        phi=kalAngleY*(PI/180);
        ivr=ivr+dtheta_r-wf;
        u=K1*(phi_r-phi) + K2*(dphi_r-dphi) + K3*(dtheta_r-wf) + K4*ivr;
        r=-u*c;
}
```

Figura 3-16. Rutina de interrupción del Timer 5

Como se ha mencionado antes los Timers elegidos han sido teniendo en cuenta tanto pines no usados como prioridad de la interrupción entre los Timers y entre Timer e interrupción externa de los encoders.

3.3. Mejora del funcionamiento del control PI

3.3.1. Concepto del controlador PID

Un controlador PID (Proporcional Integrativo Derivativo) es un mecanismo de control genérico sobre una realimentación de bucle cerrado, ampliamente usado en la industria para el control de sistemas. El PID es un sistema al que le entra un error calculado a partir de la salida deseada menos la salida obtenida y su salida es utilizada como entrada en el sistema que queremos controlar. El controlador intenta minimizar el error ajustando la entrada del sistema.

El controlador PID viene determinado por tres parámetros: el proporcional, el integral y el derivativo. Dependiendo de la modalidad del controlador alguno de estos valores puede ser nulo, en el caso de este Segway no se usará el término derivativo debido al ruido.

Cada uno de los parámetros influye en mayor medida sobre alguna característica de la salida (tiempo de establecimiento, sobreoscilación, etc.). No se puede conseguir un PID que reduzca todos estos incovenientes a cero, pero sí se puede ajustar de tal forma que se ajuste a las especificaciones requeridas.

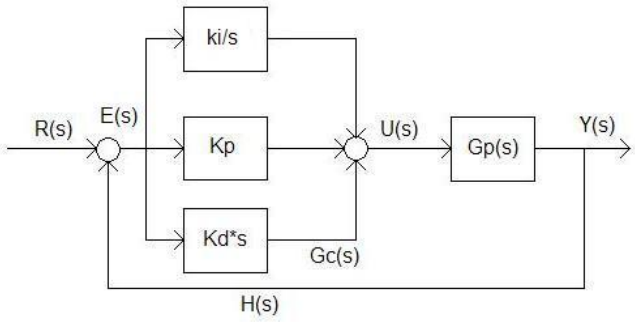

Figura 3-17. Control PID

La conocida fórmula usada para el control PID es:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt}$$

$$K_i = K_p \frac{T}{T_i}$$

$$K_d = K_p \frac{T_d}{T}$$

$$e(t) = \big(r - \theta(t)\big)$$

3.3.1.1. Acción proporcional

La respuesta proporcional es la base de los tres modos de control, si los otros dos, control integral y control derivatico están presentes, estos son sumandos a la respuesta proporcional. Este control se llama proporcional ya que la salida del controlador es un múltiplo del porcentaje del cambio en la medición. A este múltiplo se le llama ganancia del controlador.

3.3.1.2. Acción integral

La acción integral da una respuesta proporcional a la integral del error. Esta acción elimina el error en régimen estacionario, provocado por el modo proporcional. Sin embargo, esto provoca un mayor tiempo de establecimiento, una respuesta más lenta y el periodo de oscilación es mayor que en el caso de la acción proporcional. La ganancia de la acción integral es la ganancia integral.

3.3.1.3. Acción derivativa

La acción derivativa da una respuesta proporcional a la derivada del error (velocidad de cambio del error). Añadiendo esta acción de control a las anteriores se disminuye el exceso de sobreoscilaciones. Sin embargo, el control derivativo es muy sensible al ruido en este caso en la medida del ángulo, esto afecta produciendo oscilaciones de mucha frecuencia por lo que esta acción se descarta. La ganancia de la acción derivativa es la ganancia derivativa.

3.3.2. Método experimental de ajuste del PI

Este método consiste básicamente en el ajuste iterativo de los parámetros del controlador a partir de la observación de la respuesta temporal del sistema realimentado, y de la experiencia del equipo a las tendencias de las variables controladas en función de los parámetros que se quieren ajustar.

Para el análisis experimental del ajuste del PI se deben seguir unas reglas heurísticas de ajuste. Los pasos a seguir se explican a continuación.

3.3.2.1. Paso 1: Acción Proporcional

Lo primero que se debe calcular de forma aproximada experimentalmente es el término proporcional, para esto se debe poner el tiempo integral, T_i en su máximo valor ya que según la fórmula va diviendo y se consigue así anular este término.

En el caso de que hubiera tiempo derivativo, T_d, se usaría su valor mínimo también para anular el término.

En este primer paso se empieza con ganancia del controlador baja y esta se va aumentando hasta obtener las características de respuesta deseadas.

3.3.2.2. Paso 2: Acción integral

En este paso se deberá reducir hasta anular el error en estado estacionario, aunque la oscilación sea excesiva. Para poder ajustar mejor el controlador también se disminuye el valor de la ganancia, es decir K_p, ligeramente. Esto se repetirá de forma iterativa hasta obtener las características de respuesta deseadas.

3.3.2.3. Paso 3: Acción Derivativa

En el caso de necesitar este término, la acción derivativa se ajusta manteniendo la ganancia y el tiempo integral obtenidos anteriormente. Por otra parte se aumenta el T_d hasta obtener características similares pero con la respuesta más rápida. Si fuera necesario se puede aumentar ligeramente la ganancia del controlador, K_p.

3.3.2.4. Cambio en las características del sistema

En la Tabla 3-1 encontramos las características que se verán afectadas si se aumenta K_p, se disminuye T_i y se aumenta T_d. Realizando estos cambios según los pasos antes descritos habrá características del sistema que cambien.

Tabla 3–1 Características que se ven afectadas al ajustar el controlador

Característica	K_p aumenta	T_i disminuye	T_d aumenta
Estabilidad	Se reduce	Disminuye	Aumenta
Velocidad	Aumenta	Aumenta	Aumenta
Error estacionario	No eliminado	Eliminado	No eliminado
Área del error	Se reduce	Disminuye	Se reduce
Perturbación control	Aumenta bruscamente	Aumenta gradualmente	Aumenta bruscamente
Frecuencia lazo	No afecta hasta cierto punto	Disminuye	Aumenta

3.3.2.5. Desventajas de este método

Este método presenta también algunas desventajas en los ensayos específicos necesarios para este proyecto, entre ellas:

• Consume mucho tiempo, además de las consiguientes pérdidas de productividad del sistema.

• El hecho de llegar a un comportamiento cíclico continuo es objetable pues pone al sistema en el límite de la estabilidad, pudiendo inclusive provocar situaciones de riesgo.

• No es aplicable a procesos que son inestables en bucle abierto porque dichos procesos suelen ser inestables para altos o bajos valores de la ganancia.

3.3.3. Ensayos realizados para el ajuste del PI

Para el ajuste del control PI, se realizarán una serie de ensayos. Las gráficas que se muestran han sido realizadas según el método de ajuste por ensayo y error, se seguirá el método antes explicado. Para obtener estas gráficas se ha hecho uso de los datos obtenidos por Bluetooth. Estos datos serán usados en Matlab para poder sacar las gráficas donde se van a trazar la aceleración real y la referencia de la misma.

Para la realización de los ensayos se cambiará el código del control LQR donde la acción de control, es decir la entrada como referencia del control PI, será siempre constante y dado un intervalo de tiempo habrá un cambio en esta referencia. Este cambio de referencia será desde un valor de -2 a 2 donde habrá un transitorio en el que la rueda deberá cambiar el sentido de giro.

Para la realización del cambio de referencia se realiza una cuenta del tiempo. Se sabe que el temporizador 1, el usado para el control LQR, realiza la acción cada 40 ms, si se desea que a los 2000 ms cambie la referencia, la cuenta que se realizará debe llegar al valor 50.

A continuación se muestra en la Figura 3-18 la definición e inicialización de la variable *cuenta1* que sirve para contar las veces que se entra en el bucle y por lo tanto contabilizar el tiempo antes mencionado de 2000 ms.

```
/*Definición*/
int cuenta1;

/*Inicialización*/
```

```
cuenta1=0;
```

Figura 3-18. Código para la definición e inicialización de la variable *cuenta1*

El código usado para cambiar la referencia se muestra en la Figura 3-19. Este código ha sido sustituido por el que había anteriormente del control LQR.

```
ISR(TIMER1_COMPA_vect)  //Timer encargado del cálculo del control LQR
{
        cuenta1=cuenta1+1;
        if (cuenta1<50){
        r=-2;}
        if (cuenta1>=50) r=2;
}
```

Figura 3-19. Código para el cálculo de la referencia para la aceleración

En el código del valor de las constantes del control PI cambiará solo en el valor de las constantes K_p y K_i, en cada ensayo que se realice, para el ajuste del controlador.

Únicamente queda adaptar el código del envío por Bluetooth donde lo que interesa es el valor de la aceleración real, el valor de la referencia, el error y el instante de tiempo en el que se calcularon estas variables.

Es importante que una vez escrito el valor de la variable se ponga una coma seguidamente y en el último valor un punto y coma. Esto se necesita para poder crear una matriz en Matlab con estos valores y trazar así las gráficas. Se puede observar este código en la Figura 3-20.

```
void EnvioBT(){
  Serial.print(acelf); Serial.print(",");
  Serial.print(e); Serial.print(",");
  Serial.print(r);Serial.print(",");
  Serial.print(millis());Serial.println(";");
delay(1);
}
```

Figura 3-20. Código para el envío por Bluetooth

La primera gráfica que se muestra son los resultados usando los valores de K_p y K_i obtenidos para el antiguo proyecto. Se comprueba que el error es muy grande y que no se estabiliza. También se puede observar que la señal tiene mucho ruido, ya que la oscilación de los valores obtenidos parece permanecer constante. Por lo tanto se sabe que se podrá mejorar con el ajuste adecuado el controlador pero el ruido no se podrá eliminar.

En la Figura 3-21 se encuentra dicha gráfica con $K_p = 0.2$ y $K_i = 0.1$ en los valores del controlador. El valor de la constante del tiempo integral se empieza igualando al valor de la constante de tiempo del ensayo ante escalón y el valor de la ganancia proporcional es experimental. En todas la gráficas que se muestra, una gráfica azul que es la aceleración real y una función roja que es la referencia de la aceleración.

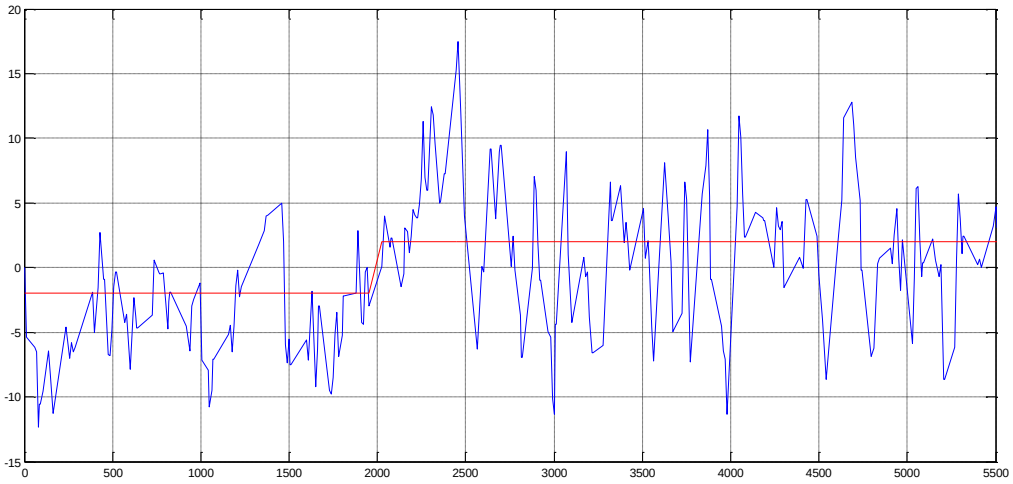

Figura 3-21. Ensayo con los valores del PI usados anteriormente

El error en ciertos puntos llega a sobrepasar un valor de 10, por lo que este controlador no es realmente eficaz.

Para comenzar con el ajuste lo primero que se realiza es el aumento de la ganancia proporcional que en un primer ensayo será $K_p = 0.25$.

A continuación se empieza a disminuir K_i, es decir aumentar T_i . Esta ganancia se disminuye hasta un valor que permita el equilibrado del sistema. En consecuencia, de forma experimental se obtiene $K_i = 0.08$ para una ganancia del controlador de 0.25.

En la Figura 3-22 se observa la nueva gráfica donde se comprueba la disminución del error que no suele sobrepasar de 5, sin embargo todavía se podría ajustar más el controlador si el ruido lo permite.

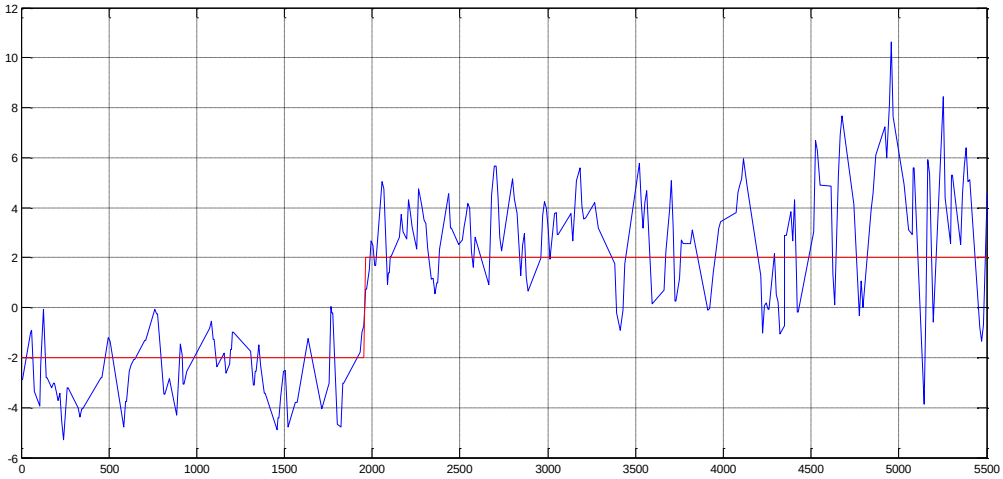

Figura 3-22. Ensayo con $K_p = 0.25$ y $K_i = 0.08$

En el siguiente ensayo se busca un control menos agresivo, es decir una respuesta más lenta. Para ello la ganancia proporcional se disminuye en cierta medida, al igual que la integral. Se consigue así una mejora en el error, aunque este sigue llegando en momentos muy puntuales a un valor de 5.

Esto se puede comprobar en la Figura 3-23 donde $K_p = 0.22$ y $K_i = 0.07$.

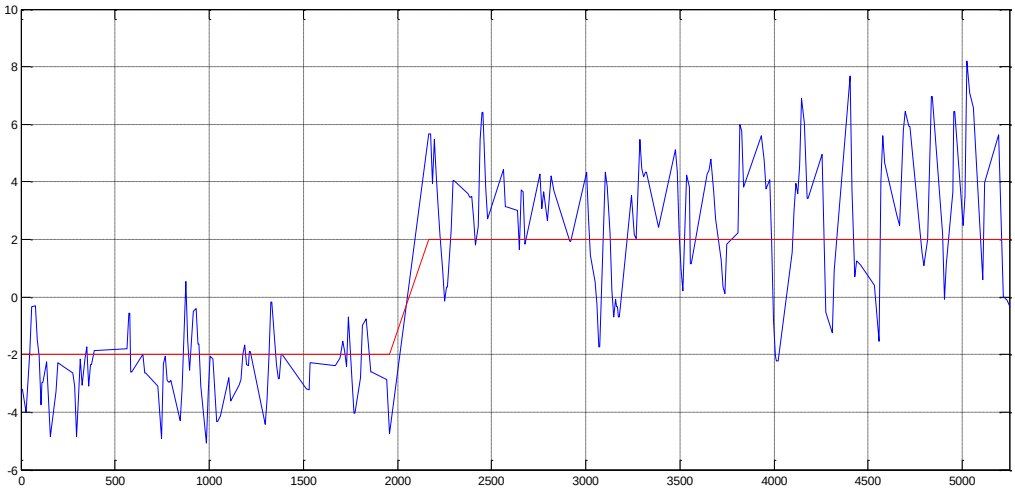

Figura 3-23. Ensayo con $K_p = 0.22$ y $K_i = 0.07$

En el siguiente experimento se realiza el cambio únicamente en la ganancia integral que se disminuye provocando que el sistema realice el control de una forma más suave. Es por ello que el valor de la ganancia proporcional permanece siendo el mismo. En este caso K_i será igual a 0.06.

Se puede observar la gráfica en la Figura 3-24. Cabe destacar que se disminuye el error que ya nunca supera una diferencia de 5 con respecto a la referencia en la aceleración, y que su media está entorno a 2.

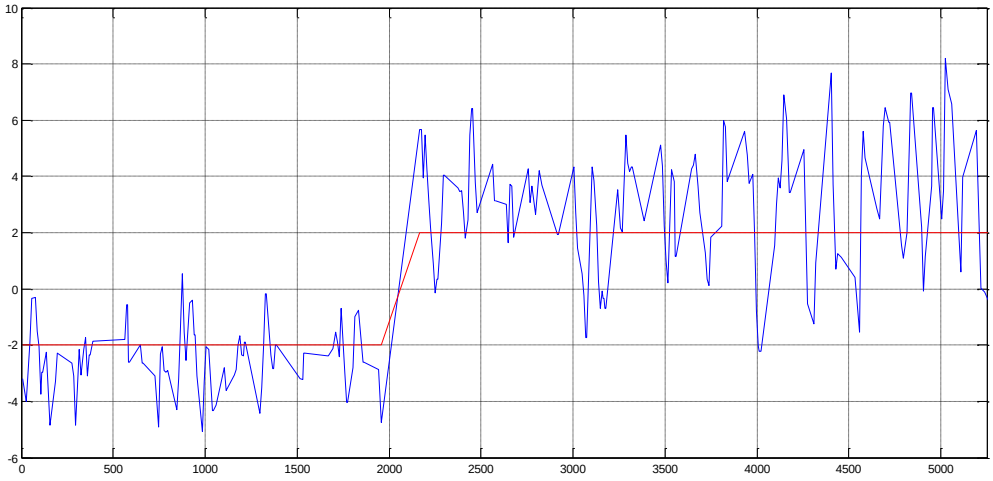

Figura 3-24. Ensayo con $K_p = 0.22$ y $K_i = 0.06$

Finalmente, se realiza un último ensayo donde el controlador sea más rápido y para ello se deberá cambiar la ganancia proporcional aumentándola. En este ensayo permanece el mismo valor de la ganancia integral y ahora K_p será 0.3.

Se comprueba el resultado en la Figura 3-25. Se puede observar que el error disminuye ligeramente. El error en esta gráfica no suele superar tampoco un valor máximo de 2 con respecto a la referencia.

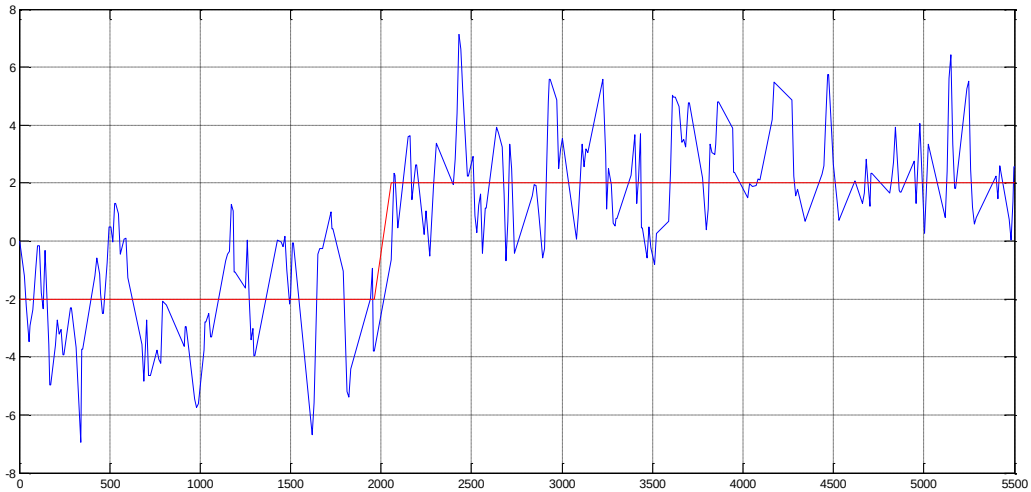

Figura 3-25. Ensayo con $K_p = 0.3$ y $K_i = 0.06$

En conclusión, solo queda realizar un ensayo donde se puedan ver más cambios de referencia una vez obtenidos los valores para el ajuste del PI. En este ensayo se comienza con una aceleración de -2, a los 3000 ms se cambia la referencia a 2. A continuación se realiza otro cambio en la referencia que tenrá el valor de -3 y se finaliza la gráfica con una referencia de 3. Este ensayo se muestra en la Figura 3-26.

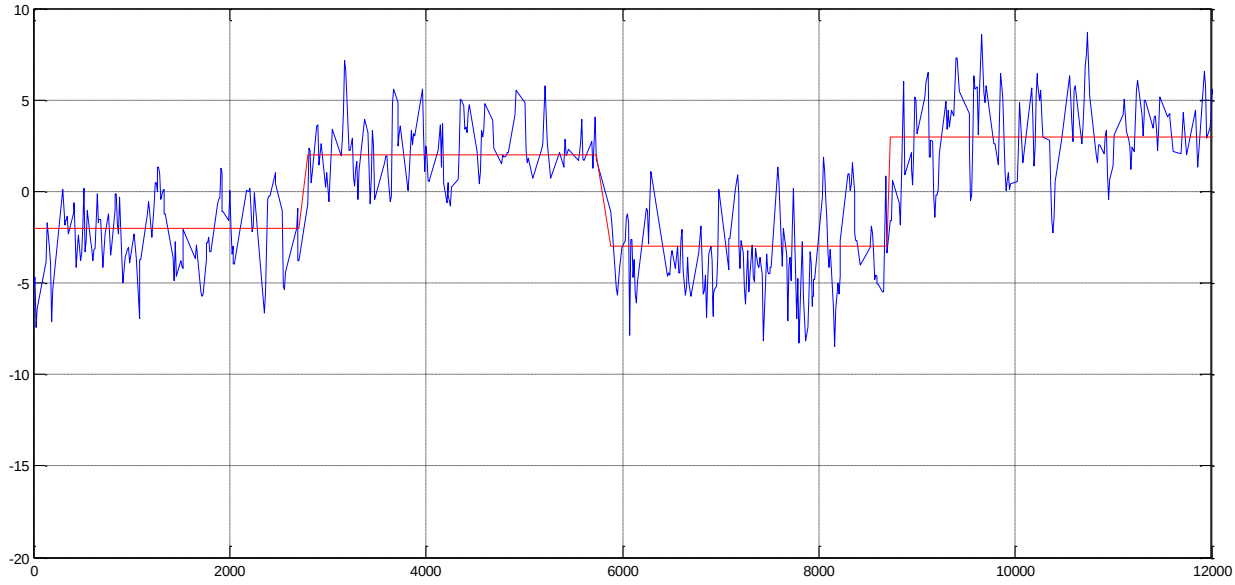

Figura 3-26. Ensayo con cuatro cambios de referencia

Se puede observar que el ruido se ve afectado a mayor valor, es decir si la referencia es de 2, la aceleración tendrá menos ruido que en la referencia de 3.

3.4. Rechazo de las perturbaciones originadas por el centro de gravedad

3.4.1. Obtención del nuevo sistema a controlar

Uno de los tipos de control usados en este proyecto es el LQR (Linear Quadratic Regulator) aplicado en sistemas definidos por un conjunto de ecuaciones diferenciales lineales cuyo objetivo es lograr un control óptimo con el mínimo coste. La función que minimiza este control se muestra a continuación:

$$J = \int_0^\infty (x^T Q x + u^T R u) dt$$

donde Q y R son matrices que ponderan el error en el estado y la señal de control respectivamente. El vector de estado es x y está descrito en el espacio de estados por:

$$\dot{x} = Ax + Bu$$
$$y = Cx$$

El control LQR consiste finalmente en la realimentación del vector de estados obteniéndose la siguiente señal de control:

$$u = -Kx$$

Este planteamiento fue utilizado inicialmente, sin embargo debido a la diferencia entre la posición del centro de gravedad y el centro geométrico, se considerará que en el sistema existe una perturbación. Esto implica que se deberá añadir un sumando en las ecuaciones del sistema en espacio de estados, que tenga en cuenta así la perturbación. Por lo tanto el nuevo sistema de ecuaciones será:

$$\dot{x} = Ax + Bu + Ew$$
$$y = Cx$$

A continuación se discretiza la primera ecuación. Se sabe además que todas las variables de estado son accesibles, es decir se pueden medir directamente sin que se tengan que estimar por otros procedimientos. Dado que es medible no se necesita observador. La ecuación discretizada será:

$$x_{k+1} = Ax_k + Bu_k + Ew_k$$

donde Ew_k es la perturbación, es decir $P_k = Ew_k$ siendo *w* una constante desconocida.

Se calcula el nuevo sistema con el modelo en variables incrementales con el que se va a trabajar:

$$x_{k+1} - x_k = A(x_k - x_{k-1}) + B(u_k - u_{k-1}) + E(w_k - w_{k-1})$$

Por tanto se reformula el sistema partiendo de las siguientes variables incrementales, que se usarán en el sistema con el modelo en variables incrementales:

$$x_{k+1} - x_k = \Delta x_{k+1}$$
$$x_k - x_{k-1} = \Delta x_k$$
$$u_k - u_{k-1} = \Delta u_k$$
$$w_k - w_{k-1} = \Delta w_k$$

Se obtiene una nueva ecuación en variables incrementales, sustituyendo las diferencias entre varibles por incrementos:

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

donde la variación de la perturbación es $\Delta P_k = E\Delta w_k$.

Es conocido que la perturbación es constante, por lo que su variación será nula, es decir:

$$\Delta P_k = 0$$

Por tanto la ecuación en variables incrementales se simplifica quitando el término nulo y se obtiene así la primera ecuación del sistema:

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

El diagrama de bloques que se obtiene añadiendo la perturación en el modelo de Simulink se muestra en la Figura 3-27.

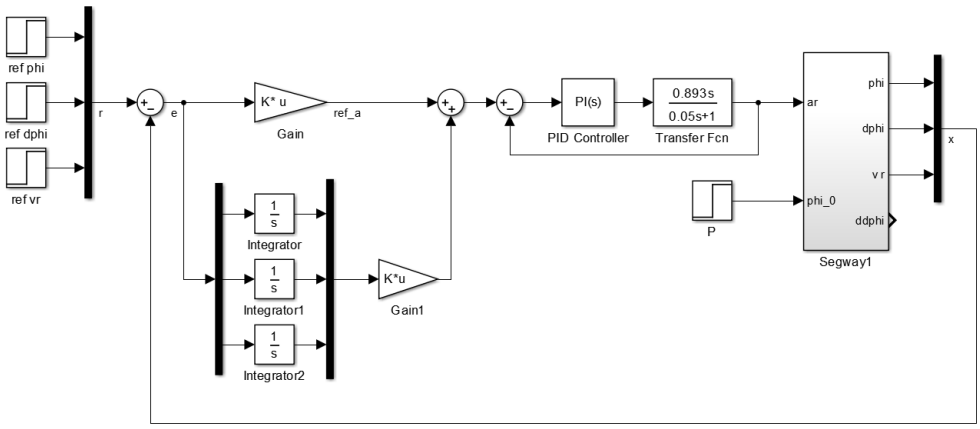

Figura 3-27. Diagrama de bloques con la perturbación

En la Figura 3-28 se encuentra el diagrama de bloques del bloque que aparece con el nombre de Segway1 del diagrama anterior, cuyas entradas son la aceleración y la perturbación. Este diagrama devuelve en la salida el ángulo ϕ, su derivada y la velocidad de la rueda.

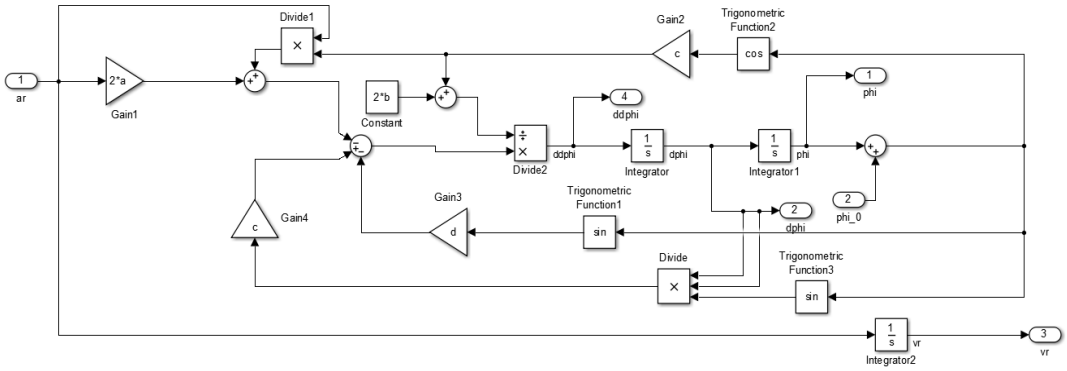

Figura 3-28. Diagrama de bloques del Segway

Se sabe según el diagrama de bloques mostrado que el error (e) es la diferencia entre la referencia (r) de los valores de las variables de estado y la salida del sistema (x) que son los valores de las variables de estado en la salida. Para que el error sea nulo la referencia y la salida deben ser iguales. La fórmula del error con el sistema discretizado es:

$$e_{k+1} = r_{k+1} - x_{k+1}$$

Si se considera que la referencia no cambia, es decir es un valor constante en cada una de las variables, usando el sistema en variables incrementales se obtiene que la variación de la referencia es nula.

$$\Delta r_{k+1} = r_{k+1} - r_k = 0$$

Utilizando las dos fórmulas anteriores y considerando el error y el vector de estados en variables incrementales se concluye en la siguiente fórmula:

$$\Delta e_{k+1} = \Delta r_{k+1} - \Delta x_{k+1} = -\Delta x_{k+1}$$

siendo $\Delta e_{k+1} = e_{k+1} - e_k$ y por tanto queda la siguiente ecuación:

$$e_{k+1} - e_k = -\Delta x_{k+1}$$

Ya que se busca la fórmula del error en k se obtiene esta fórmula:

$$e_k = e_{k-1} - \Delta x_k$$

Por tanto se deduce así el sistema completo quedando el siguiente resultado:

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

3.4.2. Cálculo del controlador LQR

A partir del sistema obtenido, se diseñará un controlador en variables incrementales cuya fórmula es:

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

donde K_x es las constante del control LQR y K_e la constante que multiplica al error.

Por último, ya que lo que interesa es la ecuación sin variables incrementales, a través del método inductivo se podrá conseguir dicha fórmula de la acción de control. Se considera que inicialmente la acción de control es nula, es decir que la condición inicial es:

$$u_0 = 0$$

Para el cálculo de u_k, se sustituye en la fórmula del controlador hasta encontrar un patrón del que se logrará tener la ley de control.

Se sustituye en la fórmula con k=1:

$$u_1 - u_0 = K_x(x_1 - x_0) + K_e e_0$$
$$u_1 = u_0 + K_x(x_1 - x_0) + K_e e_0$$

Se sustituye con k=2:

$$u_2 - u_1 = K_x(x_2 - x_1) + K_e e_1$$
$$u_2 = u_1 + K_x(x_2 - x_1) + K_e e_1$$

Se sustituye u_1 utilizando la ecuación obtenida en k=1:

$$u_2 = u_0 + K_x(x_1 - x_0) + K_e e_0 + K_x(x_2 - x_1) + K_e e_1$$

Se ordena la fórmula según el tipo de variables:

$$u_2 = u_0 + K_x(x_1 - x_0) + K_x(x_2 - x_1) + K_e e_0 + K_e e_1$$

Finalmente se obtiene u_2 agrupando:

$$u_2 = u_0 + K_x(x_2 - x_0) + K_e(e_0 + e_1)$$

Se supone la fórmula cierta para u_{n-1} siendo por tanto:

$$u_{n-1} = u_0 + K_x(x_{n-1} - x_0) + K_e \sum_{i=0}^{n-2} e_i$$

Y para calcular u_n se aplica la fórmula para k=n:

$$u_n - u_{n-1} = K_x(x_n - x_{n-1}) + K_e e_{n-1}$$
$$u_n = u_{n-1} + K_x(x_n - x_{n-1}) + K_e e_{n-1}$$

Se sustituye u_{n-1} obtenido anteriormente:

$$u_n = u_0 + K_x(x_{n-1} - x_0) + K_e \sum_{i=0}^{n-2} e_i + K_x(x_n - x_{n-1}) + K_e e_{n-1}$$

Finalmente se obtiene u_n agrupando, quedando así demostrada la fórmula:

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

3.4.3. Ensayos en Simulink

Previamente se prueba la nueva acción de control en Simulink, se hará uso del diagrama de bloques mostrado en la Figura 3.27 que se ha mostrado anteriormente. En este diagrama de bloques se añade la perturbación y el controlador debe ser capaz de provocar que cada una de las variables alcance la referencia. En las siguientes figuras se demuestra que cada una de las variables alcanza la referencia a pesar de la perturbación y es capaz de controlar el sistema.

El sistema alcanza el valor deseado del ángulo phi tal como se comprueba en la Figura 3-28 a pesar de la perturbación provocada por la diferencia entre el centro de gravedad y el centro geométrico. Ya que el Segway tendía a ir siempre hacia delante porque el centro de gravedad estaba desplazada hacia esta posición.

Además según las gráficas existe una respuesta en régimen transitorio bastante rápida. En las gráficas cabe destacar que la referencia es la curva azul y la salida es la roja.

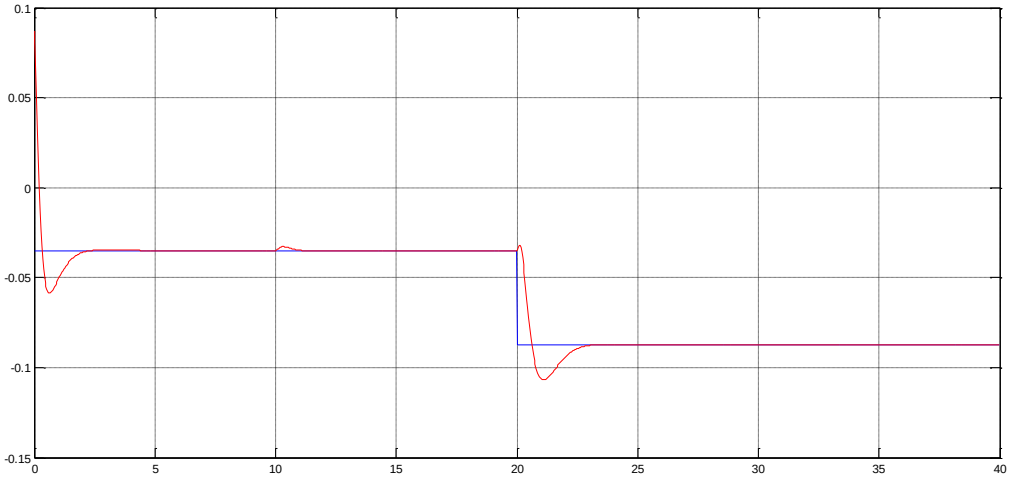

Figura 3-29. Phi frente a tiempo

También alcanza el valor deseado de la variación del ángulo en el tiempo, es decir el valor deseado de la derivada de phi, mostrado en la Figura 3-29.

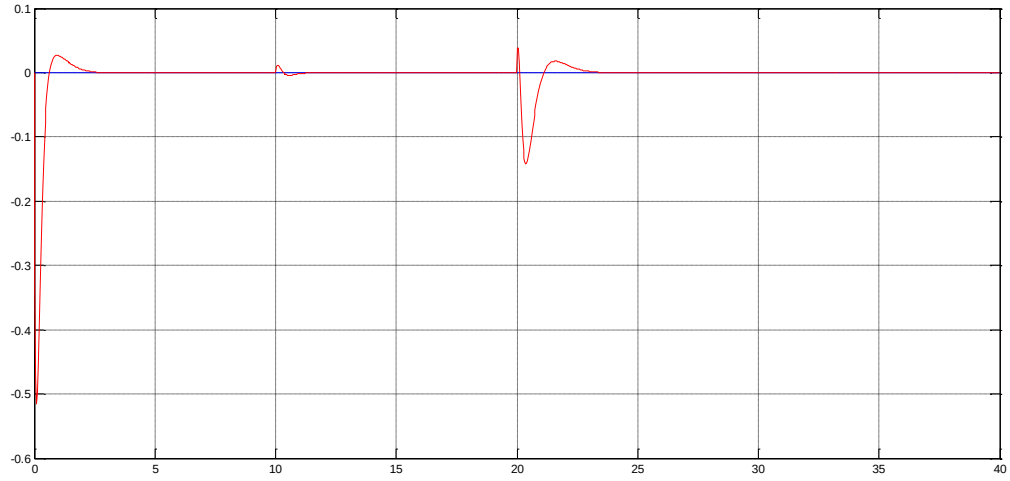

Figura 3-30. Derivada de phi frente a tiempo

Y logra alcanzar también el valor deseado de la velocidad de la rueda tal como se demuestra en la Figura 3-30 donde existe una perturbación a los 20 ms.

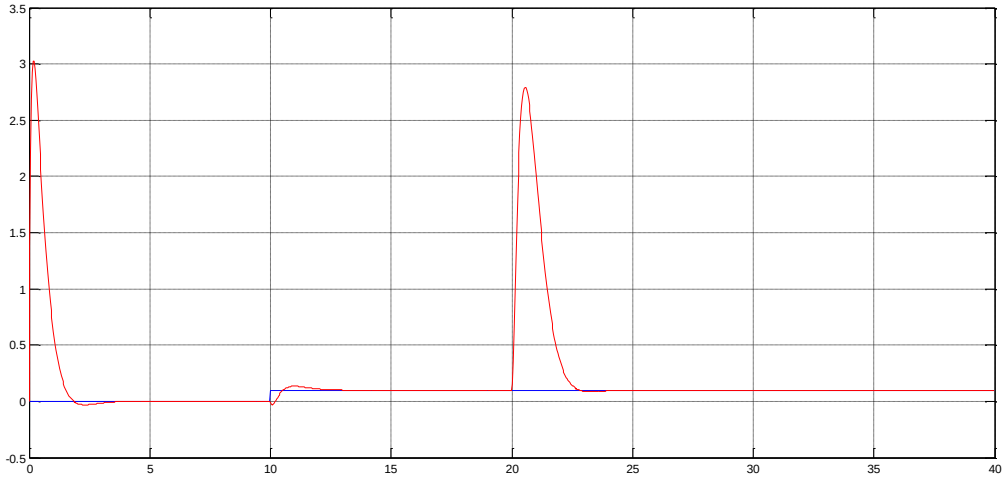

Figura 3-31. Velocidad de la rueda frente a tiempo

Se comprueba que los transitorios tanto de phi como de su derivada no son muy pronunciados a diferencia del transitorio de la velocidad de la rueda aunque se demuestra que estos transitorios son muy rápidos y que se alcanza en poco tiempo el régimen permanente.

Únicamente falta por calcular la constante del error, es decir K_e. En el ensayo antes mostrado realizado con el modelo en Simulink esta constante es:

$$K_e = 10$$

Esta constante solo puede ajustarse experimentalmente. Primeramente se simula el sistema con la perturbación en Simulink para comprobar que el resultado obtenido de la acción de control es válido. Y una vez comprobado los buenos resultados de la simulación, se debe calcular experimentalmente, con ensayos de prueba y error, la constante del error en el sistema real, es decir en el propio vehículo.

3.4.4. Cálculo de K_e en el sistema real e implementación del nuevo controlador

En la implementación del código lo primero será añadir el sumatorio del error en la acción de control LQR, es decir la integral de la diferencia de la velocidad de la rueda medida con la calculada. La acción de control se encuentra dentro de la rutina de interrupción del Timer 1. En la programación este sumatorio se realiza antes de calcular la ley de control, se suma por tanto la diferencia de velocidades (calculando así el error) y el error antes acumulado. Se debe tener en cuenta que en la inicialización de variables el sumatorio del error es cero.

Por otra parte, este sumatorio va multiplicado por una constante que según los ensayos en Simulink debe ser igual a 10, sin embargo realizando pruebas en el segway, este valor es demasiado alto. Ajustanto experimentalmente el valor de la constante, es decir disminuyéndolo en este caso, se obtiene 0.3 como valor con el que el vehículo consigue estabilizarse. Fuera de este valor el vehículo intenta o estabilizarse demasiado rápido si el valor es más alto y por tanto termina cayéndose o se intenta estabilizar demasiado lento si el valor es menor lo que provoca que se caiga igualmente. Este valor no es el mismo que en los ensayos de Simulink ya que se debe tener en cuenta que estos ensayos son con un sistema en tiempo continuo. Sin embargo, los ensayos realizados en el Segway son en tiempo discreto, además de que esta ganancia siempre se tiene que ajustar en un sistema real.

En la Figura 3-31, 3-32 y 3-33 se muestra el código implementado en el sistema. En la Figura 3-31 se muestra la inicialización de las variables.

```
void setup() {
///CONTROLADOR////
phi_r=0; dphi_r=0; dtheta_r=0;
u=0;
Kp=0.3;
Ki=0.06;
K1=-332.8746; K2=-52.7835; K3=-7; c=2.0; Dir=5.0; xg=1.0; xt=1.0; K4=-0.3; ivr=0;
//////////////////////////////
}
```

Figura 3-32. Código de inicialización de las variables del controlador

La constante que multiplica el error que teóricamente se le dio el nombre de K_e, en la programación del código es K_4. El sumatorio antes descrito se calcula mediante el uso de la variable ivr, que como aparece en el código se inicializa dándole un valor de cero.

En la Figura 3-32 se observa el código del controlador donde se añade el sumando $K_4 \cdot ivr$ a la ley de control LQR. Es importante programar antes el cálculo del sumatorio del error, donde se irá acumulando.

```
ISR(TIMER1_COMPA_vect)
{
        phi=kalAngleY*(PI/180);
        ivr=ivr+dtheta_r-wf;   //sumatorio del error
        u=K1*(phi_r-phi) + K2*(dphi_r-dphi) + K3*(dtheta_r-wf) + K4*ivr;
        r=-u*c;
}
```

Figura 3-33. Código del controlador en el Timer 1

Finalmente, en la Figura 3-33 se muestra el código añadido dentro del código de mando del motor, donde al superar el límite en el que el vehículo se equilibra, se paran los motores y se vuelve a inicializar el sumatorio del error, ivr.

```
void Mando_Motor(){
if (abs(kalAngleY)>30){
   PWMI=0;
   PWMD=0;
   ivr=0;}
}
```

Figura 3-34. Código de inicialización si el vehículo se encuentra fuera del rango de uso

Por último, se realizan dos ensayos. En uno de ellos se añade una carga descentrada una vez se ha estabilizado el vehículo. Esto provoca un cambio en el ángulo ϕ tal y como se muestra en la siguiente gráfica. Esto es debido al cambio del centro de gravedad del Segway que al no ser su centro geométrico, el vehículo tendrá un ángulo distinto de cero en la posición de reposos. La Figura 3-35 muestra la gráfica del ángulo. Se comprueba que aproximadamente a los 11000 ms se coloca la carga descentrada y el vehículo vuelve a estabilizarse.

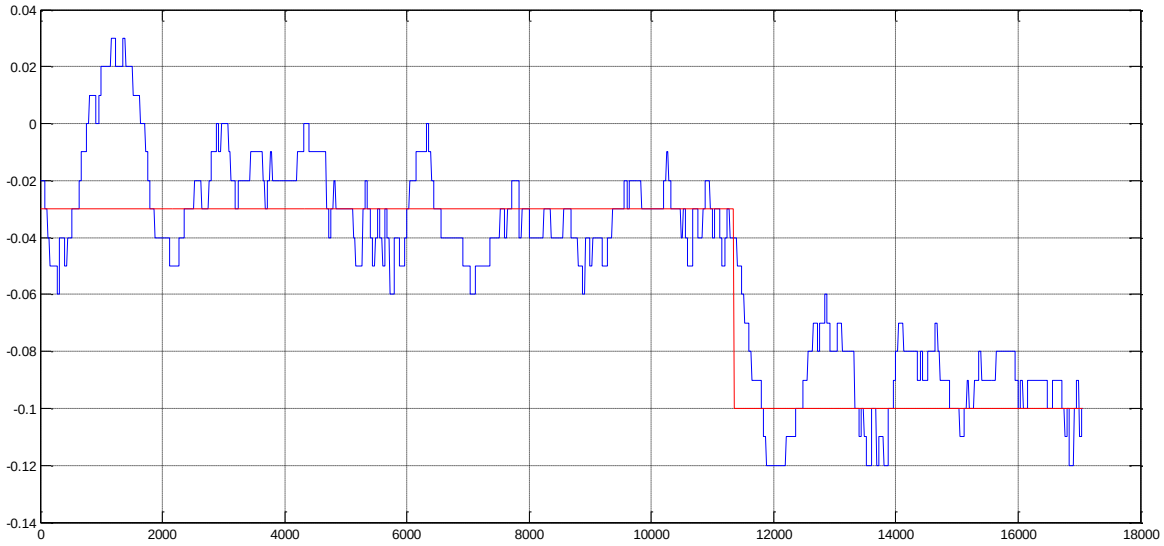

Figura 3-35. Ensayo con la carga descentrada, gráfica de ϕ frente a tiempo

A continuación se muestra la derivada del ángulo phi en la Figura 3-36. Se comprueba que la señal tiene ruido aunque el error es pequeño y nunca alcanza a 0.5 en el régimen permanente.

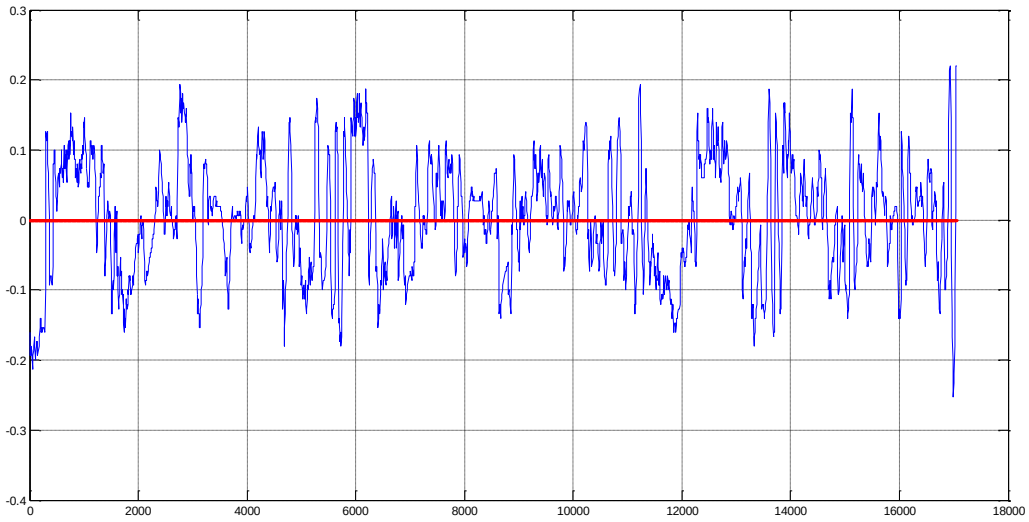

Figura 3-36. Ensayo con la carga descentrada, gráfica de derivada de ϕ frente a tiempo

Se puede comprobar el ruido en la velocidad, provocado no solo por la señal sino también por la propia estructura y la holgura de la propia rueda. Esta no permite que el vehículo permanezca completamente parado. Se puede ver el ensayo en la Figura 3-37.

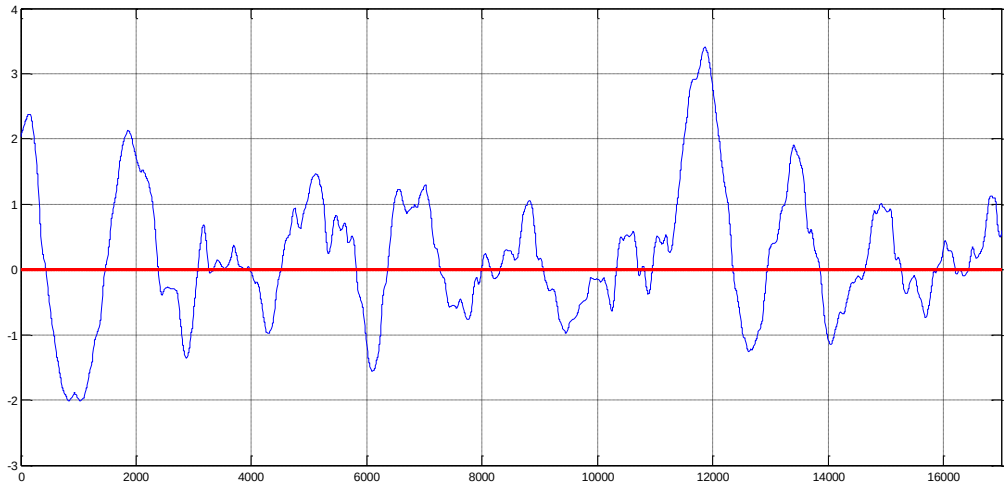

Figura 3-37. Ensayo con la carga descentrada, gráfica de velocidad de la rueda frente a tiempo

Solo queda añadir el segundo ensayo que se realiza, en el que habrá un cambio en la referencia de la velocidad de la rueda. En la Figura 3-38, se muestra la respuesta en velocidad obtenida. El cambio de referencia es de -2 a 2, lo que implica que el vehículo tenga que desplazarse para adelante y atrás.

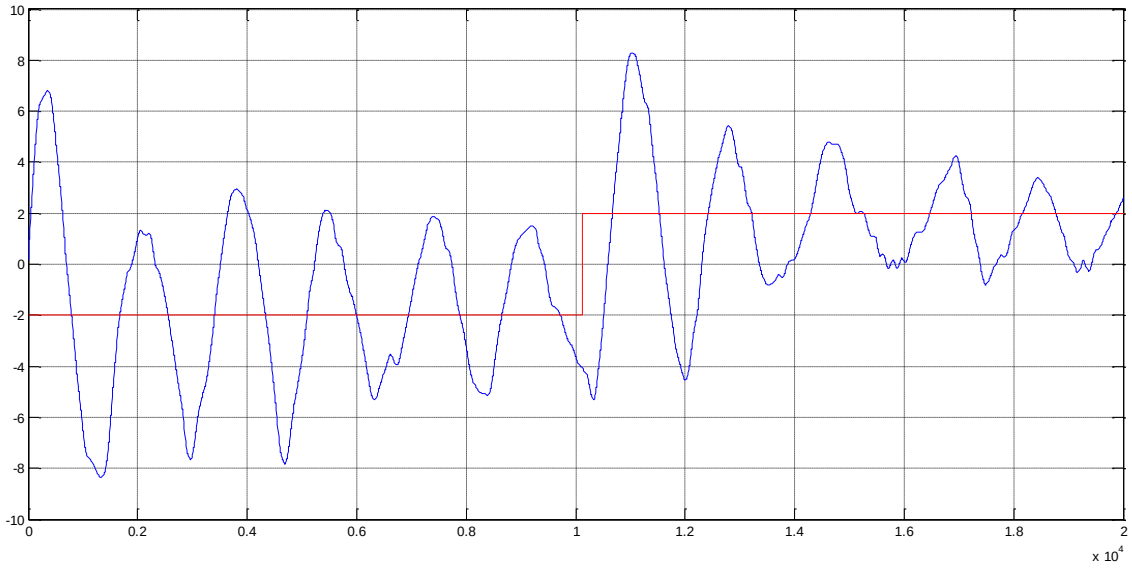

Figura 3-38. Ensayo con cambio en la velocidad, gráfica de velocidad de la rueda frente a tiempo

En la Figura 3-39 se muestra el ángulo ϕ y en la Figura 3-40 se muestra su derivada, comprobamos que con el cambio de referencia se provoca un pequeño transitorio en cada una.

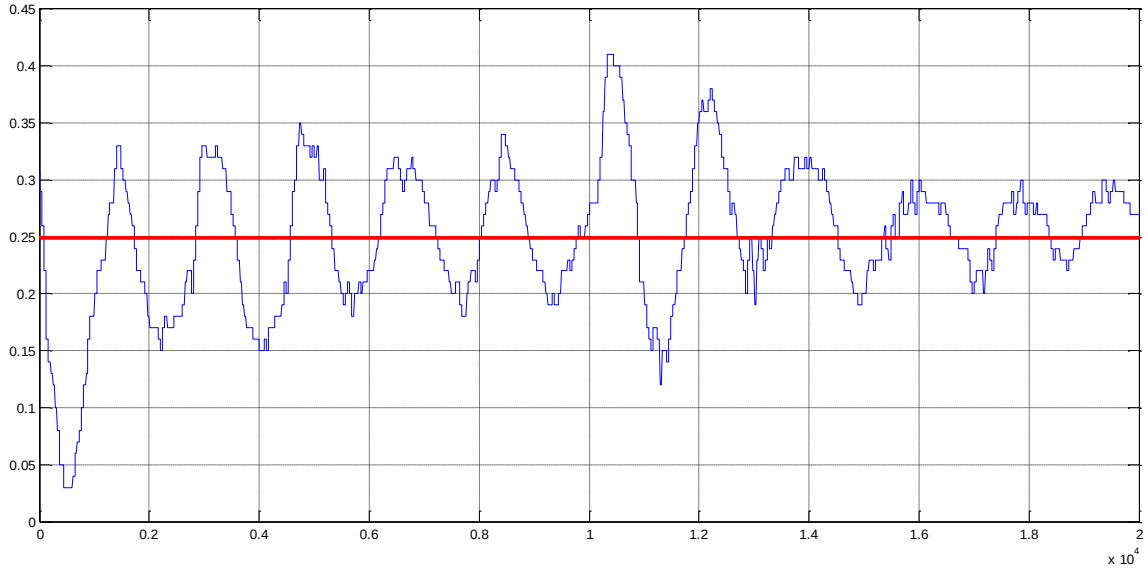

Figura 3-39. Ensayo con cambio en la velocidad, gráfica de ϕ frente a tiempo

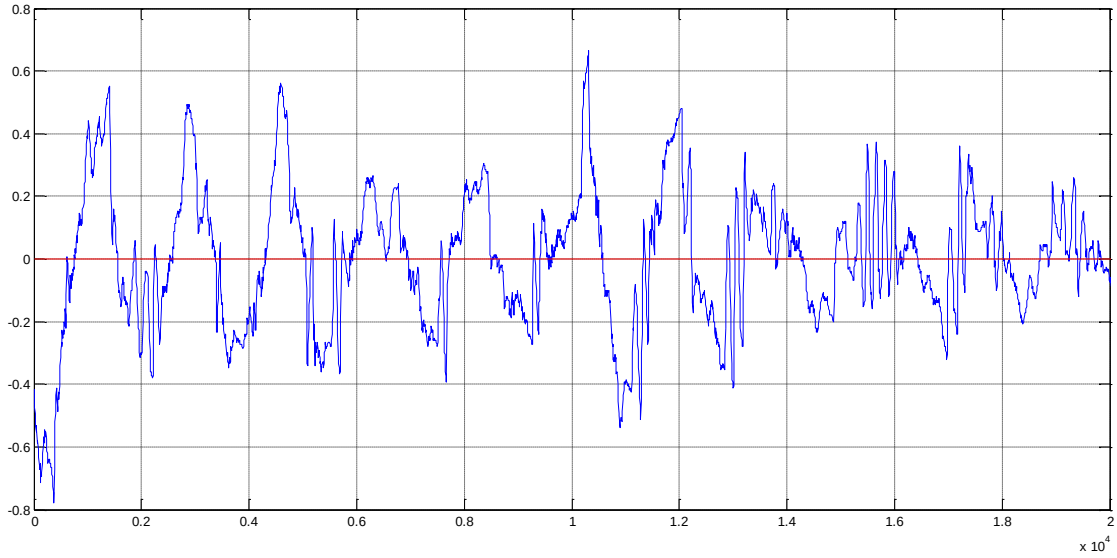

Figura 3-40. Ensayo con cambio en la velocidad, gráfica de la derivada de ϕ frente a tiempo

4. CONCLUSIONES

Durante la realización de este proyecto de mejora del funcionamiento del vehículo, se han podido comprobar las diferencias entre resultados teóricos de simulaciones y prácticos, ya que para el óptimo funcionamiento se han tenido que ajustar parámetros.

Es destacable la imperfección de la calidad de las señales, lo que provoca el ruido en la señal de medida de la velocidad y la aceleración, como se comprueba en el ruido residual en el ajuste del control PI. También se puede remarcar la falta de precisión de los motores debido a la holgura de la caja reductora, inherente en componentes de bajo coste.

Se ha conseguido mejorar la precisión del intervalo de tiempo que tarda el microcontrolador en calcular los controladores, por el uso correcto de interrupciones internas temporales.

Otro punto a destacar es el hecho de que el vehículo ha dejado de quedarse pillado por el mal funcionamiento de la IMU. Y aunque se ha conseguido solventar este problema, se podría realizar un estudio del tiempo máximo que necesita la IMU para leer correctamente los valores. Lo que realiza ahora es una lectura continua, que si falla no hace que el Segway deje de funcionar como estaba antes programado, por el contrario sigue funcionando aunque con una mala lectura en el punto de fallo.

REFERENCIAS

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

[2] Universidad de Huelva, Curso práctico online de Arduino avanzado

[3] Arduino, Libro de Proyectos de Arduino

[4] InvenSense, DataSheet MPU 6000 and MPU 6050 Product Specification Revision 3.4

[5] Varios, Apuntes de Ingeniería de Control, Universidad de Sevilla, 2005

[6] http://robologs.net/2014/10/15/tutorial-de-arduino-y-mpu-6050/

[7] http://hetpro-store.com/TUTORIALES/modulo-acelerometro-y-giroscopio-mpu6050-i2c-twi/

[8] http://forum.arduino.cc/index.php?topic=54782.0

[9] https://www.arduino.cc/en/Main/ArduinoBoardMega2560

[10] https://arduino-info.wikispaces.com/Timers-Arduino

[11] http://revistas.javeriana.edu.co/index.php/iyu/article/viewFile/1820/5216

[12] http://www.circuitoselectronicos.org/2011/04/el-temporizador-timer-0-en-los.html

[13] http://www.circuitoselectronicos.org/2011/04/temporizador-timer-1-en-los.html

[14] http://www.circuitoselectronicos.org/2011/04/temporizador-timer-2-en-los.html

[15] http://www.prometec.net/timers/

[16] http://picfernalia.blogspot.com.es/2012/06/interrupciones-conceptos-basicos.html

[17] http://gammon.com.au/interrupts

[18] https://rheingoldheavy.com/changing-the-i2c-library/

[19] http://dummyscodes.blogspot.com.es/2014/07/mpu6050-gy-521-breakout-arduino-mega.html

[20] https://geekytheory.com/matlab-arduino-serial-port-communication/

[21] https://es.wikipedia.org/wiki/ACK

[22] https://es.wikipedia.org/wiki/NACK

[23] http://control-pid.wikispaces.com/

[24] http://www.fing.edu.uy/iq/cursos/dcp/teorico/16_AJUSTE_DE_CONTROLADORES.pdf

ANEXO

```
/*CÓDIGO PRINCIPAL PARA CONTROL LQR Y CONTROL PI*/

#include <I2C.h>  // Esta librería permite  comunicar con dispositivos I2C  TWI.
#include "Kalman.h" //Librería programada para el filtro de Kalman.

// Si esta línea no se comenta queda restringido a ±90º es el eje Y
#define RESTRICT_PITCH // Si esta línea se comenta, el giro alrededor del eje X se restringe a ±90º,
                       //mientras que el giro alrededor de Y se restringe a ±180º

Kalman kalmanX;
Kalman kalmanY; //Aquí se están creando los objetos kalmanX y KalmanY de tipo Kalman.h


/* IMU Data */

double accX, accY, accZ; //componentes de la aceleración
double gyroX, gyroY, gyroZ; //componentes de giro
int16_t tempRaw;  //Aquí se almacenará la temperatura en crudo (no en ºC)

double gyroXangle, gyroYangle; // Ángulo calculado usando el giro
double compAngleX, compAngleY; //Ángulo calculado usando el filtro complementario
double kalAngleX, kalAngleY; // Ángulo calculado usando el filtro de Kalman

uint32_t timer;
uint8_t IMUData[14]; //Creación del vector IMUData reservando 14 índices (buffer de comunicación)
```

```
/* PARA LOS ENCODERS, VELOCIDAD Y ACELERACIÓN */

//Motor Izquierdo
#define encoder0PinA 18
#define encoder0PinB 19
volatile float encoder0Pos = 0;
unsigned long time1;
volatile float ivr;
float pos0;
float vel;
float acel;
float vel0;

//Motor Derecho
#define encoder1PinA 2
#define encoder1PinB 3
volatile float encoder1Pos = 0;
unsigned long time2;
float pos0d;
//////////////////////////

/* PARA EL MOTOR */
#define Pinpot1 A2
#define Pinpot2 A1
#define Pinpot3 A0
#define PinPWMD 7
#define PinPWMI 9
#define PinIn1I 8
#define PinIn2I 10
#define PinIn1D 5
#define PinIn2D 6

float PWMD;
float PWMI;

//////////////////////////
```

```
/* PARA CALCULAR EL CONTROL PID*/
float u;
float phi_r,dphi_r,dtheta_r;
float phi, dphi;
float Ahora;
float Cambiot, Ultimot;
float Kp,Ki;
float r,e;
float Ahora1, Cambiot1, Ultimot1;
float ITerm;
float u_m;

/* PARA MANIOBRAS*/
float u_mi, u_md;
float tiempoDir;
float Estado;

/* PARA EL FILTRO DE LA VELOCIDAD*/
float wf, wf_1, wf_2, wf0;
float vel_1, vel_2, acelf;
float veli, veld;

/* PARA LAS VAR DE LA COMUNICACIÓN BLUETOOTH*/
float K1, K2, K3, c, K4; //variables para las constantes del LQR
int Dir; //variable para la dirección (de 1 a 9)
float xg, xt; //velocidad de giro y de traslación
int cuenta;

float K1_r, K2_r, K3_r, c_r;    //variables para las constantes en proceso de recepción del LQR
int Dir_r;
float xg_r, xt_r;
float Kp_r, Ki_r;    //para la recepcion constantes control motor

float tiempo0;    //Para el cálculo de tiempo de bucle de programa.
float bucle;

//////////////////////////
```

```
/* INICIALIZACIÓN DE LAS VARIABLES*/
void setup() {
 Serial.begin(115200);

 /*Configuración de los Timers 1 y 5*/

 noInterrupts();          //desactiva todas las interrupciones para inicializar la interrupción temporal
 TCCR1A = 0;              //se inicializan los registros de control del Timer 1
 TCCR1B = 0;
 TCNT1  = 0;

 OCR1A = 9999;            // valor CTC que se guarda en el registro OCR1A
 TCCR1B |= (1 << WGM12);  //configuración del modo output compare
 TCCR1B |= (1 << CS10);   //configuración del preescalador de 64
 TCCR1B |= (1 << CS11);   //configuración del preescalador de 64
 TIMSK1 |= (1 << OCIE1A); // activa la interrupción por comparación temporal

 TCCR5A = 0;              //se inicializan los registros de control del Timer 5
 TCCR5B = 0;
 TCNT5  = 0;

 OCR5A = 19999;           // valor CTC que se guarda en el registro OCR5A
 TCCR5B |= (1 << WGM52);  // modo output compare
 TCCR5B |= (1 << CS51);   // preescalador de 8
 TIMSK5 |= (1 << OCIE5A); // activa la interrupción por comparación temporal

 interrupts();            // activa todas las interrupciones

 I2c.begin();
      I2c.setSpeed(1); //Poner frecuencia a 400kHz
       I2c.pullup(1); //activamos la resistencia de pull-up
      I2c.timeOut(1000);  //habilita el recibir los errores por el envío fallido y no se queda colgado ya
                          //que puede saber el tipo de error y seguir funcionando

      IMUData[0] = 7;    // >>Configuración tasa muestreo. Registro 0x19hex=25dec
      IMUData[1] = 0x00; //  >>Sincronización y Filtrado.   Registro 0x1Ahex=26dec
      IMUData[2] = 0x00; //  >>Configuración Giróscopo. Registro 0x1Bhex=27dec
```

```
        IMUData[3] = 0x00;  //>>Configuración Acelerómetros. Registro 0x1Chex=28dec


        while (IMUWrite(0x19, IMUData, 4, false)); // >>Mete los anteriores 4 valores en cada posición
                                                   //del registro
        while (IMUWrite(0x6B, 0x01, true));     // >>Deshabilita modo standby. Usa como señal de reloj,
                                                 //el giróscopo del eje X
                                                //Se guarda en el Registro 0x69hex=107dec


        while (IMURead(0x75, IMUData, 1));       //Lee los registros 0x75hex=117dec;
                                                 //y los mete en el vector IMUData.
        if (IMUData[0] != 0x68) {  // Read "WHO_AM_I" register, este registro "Quién Soy?" devuelve
                                   //el valor 0x68 para decir cuál es la dirección del dispositivo
          Serial.print(F("Error reading sensor"));
          while (1);
        }


        delay(100); // Espera a que se estabilice el sensor


        /* Seleccionar kalman y el ángulo de inicio de giro */
/*Lectura de los Registros de los acelerómetros:
0x3Bhex=59dec y 0x3Chex=60dec;
0x3Dhex=61dec y 0x3Ehex=62dec;
0x3Fhex=63dec y 0x40hex=64dec;
 y los guarda en el vector IMUData*/
        while (IMURead(0x3B, IMUData, 6));
        accX = (IMUData[0] << 8) | IMUData[1];   //(IMUData[0] << 8)=esto es un desplazamiento de 8
                                                 //bits a la izquierda de forma que el IMUData[0] queda
                                                //en las posiciones más significativas y a la derecha quedan
                                                 //ceros
        accY = (IMUData[2] << 8) | IMUData[3];  // Con la operación | se suman bit a bit a posiciones
                                                 //menos significativas con el valor IMUData[1].
        accZ = (IMUData[4] << 8) | IMUData[5];   // de 16 bit a partir de dos números de 8 bit


      #ifdef RESTRICT_PITCH
        double roll  = atan2(accY, accZ) * RAD_TO_DEG;
        double pitch = atan(-accX / sqrt(accY * accY + accZ * accZ)) * RAD_TO_DEG;
      #else
        double roll  = atan(accY / sqrt(accX * accX + accZ * accZ)) * RAD_TO_DEG;
        double pitch = atan2(-accX, accZ) * RAD_TO_DEG;
      #endif
```

```
    kalmanX.setAngle(roll); // Seleccionar ángulo de inicio
    kalmanY.setAngle(pitch);
    gyroXangle = roll;
    gyroYangle = pitch;
    compAngleX = roll;
    compAngleY = pitch;

    timer = micros();

  /////// CÓDIGO PARA LA POSICION, VELOCIDAD Y ACELERACIÓN //////

  //Motor izquierdo
  pinMode(encoder0PinA, INPUT);
  digitalWrite(encoder0PinA, HIGH);      // activar resistencias pullup
  pinMode(encoder0PinB, INPUT);
  digitalWrite(encoder0PinB, HIGH);      // activar resistencias pullup

//Recordar que son encoders de cuadratura, hay dos sensores por cada motor colocados a 90º uno del
//otro
  attachInterrupt(5, doEncoderA0, CHANGE); // Encoder A en la interrupción 5 (pin 18)
  attachInterrupt(4, doEncoderB0, CHANGE);  // Encoder B en la interrupción 4 (pin 19)

  //Motor derecho
  pinMode(encoder1PinA, INPUT);
  digitalWrite(encoder1PinA, HIGH);      // activar resistencias pullup
  pinMode(encoder1PinB, INPUT);
  digitalWrite(encoder1PinB, HIGH);      // activar resistencias pullup

//Recordar que son encoders de cuadratura, hay dos sensores por cada motor colocados a 90º uno del
//otro
  attachInterrupt(0, doEncoderA1, CHANGE); // Encoder A en la interrupción 0 (pin 2)
  attachInterrupt(1, doEncoderB1, CHANGE);  // Encoder B en la interrupción 1 (pin 3)

  time1=millis();   //Se  Inicializa tiempo y posición para el cálculo de velocidad
  time2=millis();   // Inicializamos tiempo y posición para el cálculo de velocidad
```

```
////////// MOTOR ////////////
//Para subir la frecuencia del PWM pin 7 del Mega 2560 de 489Hz a 31333Hz
TCCR4B = TCCR4B & 0b000 | 0x01; //uso del timer 4

//Para subir la frecuencia del PWM pin 9 del Mega 2560 de 489Hz a 31333Hz
TCCR2B = TCCR2B & 0b000 | 0x01; //uso del timer 2

pinMode(PinPWMD, OUTPUT);
pinMode(PinPWMI, OUTPUT);
pinMode(PinIn1I, OUTPUT);
pinMode(PinIn2I, OUTPUT);
pinMode(PinIn1D, OUTPUT);
pinMode(PinIn2D, OUTPUT);


///CONTROLADOR////
phi_r=0; dphi_r=0; dtheta_r=0;
u=0;
Kp=0.3;
Ki=0.06;
K1=-332.8746; K2=-52.7835; K3=-7; c=2.0; Dir=5.0; xg=1.0; xt=1.0; K4=-0.3; ivr=0;

//////////////////////////////

///FILTRO VELOCIDAD////
wf_2=1.0;
wf_1=1.0;
wf=1.0;
vel_2=1.0;
vel_1=1.0;
wf0=1.0;

}

//////////////////////////////////
```

```
void loop() {

 RecepcionBT();

     /////CÓDIGO PARA LA IMU////

          /* Actualización de los valores */
          while (IMURead(0x3B, IMUData, 14));
          accX = ((IMUData[0] << 8) | IMUData[1]) - 883.8;
          accY = ((IMUData[2] << 8) | IMUData[3]) - 114.5;
          accZ = ((IMUData[4] << 8) | IMUData[5]) + 134.75;
          tempRaw = (IMUData[6] << 8) | IMUData[7];
          gyroX = (IMUData[8] << 8) | IMUData[9];
          gyroY = (IMUData[10] << 8) | IMUData[11];
          gyroZ = (IMUData[12] << 8) | IMUData[13];

          double dt = (double)(micros() - timer) / 1000000; // Calcular el delta t
          timer = micros();


         #ifdef RESTRICT_PITCH
        double roll  = atan2(accY, accZ) * RAD_TO_DEG;
        double pitch = atan(-accX / sqrt(accY * accY + accZ * accZ)) * RAD_TO_DEG;
         #else
          double roll  = atan(accY / sqrt(accX * accX + accZ * accZ)) * RAD_TO_DEG;
          double pitch = atan2(-accX, accZ) * RAD_TO_DEG;
         #endif

          double gyroXrate = gyroX / 131.0; // Convierte a deg/s
          double gyroYrate = gyroY / 131.0; // Convierte a deg/s

        #ifdef RESTRICT_PITCH

// Solución al problema de transición cuando el ángulo del acelerometro salta entre -180 y 180 grados
          if ((roll < -90 && kalAngleX > 90) || (roll > 90 && kalAngleX < -90)) {
            kalmanX.setAngle(roll);
            compAngleX = roll;
            kalAngleX = roll;
```

```
    gyroXangle = roll;
  } else
    kalAngleX = kalmanX.getAngle(roll, gyroXrate, dt); //Cálculo del ángulo con el filtro
                                                       //Kalman

  if (abs(kalAngleX) > 90)
    gyroYrate = -gyroYrate; // Invierte la velocidad angular, de forma que cumple la lectura
                            //restringida del acelerómetro
  kalAngleY = kalmanY.getAngle(pitch, gyroYrate, dt);
#else

// Solución al problema de transición cuando el ángulo del acelerómetro salta entre -180 y 180 grados
  if ((pitch < -90 && kalAngleY > 90) || (pitch > 90 && kalAngleY < -90)) {
    kalmanY.setAngle(pitch);
    compAngleY = pitch;
    kalAngleY = pitch;
    gyroYangle = pitch;
  } else
    kalAngleY = kalmanY.getAngle(pitch, gyroYrate, dt); //Cálculo del ángulo con el filtro de
                                                        //Kalman

  if (abs(kalAngleY) > 90)
    gyroXrate = -gyroXrate; // Invierte la velocidad angular, de forma que cumple la lectura
                            //restringida del acelerómetro
  kalAngleX = kalmanX.getAngle(roll, gyroXrate, dt);  // Cálculo del ángulo con el filtro de
                                                      //Kalman
#endif


  gyroXangle += gyroXrate * dt; // Calcula el ángulo obtenido del giróscopo sin filtro
  gyroYangle += gyroYrate * dt;

//Cálculo del ángulo usando filtro complementario
  compAngleX = 0.93 * (compAngleX + gyroXrate * dt) + 0.07 * roll;

  compAngleY = 0.93 * (compAngleY + gyroYrate * dt) + 0.07 * pitch;

  // Resetear el ángulo del giróscopo si ha sufrido gran deriva
  if (gyroXangle < -180 || gyroXangle > 180)
    gyroXangle = kalAngleX;
```

```
            if (gyroYangle < -180 || gyroYangle > 180)
              gyroYangle = kalAngleY;



 ///////// CÓDIGO PARA POSICIÓN, VELOCIDAD Y ACELERACIÓN ////////

if ((millis()-time1)>=10){
   veli=((encoder0Pos-pos0)*1000.0/(millis()-time1))*(PI/180.0);
   pos0=encoder0Pos;
   veld=((encoder1Pos-pos0d)*1000.0/(millis()-time1))*(PI/180.0);
   pos0d=encoder1Pos;

   vel=(veli+veld)/2.0;

   wf=1.6544*wf_1 - 0.7059*wf_2 + 0.0129*vel + 0.0257*vel_1 + 0.0129*vel_2;

     wf_2=wf_1;
     wf_1=wf;
     vel_2=vel_1;
     vel_1=vel;

   acel=(vel-vel0)*1000.0/(millis()-time1);
   acelf=(wf-wf0)*1000.0/(millis()-time1);

   wf0=wf;
   vel0=vel;
   time1=millis();
 }
////////////////////////////////////////////////////////////////

///////// CÓDIGO PARA CONTROLADOR ////////

 dphi=gyroYrate*(PI/180);
 Mando_Motor();

 ////////////////////////////////////////////////////////////
```

```
 /*SECCIÓN PARA EL ENVÍO*/

 bucle=millis()-tiempo0;
 tiempo0=millis();

 EnvioBT();

}

//////// CÓDIGO PARA CONTROLADOR ////////

ISR(TIMER1_COMPA_vect)        // rutina de interrupción con Output Compare
{
        phi=kalAngleY*(PI/180);
        ivr=ivr+dtheta_r-wf;
        u=K1*(phi_r-phi) + K2*(dphi_r-dphi) + K3*(dtheta_r-wf) + K4*ivr;
        r=-u*c;
}

ISR(TIMER5_COMPA_vect)        // rutina de interrupción con Output Compare
{
    e=r-acelf;
    ITerm += Ki*e;
    if((abs(ITerm))>255.0){
      if (ITerm>0) ITerm=255.0;
      else ITerm=-255.0;}

    u_m=Kp*e+ITerm;
}

////////////////////////////////////////////////////////////
```

```
///// A CONTINUACIÓN VIENEN LAS FUNCIONES DE LAS INTERRUPCIONES PARA LOS ENCODERS ///////

//INTERRUPCIONES ENCODERS MOTOR IZQUIERDO
void doEncoderA0(){
 if (digitalRead(encoder0PinA) == HIGH) {   // se busca un cambio de bajo-a-alto en el canal A
  if (digitalRead(encoder0PinB) == LOW) {   //se  comprueba el canal B para saber en qué sentido gira
    encoder0Pos++;}       // Horario
  else {encoder0Pos--;}   // Antihorario
 }

 else{  // en otro caso sería un cambio de alto-a-bajo en canal A
  if (digitalRead(encoder0PinB) == HIGH) {  // se comprueba el canal B para saber en qué sentido gira
    encoder0Pos++;}         // Horario
  else {encoder0Pos--;}     // Antihorario
 }
}

void doEncoderB0(){
 if (digitalRead(encoder0PinB) == HIGH) {   //se busca un cambio de bajo-a-alto en el canal B
  if (digitalRead(encoder0PinA) == HIGH) {  //se comprueba el canal A para saber en qué sentido gira
    encoder0Pos++;}        // Horario
  else {encoder0Pos--;}     // Antihorario
 }

 else {  // en otro caso sería un cambio de alto-a-bajo en canal B
  if (digitalRead(encoder0PinA) == LOW) {   //se comprueba el canal A para saber en qué sentido gira
    encoder0Pos++;}         // Horario
  else {encoder0Pos--;}     // Antihorario
 }
}

 //INTERRUPCIONES ENCODERS MOTOR DERECHO
void doEncoderA1(){
 if (digitalRead(encoder1PinA) == HIGH) {   //se busca un cambio de bajo-a-alto en el canal A
  if (digitalRead(encoder1PinB) == LOW) {   //se comprueba el canal B para saber en qué sentido gira
    encoder1Pos--;}       // Horario
```

```
  else {encoder1Pos++;}   // Antihorario
 }

 else{  // en otro caso sería un cambio de alto-a-bajo en canal A
  if (digitalRead(encoder1PinB) == HIGH) {  //se comprueba el canal B para saber en qué sentido gira
    encoder1Pos--;}         // Horario
  else {encoder1Pos++;}      // Antihorario
 }
}

void doEncoderB1(){
 if (digitalRead(encoder1PinB) == HIGH) {   //se busca un cambio de bajo-a-alto en el canal B
  if (digitalRead(encoder1PinA) == HIGH) {  //se comprueba el canal A para saber en qué sentido gira
    encoder1Pos--;}        // Horario
  else {encoder1Pos++;}     // Antihorario
 }

 else {  // en otro caso sería un cambio de alto-a-bajo en canal B
  if (digitalRead(encoder1PinA) == LOW) {   //se comprueba el canal A para saber en qué sentido gira
    encoder1Pos--;}         // Horario
  else {encoder1Pos++;}      // Antihorario
 }
}

//////////////////////////////////////////////////


/*CÓDIGO DE LAS FUNCIONES DE COMUNICACIÓN CON LA IMU*/

const uint8_t IMUAddress = 0x68;

uint8_t IMUWrite(uint8_t registerAddress, uint8_t data, bool sendStop) {
 return IMUWrite(registerAddress, &data, 1, sendStop); // devuelve 0 si tiene éxito
}

uint8_t IMUWrite(uint8_t registerAddress, uint8_t *data, uint8_t length, bool sendStop) {
//registerAddress es el registro de la IMU
 uint8_t rcode=I2c.write(IMUAddress,registerAddress,*data);   //si devuelve 0 es que ha tenido éxito
```

```
  if (rcode) {
    Serial.print(F("IMUWrite failed: "));
    Serial.println(rcode);
  }
  if (sendStop=true){
    I2c.end();
    Serial.println(F("libera bus"));
  }
  return rcode;
}

uint8_t IMURead(uint8_t registerAddress, uint8_t *data, uint8_t nbytes) {
  uint32_t timeOutTimer;
  I2c.read(IMUAddress,registerAddress,nbytes);
  for (uint8_t i = 0; i < nbytes; i++) {
    if (I2c.available())
      data[i] = I2c.receive();
    else {
      timeOutTimer = micros();
      while (((micros() - timeOutTimer) < 1000) && !I2c.available());
      if (I2c.available())
        data[i] = I2c.receive();
      else {
        Serial.println(F("IMURead timeout"));
        return 5;
      }
    }
  }
  return 0; //Si tiene éxito
}

////////////////////////////////////////////////


/*CÓDIGO DE LA FUNCIÓN Mando_Motor*/

void Mando_Motor(){
 switch(Dir){
```

```
case 1:
if ((millis()-tiempoDir)<1000){
  u_md=-u_m+10.0*xg;
  u_mi=-u_m-10.0*xg;
  Estado=10;}
if((millis()-tiempoDir)>=1000){
  dtheta_r=0.76*PI*xt;
  u_mi=-u_m;
  u_md=u_mi;
  Estado=101;}
break;

case 2:
  dtheta_r=0.76*PI*xt;
  u_mi=-u_m;
  u_md=u_mi;
  Estado=20;
break;

case 3:
if ((millis()-tiempoDir)<1000){
  u_mi=-u_m+10.0*xg;
  u_md=-u_m-10.0*xg;
  Estado=30;}
if((millis()-tiempoDir)>=1000){
  dtheta_r=0.76*PI*xt;
  u_mi=-u_m;
  u_md=u_mi;
  Estado=301;}
break;

case 4:
  u_md=-u_m+10.0*xg;
  u_mi=-u_m-10.0*xg;
  Estado=40;
break;
```

```
case 5:
  dtheta_r=0;
  u_mi=-u_m;
  u_md=u_mi;
  Estado=50;
break;

case 6:
  u_mi=-u_m+10.0*xg;
  u_md=-u_m-10.0*xg;
  Estado=60;
break;

case 7:
if ((millis()-tiempoDir)<1000){
  u_mi=-u_m+10.0*xg;
  u_md=-u_m-10.0*xg;
  Estado=70;}
if((millis()-tiempoDir)>=1000){
  dtheta_r=-0.76*PI*xt;
  u_mi=-u_m;
  u_md=u_mi;
  Estado=701;}
break;

case 8:
  dtheta_r=-0.76*PI*xt;
  u_mi=-u_m;
  u_md=u_mi;
  Estado=80;
break;

case 9:
if ((millis()-tiempoDir)<1000){
  u_md=-u_m+10.0*xg;
  u_mi=-u_m-10.0*xg;
  Estado=90;}
if((millis()-tiempoDir)>=1000){
```

```
      dtheta_r=-0.76*PI*xt;
      u_mi=-u_m;
      u_md=u_mi;
      Estado=901;}
    break;

   default:
    dtheta_r=0;
      u_mi=-u_m;
      u_md=u_mi;
      Estado=112;
}

 if (u_mi>=0){
   digitalWrite(PinIn1I, HIGH);
   digitalWrite(PinIn2I, LOW);
   PWMI=14+abs(u_mi);}
else {
   digitalWrite(PinIn1I, LOW);
   digitalWrite(PinIn2I, HIGH);
   PWMI=14+abs(u_mi);}


 if (u_md>=0){
   digitalWrite(PinIn1D, HIGH);
   digitalWrite(PinIn2D, LOW);
   PWMD=14+abs(u_md);}
 else {
   digitalWrite(PinIn1D, LOW);
   digitalWrite(PinIn2D, HIGH);
   PWMD=14+abs(u_md);}

 if (PWMI>255){
   PWMI=255;}
 if (PWMD>255){
   PWMD=255;}
```

```
 if (abs(kalAngleY)>30){
   PWMI=0;
   PWMD=0;
   ivr=0;}

  analogWrite(PinPWMI, PWMI);
  analogWrite(PinPWMD, PWMD);

}

//////////////////////////////////////////////

/*CÓDIGO PARA LA TRANSFERENCIA POR BLUETOOTH*/

void RecepcionBT(){
  while (Serial.available()>0)
   {
      xg_r = Serial.parseFloat();  //Esta es la velocidad de giro, la posición del slider
      xt_r = Serial.parseFloat();
      Kp_r = Serial.parseFloat();
      Ki_r = Serial.parseFloat();
      K1_r = Serial.parseFloat();
      K2_r = Serial.parseFloat();
      K3_r = Serial.parseFloat();
      c_r = Serial.parseFloat();
      Dir_r = Serial.parseInt();

//Cuando lea el carácter fin de línea ('\n') quiere decir que ha finalizado el envío de los tres valores
      if (Serial.read() == '\n')
      {
         //Se actualizan las variables reales si procede
            if (K1_r != 0)
              {Kp=Kp_r; Ki=Ki_r; K1=K1_r; K2=K2_r; K3=K3_r; c=c_r;}

            if (Dir_r != 0)
              {Dir=Dir_r;
               if(Dir==1||Dir==3||Dir==7||Dir==9)
```

```
                {tiempoDir=millis();}
              }

            if (xg_r != 0)
              {xg=xg_r;}

            if (xt_r != 0)
              {xt=xt_r;}

        }
    }

}

void EnvioBT(){

  Serial.print("^");
  Serial.print(Kp); Serial.print(",");
  Serial.print(Ki);  Serial.print(",");
  Serial.print(K1); Serial.print(",");
  Serial.print(K2); Serial.print(",");
  Serial.print(K3); Serial.print(",");
  Serial.print(c);Serial.print(",");
  Serial.print(bucle);Serial.print(","); //Tiempo que tarda en dar una vuelta el programa
  Serial.print(Dir);Serial.print(",");
  Serial.print(Estado);Serial.print(",");
  Serial.print(xg);Serial.print(",");
  Serial.print(xt);Serial.print(",");

    Serial.print(veli); Serial.print("\t");
    Serial.print(veld); Serial.print("\t");
    Serial.print(u_m); Serial.print("\t");
    Serial.print(PWMI); Serial.print("\t");
    Serial.print(PWMD);Serial.print("\t");
    Serial.print(millis());Serial.print("\t");
    Serial.print("\r\n");
    delay(1);
```

```
}

//////////////////////////////////////////////////

/*LIBRERÍA DEL FILTRO DE KALMAN*/

#ifndef _Kalman_h
#define _Kalman_h

class Kalman {
public:
  Kalman() {
    Q_angle = 0.001;
    Q_bias = 0.003;
    R_measure = 0.03;

    angle = 0;
    bias = 0;

    P[0][0] = 0;
    P[0][1] = 0;
    P[1][0] = 0;
    P[1][1] = 0;
  };
  double getAngle(double newAngle, double newRate, double dt) {

    /* Step 1 */
    rate = newRate - bias;
    angle += dt * rate;

    /* Step 2 */
    P[0][0] += dt * (dt*P[1][1] - P[0][1] - P[1][0] + Q_angle);
    P[0][1] -= dt * P[1][1];
    P[1][0] -= dt * P[1][1];
    P[1][1] += Q_bias * dt;

    /* Step 4 */
```

```
        S = P[0][0] + R_measure;
        /* Step 5 */
        K[0] = P[0][0] / S;
        K[1] = P[1][0] / S;

        /* Step 3 */
        y = newAngle - angle;
        /* Step 6 */
        angle += K[0] * y;
        bias += K[1] * y;

        /* Step 7 */
        P[0][0] -= K[0] * P[0][0];
        P[0][1] -= K[0] * P[0][1];
        P[1][0] -= K[1] * P[0][0];
        P[1][1] -= K[1] * P[0][1];

        return angle;
    };
    void setAngle(double newAngle) { angle = newAngle; };
    double getRate() { return rate; };

    void setQangle(double newQ_angle) { Q_angle = newQ_angle; };
    void setQbias(double newQ_bias) { Q_bias = newQ_bias; };
    void setRmeasure(double newR_measure) { R_measure = newR_measure; };

    double getQangle() { return Q_angle; };
    double getQbias() { return Q_bias; };
    double getRmeasure() { return R_measure; };

private:
    double Q_angle;
    double Q_bias;
    double R_measure;

    double angle;
    double bias;
```

```
    double rate;

    double P[2][2];
    double K[2];
    double y;
    double S;
};

#endif
```