Universidad del País Vasco
Euskal Herriko Unibertsitatea

Facultad de Informática

Grado de Ingeniería Informática

▪Proyecto Fin de Grado▪

Ingeniería de Computadores

Estudio de la cinemática de un móvil de dos ruedas y su control para mantener el equilibrio disponiendo de encoder

AnartzRecalde

2015

informatika fakultatea
facultad de informática

RESÚMEN

Como el nombre del proyecto indica, este proyecto presentamos el diseño e implementación de un móvil de dos ruedas, en concreto un robot balancín. En él se estudiarán las teorías necesarias para que un móvil de dos ruedas se mantenga en equilibrio y poner dichas teorías en práctica. Estas teorías son las mismas que se usan para el control del péndulo invertido en el cual un móvil intenta mantener un péndulo en equilibrio en todo momento.

En este proyecto se diferencian 3 fases:

La primera trata sobre el control del equilibrio del robot, en el cual se pretende lograr que el móvil de dos ruedas se mantenga en equilibrio en todo momento incluso aplicando fuerzas externas. Para ello, se tendrán que interpretar los datos leídos de un sensor del cual conseguiremos el ángulo que el robot ha girado respecto a la posición de equilibrio y dar potencia a las ruedas para conseguir que el robot se equilibre. Tanto en el proceso de interpretar el ángulo, como en el de dar potencia a las ruedas se aplican diferentes métodos de control del robot para que el móvil sea capaz de mantenerse en equilibrio.

En la segunda fase, partiendo de que el robot se encuentra en equilibrio constará de utilizar los *encoders* de los motores para mejorar el control del robot. Además de poder conseguir la distancia que el robot ha recorrido, también se pretende que el robot, una vez desplazado por alguna fuerza exterior, vuelva a la posición en la que se encontraba antes de ejercer dicha fuerza.

Por último, también se pretende controlar el robot vía Bluetooth, pudiendo controlarlo mediante algún dispositivo de forma remota.

Este proyecto se ha desarrollado junto a otro compañero llamado Markel Picado. Junto a él se ha trabajado en la primera fase, en la cual conseguimos que el robot se mantenga en equilibrio y a partir de ahí nos dividimos el trabajo, siendo la segunda fase para que yo la desarrollase y la tercera fase para que Markel la desarrollase. Por tanto, este proyecto trata sobre el control necesario para mantener el robot en equilibrio y el uso de *encoders* para añadir algunas funcionalidades extras.

En cuanto a la documentación, he de comentar que en el apartado de sistema de control la parte realizada en común se ha redactado entre Markel Picado y yo, y la parte de problemas encontrados, comunes a ambos, también.

ÍNDICE

ÍNDICE FIGURAS

1 INTRODUCCION

1.1 VISIÓN GENERAL

Las tecnologías de movilidad robótica y los autómatas a los que estos se aplican se han vuelto muy populares en los últimos años, tanto en los sectores comerciales como en los gubernamentales. Se han desarrollado técnicas para mejorar la movilidad de los autómatas en entornos dinámicos, es decir, en movimiento e incluso aunque se apliquen fuerzas externas. Gracias a estas técnicas, se ha conseguido construir autómatas capaces de mantener el equilibrio en entornos muy inestables. Unos claros ejemplos de estos podrían ser los *Segway*.

El *Segway* es un vehículo de transporte ligero giroscópico eléctrico de dos ruedas, con autobalance controlado por ordenador o microcontrolador. Fue presentado en 2001 por Dean Kamen y desde su presentación a la sociedad, hasta hoy en día han ido ganando seguidores poco a poco. Tanto es así, que hoy en día es muy normal ver uno de estos aparatos en algún centro comercial donde patrullan guardias montados en ellos, o incluso en algunos lugares de España, como por ejemplo Alicante, Madrid o Valencia entre otros, donde la seguridad pública ha incorporada algunos de estos vehículos.

La idea del robot balancín surge del *Segway*. Dado que construir un *Segway* puede ser bastante complicado tanto a la hora de comprar los materiales para construirlo como a la hora de hacer las pruebas de funcionamiento o de ajuste de valores, puesto que un fallo podría ocasionar alguna lesión en la persona que lo prueba. Para ello, surgió una alternativa interesante, la de crear un *Segway* en miniatura o dicho de otro modo, un robot balancín.

1.2 OBJETIVOS

El objetivo principal de este proyecto consiste en conseguir un robot balancín capaz de mantenerse estable por sí mismo en cualquier situación. Además, también se quiere conseguir la velocidad de desplazamiento o la distancia recorrida mediante los *encoders* o conseguir alguna funcionalidad extra mediante éstos.

Para cumplir estos objetivos principales debemos ir avanzando poco a poco y cumplir antes otros objetivos:

- Profundizar en la programación de microcontroladores.
- Conocer las Unidades de Medición Inercial (IMU), que sensores llevan, su funcionamiento y como se comunican con el microcontrolador e interpretar los valores recibidos.

- Comprender como funcionan los controladores del motor y comunicarlos con el microcontrolador para hacer mover las ruedas.
- Conocer los fundamentos físicos necesarios para el mantenimiento en equilibrio del robot balancín.
- Hacer que el robot balancín se mantenga en equilibrio.
- Controlar la distancia y la velocidad con la que el robot se ha desplazado.
- Lograr que el robot vuelva al sitio donde se encontraba antes de que le empujasen.

1.3 PLANIFICACIÓN

Este proyecto puede dividirse en dos grandes partes, la primera trata sobre cómo conseguir que el móvil se mantenga en equilibrio en todo momento, la cual se ha realizado en compañía de Markel Picado, y una segunda que es la de añadir funcionalidades extra mediante el uso de *encoders*.

Dado que la primera parte del proyecto se ha realizado junto a mi compañero Markel Picado y la segunda parte ha sido realizada individualmente hemos tenido que separar el proyecto en dos fases. Esto hay que tenerlo en cuenta, puesto que mi residencia durante el curso académico se encuentra en Donostia/San Sebastián y comparto piso con el compañero, Markel Picado, pero entre junio y agosto la residencia de cada uno será su casa particular.

Para completar con éxito el proyecto, este se divide en tareas que deben ser completadas para su correcto desarrollo. En la siguiente EDT (Estructura de Desglose de Trabajo) se muestran gráficamente:

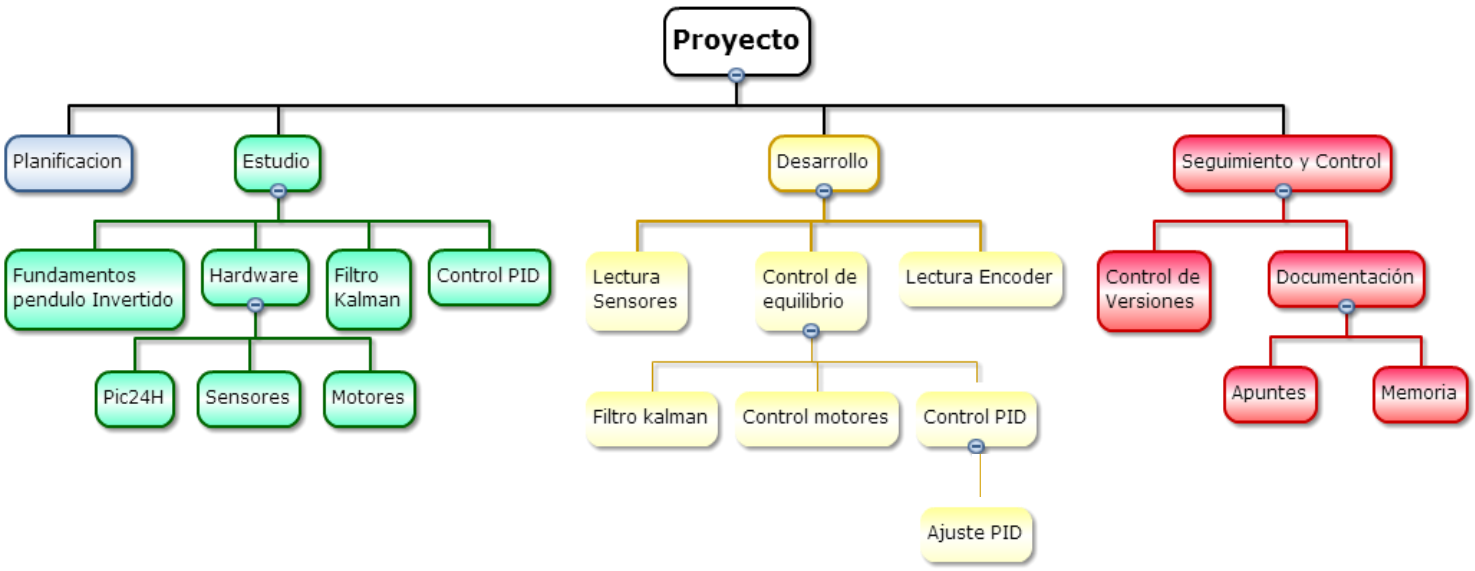


Figura 1: Estructura de desglose de trabajo (EDT)

La descripción de las tareas principales es la siguiente:

- Planificación: Etapa de planteamiento del proyecto y puesta a punto de todo lo necesario para empezar con el proyecto.
- Estudio: Tarea dedicada a buscar información y entender las diferentes tecnologías, fundamentos y teorías necesarias para el proyecto.
- Desarrollo: Diseño e implementación del código del proyecto.
- Seguimiento y Control: Parte destinada a revisar el desarrollo del proyecto, gestionar las versiones del código, y tomar apuntes para la documentación.

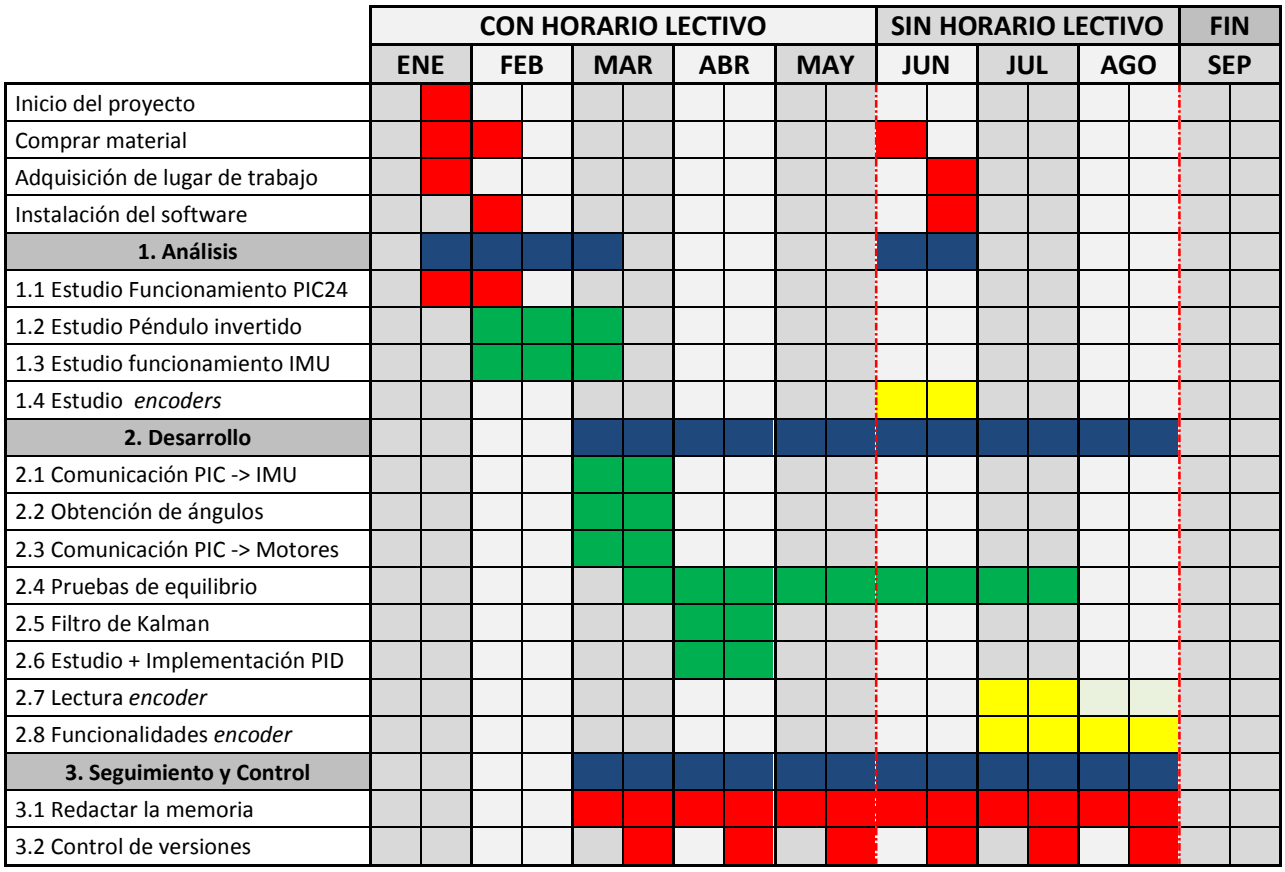
1.3 HITOS

Para poder planificar y organizar mejor los tiempos del proyecto, se creado unos puntos de control o hitos, que se alcanzan al completar algunas de las tareas designadas. Los hitos en cuestión son los siguientes:

- H1: Fin de la planificación
- H2: Fin del estudio de sensores de la *IMU*
- H3: Lectura correcta de sensores de la *IMU*
- H4: Calculo del ángulo y comprobación del correcto cálculo de este.
- H5: Fin del estudio del control PID.
- H6: Fin del diseño e implementación del control PID.
- H7: Fin de la implementación de Motores.
- H8: Fin de la implementación del filtro de Kalman.
- H9: Pruebas de Equilibrio.
- H10: Fin estudio de *encoders* del motor.
- H11: Fin diseño e implementación de *encoders*.
- H12: Fin de la documentación

1.4 DIAGRAMA DE GANTT

Teniendo todo lo anterior en cuenta se procede a planificar un diagrama de Gantt. En él se muestran todas las fases del proyecto que hemos considerado y las fechas en las que se ha supuesto que el proyecto durase. Gracias a ello podemos ver un desglose de las tareas que hemos de completar para que el proyecto termine satisfactoriamente de manera sencilla e intuitiva.



	Primera Fase
	Segunda Fase
	Común ambas Fases

Aquí podemos observar como el proyecto ha evolucionado en el tiempo. Al principio, se comienza con el análisis y estudio de los componentes a utilizar durante la primera fase del mismo. Una vez transcurrido esta fase, pasamos a la de desarrollo, en la que intentamos poner el robot en equilibrio. Se puede observar que a la hora de poner el robot en equilibrio se le dedicó demasiado tiempo, dado que tuvimos varios problemas como se explicará más adelante. Y por último queda el estudio de los *encoder* y sus funcionalidades extras mediante el uso de los *encoder*. Mientras tanto se lleva a cabo la redacción de la memoria.

1.5 ADQUISICIÓN DE LUGAR DE TRABAJO

Dado que la primera parte del proyecto se ha realizado junto a mi compañero Markel Picado y la segunda parte ha sido realizada individualmente hemos tenido que separar el proyecto en dos fases. Esto hay que tenerlo muy en cuenta, puesto que mi residencia durante el curso académico se encontró en Donostia/San Sebastián y compartiendo piso con el compañero del

proyecto, Markel Picado, pero entre junio y agosto la residencia de cada uno ha sido su casa particular.

Esto no solo implica que a la hora de tener que trabajar juntos debamos encontrarnos cerca, sino que también a la hora de trabajar con el robot, éste deba estar en un lugar accesible fácilmente para ambos para poder trabajar cómodamente con él. Mientras estábamos en Donostia teníamos un laboratorio para poder trabajar los dos juntos sin problema alguno, pero llegado verano la cosa cambio. Por ello el profesor nos ayudó para conseguir un aula en la Escuelas Universitarias de Ingeniería Técnica Industrial (EUITI) situada en Bilbao. Esto supuso una gran ayuda para nosotros, puesto que disponíamos de un sitio en el que ambos podíamos trabajar con el robot cómodamente.

1.6 PLAN DE COMUNICACIÓN

A la hora de comunicarnos con el director entre enero y mayo fue directa, es decir, en el momento que necesitábamos algo pasábamos por su despacho puesto que vivíamos al lado de la facultad. Una vez terminado el periodo académico, a comienzos de junio, la situación cambio. Como se ha mencionado, nos tuvimos que volver a nuestras casas (cerca de Bilbao) y la ida y vuelta a San Sebastián suponía un largo trayecto. Por ello se acordó reunirnos una vez a la semana, concretamente los jueves, para poder llevar un seguimiento del proyecto. También se acordó de que en caso de que surgiese algún inconveniente grave que impidiese la continuación del trabajo se viajase a la facultad tras mandarle un correo para avisar al director cuanto antes para acordar una cita.

1.7 PLAN DE RIESGOS

En cuanto a la gestión de riesgos a lo que ha perdida de datos se refiere, se irán subiendo a una carpeta del google drive versiones, después de cada sesión de trabajo de lo avanzado en cuanto a memoria o código del robot balancín. Además, cada semana se efectuará una copia de seguridad del documento y código más avanzados en un disco duro que poseo para mayor seguridad.

2 ROBOT BALANCÍN

2.1 DEFINICIÓN

Un robot balancín es un robot que se sostiene sobre dos ruedas y es capaz de mantener el equilibrio dando fuerza a estas ruedas para contrarrestar las fuerzas que puedan afectar al desequilibrio del robot.

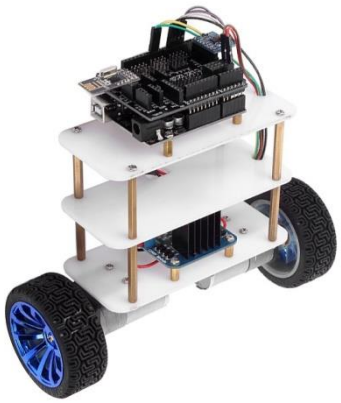


Figura 2: Robot Balancín

2.2 HISTORIA

Las tecnologías de movilidad robótica y los autómatas a los que estos se aplican se han vuelto muy populares en los últimos años, tanto en los sectores comerciales como en los gubernamentales. Se han desarrollado técnicas para mejorar la movilidad de los autómatas en entornos dinámicos, es decir, en movimiento e incluso aunque se apliquen fuerzas externas. Gracias a estas técnicas se ha conseguido construir autómatas que sean capaces de mantener el equilibrio en entornos muy inestables. Unos claros ejemplos de estos podrían ser los *Segway*.

El *Segway* es un vehículo de transporte ligero giroscópico eléctrico de dos ruedas, con autobalance, controlado por ordenador o microcontrolador. Este tiene unas características muy atractivas, puesto que es ecológico, no emite ruido apenas y su conducción es muy intuitiva y por tanto se aprende a usar rápidamente. Además alcanza velocidades de hasta 20 km/h y tiene una autonomía de alrededor de 40 km. En cuanto al mantenimiento, solo hace falta revisar la presión de los neumáticos y cargar las baterías diariamente en un enchufe.

Figura 3: *Segway*

El *Segway* fue presentado el 3 de diciembre del 2001 en Nueva York. Cuando se presentó se supuso que iba a revolucionar la industria del transporte. Se trataba de un medio de transporte ecológico (eléctrico), ágil y sencillo de usar, ideal para las ciudades. Tenía las bendiciones de gente como Steve Jobs (anterior director de Apple), Jeff Bezos (fundador de Amazon) o John Doerr, los cuales decían que podría ser el transporte del futuro. A pesar de dichas bendiciones, cuando al final salió al mercado fue un fracaso. Además de ser caro casi nadie lo quería. Tanto fue así, que la compañía tan solo vendió 6.000 unidades tras 21 meses, de las 50.000 unidades que habían esperado para el primer año.

Su alto precio, entre 4000 y 5500 dólares, y su aspecto, que decían que era para niños “pijos”, se consideran los principales factores de que fracasase la venta de estos. Para mejorar la imagen asociada al vehículo *Segway* Inc abrió concesionarios por todo Estados Unidos en los que se pueden examinar y probar los *Segway*.

Además de los inconvenientes del producto, la propia marca ha vivido una historia trágica. Creada en 2001, en 2010 fue vendida a una empresa propiedad del emprendedor inglés Jimi Heselden, que falleció a causa de un accidente cuando se desplazaba precisamente sobre un *Segway* y éste cayó por un barranco.

Hoy en día, a pesar de que sus ventas están aumentando considerablemente, todavía no está siendo la revolución que se esperaba. Dado que su diseño facilita los desplazamientos por superficies poco accidentadas, lo convierte en un método perfecto, por ejemplo, para conocer una ciudad o desplazarse por una feria de grandes dimensiones. Por ello, últimamente se puede observar en algunas ciudades el uso de estos vehículos. Se están empezando a usar para hacer tours por las ciudades o incluso para el patrullaje de la seguridad pública. Cuerpos de Policía Municipal de múltiples Ayuntamientos de España utilizan ya el *Segway* para desempeñar su trabajo. Algunos de ellos son: Alicante, Alcobendas, Castelldefels, Benidorm, Cádiz, Madrid, Ourense, Sant Cugat, Tenerife Sur, Torremolinos o Valencia.

Figura 4: Cuerpo de policía usando *Segway* para patrullar

A pesar de que el *Segway* no ha tenido gran aceptación a la hora de su salida al mercado, hoy en día se sigue fabricando e investigando sobre este tema e incluso han salido al mercado otros dispositivos que sirven para desplazarse. Ejemplo de ellos, podemos tener el WalkCar o IO Hawk entre otros.

Figura 5: IO Hawk

Figura 6: WalkCar

Pero hay muchos nuevos vehículos de esta índole a tener en cuenta, puesto que hoy en día están saliendo al mercado uno tras otro y suponen una forma similar de moverse. Esto quiere

decir que las empresas ven un gran potencial en este ámbito y el desarrollo de la tecnología pueda ser esencial en un futuro.

Un ejemplo de uno de los últimos vehículos de este tipo es el Honda U3-X. Este no se conforma con mantenerse en equilibrio en una sola rueda, lo cual ya es un gran avance, sino que este vehículo puede moverse en cuatro direcciones: hacia adelante, hacia detrás, hacia la derecha y hacia la izquierda. Este movimiento lateral del robot podría suponer una gran revolución en el mercado.

Figura 7: Honda U3-X

También se ha hecho oficial que la empresa *Segway* ha sido comprada en abril de este año por su rival chino, *Ninebot*. Esta empresa, *Ninebot*, está especializada en robótica y uniciclos y está participada por el gigante tecnológico *Xiaomi* (empresa china dedicada al diseño, desarrollo y venta de teléfonos inteligentes, apps y otros productos electrónicos). El producto estrella de la compañía *Ninebot* es un uniciclo, el cual es capaz de mantenerse el equilibrio también. Tras la operación de compra, las dos firmas mantendrán sus marcas pero colaborarán en el desarrollo conjunto de productos conjuntos en los sectores de los vehículos sostenibles en los que la tecnología móvil y de internet también jugará un papel importante.

Figura 8: NinebotOne

2.3 PÉNDULO INVERTIDO

Para construir un robot balancín hace falta combinar diferentes conocimientos sobre matemáticas, física y programación puesto que el robot necesita de algoritmos matemáticos para estabilizar las diferentes fuerzas que se aplican sobre el mismo. Para comprender como el robot balancín se mantiene en equilibrio nos basamos en la teoría del péndulo. En el péndulo invertido se pretende que un móvil, mediante el movimiento hacia adelante y hacia atrás mantenga en equilibrio un péndulo que estaría colocado de manera invertida, es decir, la base del péndulo estaría colocado en el móvil y unido mediante alguna vara o especie de palo en la otra parte de este, tendríamos una masa que oscilaría.

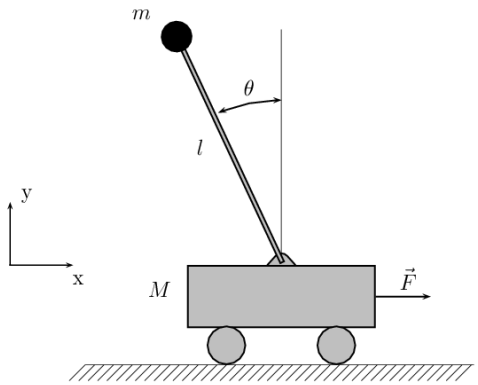


Figura 9: Péndulo Invertido y las fuerzas que actúan en él

El sistema del péndulo invertido es un sistema inestable, puesto que puede caer en cualquier momento a menos que se le aplique una fuerza adecuada. En el péndulo invertido, para conseguir el equilibrio (que el péndulo no caiga) lo que se pretende es que el ángulo (θ) se mantenga a 0°. Para ello aplicaremos fuerzas a las ruedas haciendo que generen una fuerza (F) en la dirección a la que el péndulo se ha desplazado.

El control del péndulo invertido presenta una enorme variedad de problemas que han hecho de él uno de los sistemas concretos para el ensayo de leyes de control más analizados en estos últimos tiempos. Los primeros péndulos invertidos se construyeron en los años 70, y aún hoy, treinta años después, se sigue experimentando con el péndulo invertido y tratando de analizarlo desde un punto de vista teórico. Es notable, que después de tantos años aún no se haya conseguido un estudio unitario y satisfactorio de dicho problema.

En el control del péndulo invertido se presentan básicamente dos problemas: el problema de la estabilidad local en torno a la posición de equilibrio, que es análogo al problema del malabarista que pretende mantener un palo en la punta de un dedo; y el problema de levantar

el péndulo desde su posición de reposo hasta la posición en la que se mantiene erguido hacia arriba. Este último problema se conoce con su denominación inglesa de problema del *swing up.*

El primero de los problemas, el de la estabilización local, puede resolverse satisfactoriamente mediante la linealización en torno a la posición de equilibrio. Sin embargo, la ley de control así obtenida tiene un carácter exclusivamente local. Para perturbaciones de una cierta magnitud el control del péndulo se pierde.

En el problema del swing up es necesario considerar todo el espacio de estados, de modo que el problema no puede abordarse con una perspectiva local. Se trata, por tanto, de un problema al que se aplican las consideraciones relativas a la imposibilidad de linealizar el sistema y de tratarlo exclusivamente en forma local.

Sobre el problema de control del péndulo invertido, conviene resaltar que se puede considerar como un problema arquetipo de lo que es la ingeniería. No existe un método universal para resolverlo, ya que se podría resolver por ejemplo calculando fuerzas necesarias basándonos en la segunda ley de Newton y en la ley de sumatorio de momentos o bien, como lo haremos nosotros, se podría usar un tipo de control PID.

$$\sum F = m \cdot a$$

Segunda ley de Newton

$$\sum M = I \cdot \alpha$$

Ley de Sumatorio de Momentos

3 DESCRIPCIÓN GENERAL

3.1 TECNOLOGÍA UTILIZADA

Para no empezar nuestro proyecto de cero se decidió partir de un robot balancín ya montado y en funcionamiento modificando el microprocesador que traía (freescale MC9S12XS128MAL). Por motivos de averías y demás al final tan solo conservamos el chasis, la batería y las ruedas del robot balancín original. Por tanto, no hablaremos demasiado de estos elementos ya que la web de compra del fabricante no da demasiada información de estos.

3.1.1 Hardware

En el apartado físico de nuestro robot podemos encontrar los diferentes elementos físicos que componen nuestro robot. Entre ellos podemos encontrar controladores de motor, motores, batería y microcontrolador.

Chasis

El chasis usado por el robot es como hemos comentado el que teníamos en la primera versión del robot, ver figura.

Figura 10: Robot balancín del que partimos

Se ha sustituido la placa superior por otra de dimensiones 14,6 cm x 12 cm para sujetar mejor la Explorer16 y las otras dos inferiores son de 15 cm x 7 cm. La altura del robot, incluyendo las ruedas, es de 20,9 cm.

Motores

Después de tener algunos problemas con los motores iniciales decidimos cambiarlos. Los actuales son unos motores de *Metal Gearmotor* con *encoder* incorporado DC de 12 voltios y relación de 29:1. Sus medidas son: 37 cm de diámetro y 52 cm de largo. Estos motores traen incorporados unos *encoders* que tienen una precisión de 64 cuentas por vuelta.

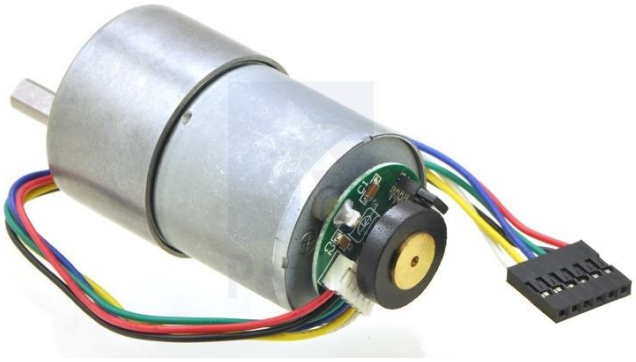


Figura 11: Motor *Metal Gearmotor* con *encoder* incorporado

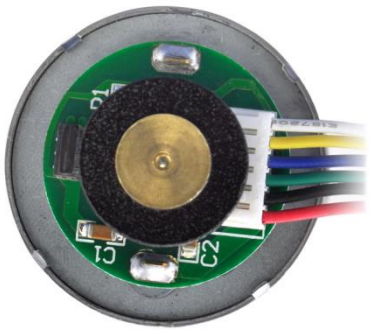


Figura 12: Motor *Metal Gearmotor* con *encoder* incorporado visto por detrás

El motor dispone de 6 cables de diferentes colores. Cada uno tiene la función que se ilustra en la siguiente tabla:

Color	Función
Rojo	potencia del motor positiva
Negro	potencia del motor GND
Verde	*encoder* GND(tierra)
Azul	*encoder*Vcc(alimentación) (3.5 – 20 V)
Amarillo	salida A de *encoder*
Blanco	salida B de *encoder*

Controlador de Motores

En cuanto a los controladores de motor usamos los *Simple-H* de *robot power*. En concreto hemos usado dos, uno para cada motor. A diferencia de otros controladores, estos solo controlan un motor, al menos si queremos hacer que ambos motores puedan girar en ambas direcciones. Soportan mucha corriente, por tanto no hay apenas posibilidades de que se calienten. Estos son los que se encargan de suministrar a los motores la fuerza que el microcontrolador les ordena a los controladores.

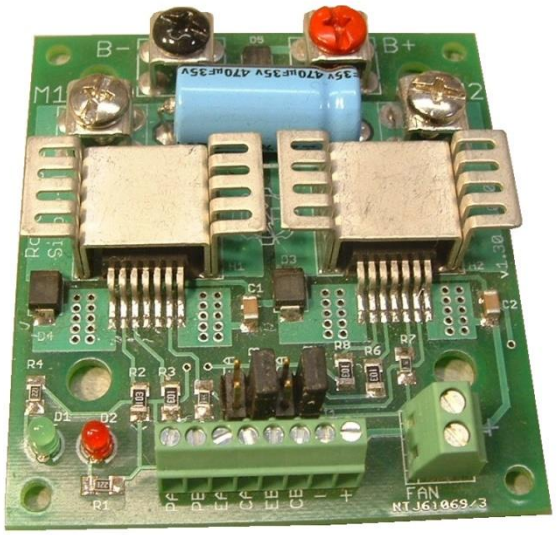


Figura 13: *Simple-H* de *robot power.*

Algunas de las características más importantes de este controlador de motor son las siguientes:

- *Simple-H* de *Robot Power*:
    - Tensión de alimentación entre 5V y 28V.
    - Corriente de salida continúa de 20A y 25A con ventilador.
    - Frecuencia PWM de 20 kHz

Batería

Para alimentar todo el sistema se ha utilizado una batería LiPo (11.8V) de 4200mAh. Con esta batería se alimentan tanto los motores como los componentes electrónicos (IMU, PIC…).

Microcontrolador

Como controlador de vuelo del robot balancín, usamos un microcontrolador PIC de 16 bits, en concreto el PIC24HJ256GP610A, con el que ya estamos familiarizados gracias a la asignatura *Diseño de Sistemas Integrados.*

Casi todos los demás componentes comentados anteriormente están conectados al microcontrolador, de manera que todo el proceso de movimiento del robot balancín está controlado por el microcontrolador. Para la programación del microcontrolador y la depuración del código se ha utilizado el *debugger* REAL ICE y la placa de desarrollo Explorer 16, ambos del fabricante Microchip.

Figura 14: *Debugger* Real Ice conectado al microcontrolador mediante Explorer 16

Las características que más nos importan del microcontrolador son las siguientes:

- Velocidad de CPU: 40 MIPS. -256 KB de memoria de programa.
- 2 buses I^2C.
- 2 módulos RS232.
- 8 módulos de captura de entradas (*Input Capture*).
- 9 *timers* de 16 bits o 4 de 32 bits.
- *Prescaler* ajustable para los *timers*.
- Capacidad de programar casi todo mediante interrupciones, para agilizar la ejecución.

Sensor IMU

La Unidad de Medición Inercial o IMU (*Inertial Measurement Unit)*, es el dispositivo compuesto de diferentes sensores que se encarga de medir la velocidad, la posición y la aceleración (o fuerza gravitacional) sufrida por nuestro robot balancín. Estos datos nos sirven para calcular la fuerza que la gravedad afecta a nuestro robot. En nuestro caso, hemos utilizado la placa 9DOF Stick de Sparkfun, que contiene un acelerómetro, un giroscopio y un magnetómetro (o brújula), todos tridimensionales y conectados al bus I^2C para su configuración y uso, aunque en la práctica tan solo utilizamos dos dimensiones del acelerómetro y uno del giroscopio.

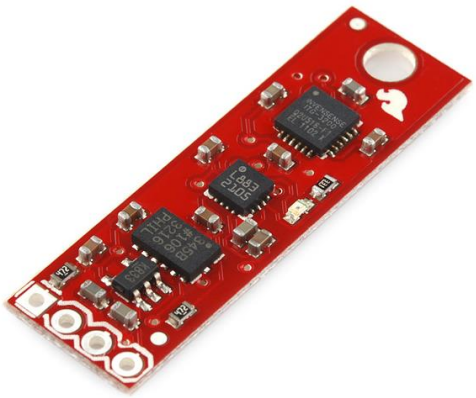


Figura 15: Unidad de Medición Inercial (IMU) 9DOF Stick de Sparkfun

Las características principales de los sensores son las siguientes:

- Acelerómetro ADXL345:
    - El acelerómetro es el instrumento que sirve para medir la aceleración de movimiento. Esta aceleración se suele medir en este caso en fuerzas g(fuerza que la gravedad afecta sobre cualquier objeto en la corteza de la tierra ~9.8 m/s)
    - Resolución ajustable de 10 a 13 bits.
    - Lectura máxima: ±16g (con 13 bits).
    - Sensibilidad de 4mg/LSB, permitiendo detectar cambios menores a 1° en la inclinación.
    - Datos de salida formateados a 16bits en Complemento a 2.
    - Soporta I^2C a 100kHz y a 400kHz, trasmisiones simples y múltiples.
- Giroscopio ITG-3200:
    - El giroscopio es un dispositivo mecánico que sirve para medir la orientación o el cambio de orientación en un dispositivo.
    - 16 bits de resolución.
    - Lectura máxima: ±2000°/s.
    - Sensibilidad de 14,375 LSB/(°/s).
    - Frecuencia de muestreo ajustable de 1 a 8kHz.
    - Frecuencia de salida ajustable de 8.000 a 3,9 muestras por segundo.
    - Datos de salida formateados a 16bits en Complemento a 2.
    - Soporta I^2C a 100kHz y a 400kHz, trasmisiones simples y múltiples, *SlewRate* necesario.
    - Sensor de temperatura integrado.
- Magnetómetro HMC5883L:
    - El magnetómetro es Instrumento para medir la fuerza y la dirección de un campo magnético.
    - 12 bits de resolución.
    - Lectura máxima: ±8 Gauss.
    - Sensibilidad de 2 miligauss/LSB, en grados de 1° a 2°.
    - Frecuencia de muestreo máxima de 160 Hz.
    - Varias muestras para calcular el valor de salida, ajustable de 1 a 8 muestras.
    - Frecuencia de salida ajustable de 0.75Hz a 75Hz (por defecto 15Hz).
    - Datos de salida formateados a 16bits en Complemento a 2.
    - Soporta I^2C a 100kHz y a 400kHz, trasmisiones simples y múltiples.

PC

Se han usado varios PC durante el proceso de desarrollo del proyecto, y se han instalado en los sistemas operativos Windows XP, Windows 7 y Windows 8.1 sin problema alguno. Por tanto, el requisito mínimo sería un pc que pueda albergar Windows XP y el software MPLAB X.

3.1.2 Software

Para programar el microcontrolador del robot balancín hemos usado un programa para desarrollar aplicaciones para microcontroladores Microchip y controladores de señales digitales, MPLAB X. MPLAB X es un entorno de desarrollo integrado (IDE), para desarrollar código para microcontroladores embebidos.

Hemos trabajado con el lenguaje C dentro de este IDE y para traducir este lenguaje C a un lenguaje que el microcontrolador entiende (de 16 bits) se ha usado el compilador XC16.

4 SISTEMA DE CONTROL

Una vez tenemos listo todo el hardware y software para utilizar, comenzamos con lo que es el proyecto en sí. En este apartado se describirán las técnicas usadas para que el robot cumpla los objetivos necesarios, el mantenimiento del equilibrio de forma dinámica e individual y reaccionando a fuerzas externas al robot, y el añadido de funcionalidades extras mediante el uso de los *encoder*.

4.1 LECTURAS DE SENSOR IMU

En este apartado se hablará sobre la fase en la que tratamos sobre la lectura de la unidad de medición inercial IMU. Hablaremos de cómo se conectan los diferentes sensores de éste con la placa y cómo interpretar los valores que obtenemos de ellos para, a partir de ahí, saber el ángulo de inclinación del robot.

4.1.1 Bus de comunicación I^2C

La primera tarea que hemos realizado en cuanto a la construcción del robot consiste en la lectura de los sensores. Esta lectura se realiza mediante el bus de comunicación I^2C (*Inter-integrated circuit*). Este bus de comunicaciones es muy usado en sistemas integrados para la comunicación con sus periféricos como para comunicar varios circuitos integrados entre sí.

En el caso de nuestro robot, usaremos este bus de comunicaciones para comunicar el acelerómetro y el giroscopio. Para implementar este código lo que hemos decidido es reutilizar el código del I^2C que teníamos de otro proyecto y adecuarlo al nuestro.

4.1.2 Lectura de datos de sensores acelerómetro y giroscopio.

Ahora que comprendemos cómo funciona el bus de comunicación I^2C procederemos a explicar cómo leemos estos datos. Para empezar tenemos que llamar a la función de leer los datos pero la primera pregunta que se nos plantea es ¿Cada cuánto tiempo debemos leer los datos? Tras investigar en diferentes proyectos sobre péndulos invertidos y robots balancines asumimos que leer los datos cada 10 milisegundos es un tiempo adecuado para leerlos, ya que no leemos con una frecuencia demasiado alta como para que colapse el funcionamiento del robot y tampoco tan lento para que al robot no le dé tiempo a corregir la trayectoria.

Para que la lectura sea lo más fiel posible respecto a ese tiempo hemos decidido crear una interrupción y en ella gestionar la lectura de los datos y la interpretación de estos, puesto que de nada serviría interpretar los datos sin antes haberlos leído (tendríamos datos de la lectura anterior en vez de la actual que es la que nos interesa).

Una vez que decidimos cada cuanto se actualizan los datos relacionados con la IMU se procedió a programar una interrupción con un temporizador donde se llama a las funciones de leer acelerómetro y leer giroscopio.

```
void _ISR _T7Interrupt (void)// Prioridad 1
{
    I2C1CONbits.SEN =1;
if(estado_int==0){
Leer_Acel();
estado_int=1;
}
//disp=1;
else{
Leer_Giro();
.
.
.
```

Se puede observar que hemos creado una máquina de estados para que así cada llamada lea el acelerómetro o el giroscopio. Esto se debe a que el proceso de lectura de acelerómetro o giróscopo son costosas en cuanto a tiempo y por tanto si hacemos las dos llamadas juntas la posibilidad de que el robot se quedase bloqueado en esos bucles mientras programábamos aumentaban y por tanto nos daba más errores la puesta en marcha después de programar.

4.1.3 Calculo de ángulos partiendo de la lectura de los sensores

Una vez que obtenemos los valores del giroscopio y del acelerómetro con un periodo de 10 milisegundos toca interpretar estos valores.

El giroscopio nos da, cada vez que leemos, la velocidad de giro respecto a la lectura anterior. Es decir, si leemos y a continuación giramos 90 grados y volvemos a leer el giroscopio nos dará un valor que corresponde a esos 90 grados. Pero el dato que nos da el sensor no está en °/seg (grados angulares entre segundos), que es lo que nos interesaría.

Para saber qué tipo de ángulo estamos leyendo es conveniente saber que es la sensibilidad (*Sensitivity*) de un sensor. La sensibilidad de un sensor es el menor cambio de magnitud que un sensor puede percibir, el cual se indica normalmente en LSB/unidad (*Least Significant bit*). En caso del giroscopio nos indica cuantos °/seg equivale cada dato de salida del sensor. Según la

ficha técnica del giroscopio la sensibilidad es de 14.375 LSB/ (°/seg), es decir, si obtenemos el valor de 2 en el giroscopio esto corresponderá a 2*1/14.375 °/seg.

Por tanto una vez leído el valor del giroscopio, tan solo falta multiplicar este valor por 1/14.375 para obtener el dato en °/seg, y para pasar este dato a un ángulo que al fin y al cabo es la unidad que nos interesa tan solo tenemos que multiplicar el dato por el tiempo que se tarda entre lectura y lectura el cual está fijado a 10 milisegundos.

Con todo ello, la formula final quedaría:

$$angulo(°) = X\ LSB * \frac{1\ {}^{\circ}/_{seg}}{14.375\ LSB} * 0.01\ seg$$

Donde X es el valor leído del giroscopio.

El ángulo obtenido mediante esta fórmula representa el ángulo que ha girado desde la última lectura. Por tanto, habría que sumarle al ángulo anterior que teníamos y repitiendo este proceso obtener el ángulo en cualquier momento. Cabe destacar que el giroscopio da valores respecto a los ejes X, Y y Z aunque a nosotros tan solo nos interesa el del eje X ya que es el que representa la inclinación del robot.

El giroscopio presenta dos problemas. El primero es que no sabe dónde están los "0" grados, es decir, tomará los cero grados en el estado en el que el robot se encienda. Y el segundo problema es el error de offset. El error de offset se define como la salida que se presenta cuando a la entrada se introduce el código correspondiente al cero, es decir en nuestro caso sería el valor que el giroscopio devuelve cuando no se ha movido el robot.

Este error hay que tenerlo muy en cuenta puesto que si no lo tomamos en cuenta, el ángulo que vamos leyendo, con cada lectura suma ese offset. Por tanto al de ciertas lecturas el ángulo empezará a desviarse e incrementará ese desvío a medida que el programa sigue en marcha. Para reducir ese error lo que hemos hecho es leer 5000 veces el giroscopio estando este quieto, cuando el robot se pone en marcha. Por tanto calculamos 5000 valores de errores offset. Con estos valores hacemos una media y en cada lectura que hacemos del giroscopio le restamos este offset. Con ello conseguimos que dicho error perjudique lo mínimo en el ángulo obtenido.

Para que el robot calcule el ángulo, no usamos solo el giroscopio puesto que si solo usásemos éste, llegaría un momento que el error acumulado nos haría que el robot cayese. Por ello usamos también el acelerómetro.

El acelerómetro es otro sensor de la IMU que usamos para calcular el ángulo. Este sensor sirve para medir la aceleración o cambio de velocidad y convertirlo en una señal eléctrica que podamos interpretar.

Igual que con el giroscopio, una vez leamos el valor que recibimos del acelerómetro, este nos dará un valor que tenemos que interpretar. Para ello debemos conocer la sensibilidad (*Sensitivity*) del sensor. Mirando el manual podemos encontrar que la sensibilidad del acelerómetro es de 256 LSB/g, es decir, que el dispositivo puede leer 256 valores dentro de

una unidad de la aceleración de la gravedad (9.81 m/s^2). Por tanto utilizando la siguiente formula podríamos descubrir la aceleración que recibe un eje:

$$aceleracion(m/s^2) = X\ LSB * \frac{1\ {}^{m}/_{s^2}}{256\ LSB}$$

**Donde X es el valor que el acelerómetro lee en uno de los ejes*

A diferencia del giroscopio en el que tan solo con leer el valor del eje X podíamos conocer el ángulo de giro, en el acelerómetro necesitamos dos valores(los correspondientes a los ejes X y Z) y a partir de estos calcular el ángulo. Para ello nos apoyaremos en la figura 16.

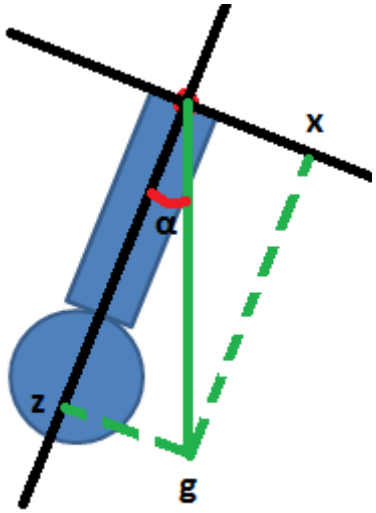


Figura 16: Efecto de la fuerza de la gravedad sobre el robot.

En este dibujo el robot representa la figura azul, el punto donde se cruzan las rectas negras representa donde se encuentra el acelerómetro, las rectas negras los ejes X y Z (el eje y seria perpendicular a estos) y la recta verde continua la fuerza que la gravedad aplica en el acelerómetro (la línea verde discontinua corresponde a la fuerza de la gravedad distribuida en los ejes X y Z). Teniendo en cuenta que desde el acelerómetro podemos medir las fuerzas que la gravedad aplica sobre los ejes X y Z, resulta fácil calcular el ángulo α, puesto que este ángulo representa lo que el robot ha girado desde la posición vertical o bien es lo que hace falta que gire para llegar a la posición perpendicular respecto al suelo.

Para calcular dicho ángulo α debemos calcular el arco tangente de X entre Y, es decir:

$$\alpha(°) = \arctan(X/Y)$$

**Donde X e Y son los valores medidos por el acelerómetro en esos mismos ejes*

El ángulo así calculado no es muy exacto. Este es el mayor problema del acelerómetro, que el ángulo calculado no es exacto y además es extremadamente inestable, tanto que muchas de las veces no se corresponde con la realidad. Para ello, podemos ver la siguiente grafica en donde se ilustra el cálculo del acelerómetro (en rojo) respecto al valor real (azul):

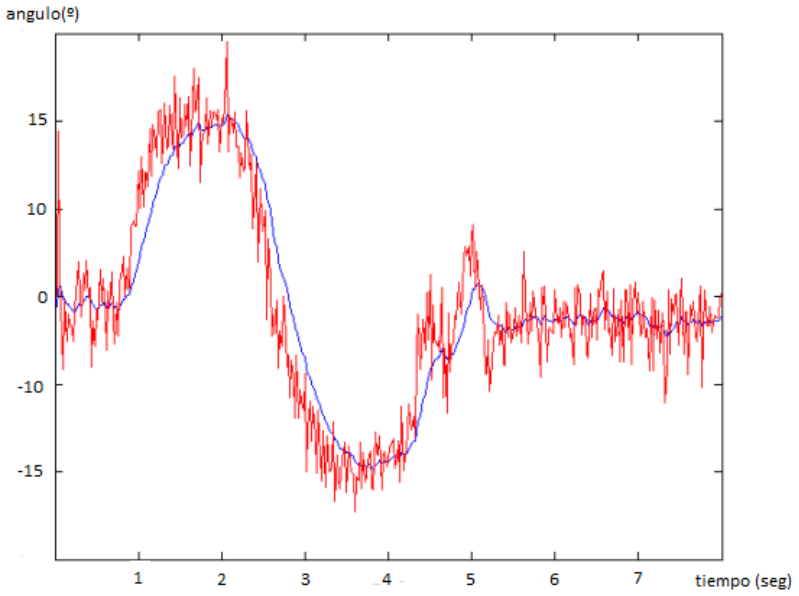


Figura 17: Relación entre los ángulos obtenidos por el acelerómetro y el ángulo real.

Aquí podemos observar que el ángulo medido a veces tiene más de 10° respecto a la realidad por lo que hay que tener mucho cuidado al utilizar este ángulo.

Dado que el ángulo obtenido del acelerómetro no es muy fiable, y el del giroscopio tiene varios errores, surge la idea de que hay que utilizar algún sistema para fusionar estos datos convenientemente, los filtros digitales.

Los filtros digitales son un sistema que, dependiendo de las variaciones de las señales de entrada en el tiempo y amplitud, realizan un procesamiento matemático sobre dicha señal, obteniéndose en la salida el resultado del procesamiento matemático o la señal de salida.

En un comienzo, usamos los filtros más sencillos, en los cuales tan solo cogemos una parte de la señal que cada sensor calcula, por ejemplo:

$$y = anguloGiro * 0.95 + anguloAcel * 0.05$$

**Donde y es la señal de salida, anguloGiro es el ángulo calculado por el giroscopio y anguloAccel el ángulo calculado por el acelerómetro*

En el filtro digital que podemos observar aquí arriba damos más importancia al valor calculado por el giroscopio que al calculado por el acelerómetro. Esto se debe a que la señal del giroscopio es mucho más precisa que la del acelerómetro, que como hemos visto varía mucho,

pero como al mismo tiempo el acelerómetro no tiene los errores que el giroscopio posee tampoco se puede excluir del todo.

Aun así, el comportamiento que obtenemos no es del todo el deseado, ya que el ángulo calculado con este filtro no es tampoco muy fiel a la realidad a pesar de que sea mejor que el de los sensores independientemente. Para mejorar este filtro se decide implementar un tipo de filtro muy común en este tipo de problemas, el filtro de Kalman.

4.2 FILTRO DE KALMAN

Desde su introducción en 1.960 el Filtro de Kalman ha llegado a ser un componente integral dentro de miles de sistemas de navegación tanto militares como civiles. Este algoritmo digital recursivo, aparentemente simple, es el favorito para integrar convenientemente o fusionar los datos de los sensores de navegación para alcanzar un rendimiento óptimo de todo el sistema.

En nuestro proyecto no entramos demasiado a fondo con el filtro de Kalman en cuanto a las fórmulas, debido a que este ya está bastante trabajado e implementado por mucha gente. Además, si quisiéramos entrar a fondo con el filtro de Kalman supondría un trabajo enorme, ya que la teoría del filtro de Kalman abarca muchos temas de las matemáticas. Se dispone de información más específica sobre la teoría del filtro de Kalman en el apéndice. Esta información no es necesaria para implementar el filtro de Kalman debido a que ya existen muchísimas implementaciones del algoritmo en la red.

Para implementar este algoritmo hemos utilizado un modelo extraído del código que tenía el robot balancín del que partimos, en su microprocesador *Freescale MC9S12XS128MAL*. A partir de este modelo, lo hemos adaptado a nuestro robot para que se adecuase a las necesidades de nuestro robot.

Una vez implementado este filtro notamos una gran mejoría en cuanto al ángulo de salida respecto al real, permitiendo que nuestro control del equilibrio del robot mejorase enormemente.

Gracias a la implementación del filtro de Kalman, logramos corregir los errores que el acelerómetro y el giroscopio tienen de por si como ya se ha comentado. Utiliza datos de ambos para conseguir el ángulo “0”, con ello logra que el error del giroscopio al calcular el “0” se solucionará. Además, también se elimina parte del error del acelerómetro ya que al fusionarse los datos le da una mayor importancia al ángulo del giroscopio y por tanto, el ángulo logrado no oscila alrededor de un valor como lo hace si lo calculásemos sólo con el acelerómetro.

4.3 CONTROL DEL EQUILIBRIO

En este apartado se detalla la manera en la que se consigue que el robot se mantenga en equilibrio. Para ello, hay tres cosas importantes. Los controladores de los motores: que entregan la potencia y el sentido en el que tienen que girar las ruedas, el control de los motores mediante PWM - *pulse-width modulation* (modulación por ancho de pulso) y el control PID: que se encarga de calcular el valor del ancho del pulso para el PWM).

4.3.1 Control PWM

El PWM, es una técnica que utilizamos para regular la velocidad de giro de los motores eléctricos. Mantiene el par motor constante y no supone un desaprovechamiento de la energía eléctrica. Aunque aparte de esto, mediante la técnica del PWM también podemos transmitir información analógica. Esto se suele utilizar junto a otros elementos para implementar un conversor ADC.

A diferencia con otros sistemas para el control de los motores, el PWM permite el máximo aprovechamiento de los motores. Los otros sistemas regulan la tensión eléctrica con lo que se disminuye el par motor. Sin embargo, en el PWM, la tensión eléctrica siempre es la misma, lo único que cambia es el tiempo en el que se está ejerciendo dicha energía. Para que se entienda mejor, con un sistema de control que regula la tensión, si queremos que el motor vaya despacio, reduciríamos la tensión eléctrica. Esto haría que el par motor se redujese y que no tuviera a penas fuerza para mover el motor. Mientras que usando la técnica del PWM, al motor siempre se le entregaría la misma tensión eléctrica por lo que el par motor siempre será el mismo. Si queremos que el motor vaya despacio simplemente se modificará el tiempo del ancho del pulso.

En el proyecto se gestionan los motores de manera similar, solo que se utiliza el módulo PWM implementado que ofrece el PIC24H, en el que tenemos que configurar el *Timer* que vamos a utilizar para generar los ciclos.

En el siguiente fragmento de código se puede ver la configuración de inicio del *Timer* que utilizamos para el módulo PWM.

```
    OC1CONbits.OCTSEL =0;// Selección del Timer 2 como timer de
referencia
     T2CON =0;// Reset del Timer
// T2CONbits.TCS = 0; // Selecciona reloj de CPU (Fosc/2)
// T2CONbits.TGATE = 0; // Modo Gated deshabilitado
     T2CONbits.TCKPS =0b00;// Select 1:1 Prescaler
//T2CONbits.TCKPS0=0;
//T2CONbits.TCKPS1=1;
     TMR2 =0x00;// Borra Timer
PR2 = PRT_PWM;// Determina periodo
```

```
//      PRT_PWM = 8000; // Determina periodo
      _T2IP =0x01;// Define nivel de prioridad del Timer 2
      _T2IF =0;// Borra el flag de interrupción Timer 2
      _T2IE =1;// Habilita interrupción Timer 2
      T2CONbits.TON =1;// Activa Timer
```

La configuración de inicio del módulo PWM es la siguiente:

```
// Inicialización del módulo de Salida Comparada
    Motor_A2_Tris=0;//_TRISB10
    Motor_B1_Tris=0;//_TRISB11
    Motor_A1_Tris=0;//_TRISB12
    Motor_B2_Tris=0;//_TRISB13

    OC1CON =0;// Reset del módulo
    OC2CON =0;
    OC3CON =0;
OC4CON =0;
//OC1CONbits.OCTSEL = 0; // Selección del Timer 2 como timer de
referencia
    OC2CONbits.OCTSEL =0;
    OC1R =0;// Define el dutycycle(ciclo de trabajo) para el primer
pulso PWM
    OC2R =0;
OC3R =0;
OC4R =0;

    OC1RS =0;// Define el dutycycle para el segundo pulso PWM
    OC2RS =0;
    OC3RS=0;
    OC4RS=0;

    OC1CONbits.OCM =0b110;// Selecciona modo PWM sin protección
    OC2CONbits.OCM =0b110;
OC3CONbits.OCM =0b110;
OC4CONbits.OCM =0b110;
```

Tanto el OC1, OC2, OC3 y OC4 (módulos correspondientes al PWM implementados por el PIC) se configuran para funcionar con el *Timer 2* del microcontrolador. Si nos fijamos como está configurado el *Timer 2* vemos que cada 8000 pulsos de reloj se reiniciará es decir que tiene un periodo de 8000 pulsos por segundo.

```
     PR2 = PRT_PWM;// Determina periodo
//      PRT_PWM = 8000; // Determina periodo
```

Por lo tanto si queremos dar a los motores la máxima potencia y definir un ciclo de trabajo del 100% a las variables OC1RS, OC2RS, OC3RS y OC4RS que son las que definen el ciclo de trabajo deberíamos asignarles el valor 8000, sin embargo si queremos que el ciclo de trabajo sea del 50% (a la mitad de potencia) deberíamos asignar el valor 4000.

Resumiendo, una vez configurado tanto el Timer como los módulos PWM, basta con asignar a la variable un valor para el ciclo de trabajo, para que el PIC internamente y de forma

transparente al usuario (con el módulo implementado del PWM) gestione los anchos de pulso con el ciclo de trabajo que hemos definido.

Para el caso de nuestros motores no será tan sencillo como definir un simple ciclo de trabajo estático, más bien serán asignaciones dinámicas que estarán en constante cambio, dependiendo del ángulo en el que se encuentre el robot (información extraída del IMU), para así poder mantenerlo en equilibrio. Para controlar esto existen distintos sistemas o controladores, en nuestro caso hemos utilizado un control PID (Proporcional Integrador y Derivativo), que va asignando distintos ciclos de trabajo para dar mayor o menor potencia a los motores y conseguir que el robot se mantenga en todo momento en posición vertical, dependiendo del ángulo del robot (mayor a "0" o menor a "0") cambiara el sentido en el que giran las ruedas y el ancho de pulso.

El código que se muestra a continuación es un fragmento de código encargado de gestionar el ancho de pulso del PWM y la dirección en el que giran las ruedas.

```
if(outputPID>0){

if(dir==1){
dir=0;
                Integral=0.0;
}
            // Motor 1
            OC1RS=abs(outputPID);
            OC3RS=0
            // Motor 2
            OC2RS=abs(outputPID);
            OC4RS=0;
}
elseif(outputPID<0){
if(dir==0){
dir=1;
                Integral=0.0;
}
            // Motor 1
            OC1RS=0;
            OC3RS=abs(outputPID);
            // Motor 2
OC2RS=0;
            OC4RS=abs(outputPID);
}
```

Como se ve en el código, el valor "outputPID" es el que se le asigna a los OCXRS. Este valor ha sido calculado previamente por el controlador PID. Como ya hemos dicho antes, cada motor es controlado por dos señales de PWM, en este caso OC1RS y OC3RS controlan un motor mientras que OC2RS y OC4RS controlan el otro motor, por lo que si el "outputPID" da un valor mayor a "0" (el robot está inclinado hacia un lado) la asignación del ancho del pulso es:

```
// Motor 1
            OC1RS=abs(outputPID);
            OC3RS=0;
            // Motor 2
```

```
            OC2RS=abs(outputPID);
            OC4RS=0;
```

Esto hace que ambos motores giren en el mismo sentido y a la misma velocidad. Por el contrario si la variable “outputPID” tiene un valor menor a “0” (el robot está inclinado hacia el otro lado) la asignación del ancho del pulso es:

```
// Motor 1
            OC1RS=0;
            OC3RS=abs(outputPID);
            // Motor 2
OC2RS=0;
            OC4RS=abs(outputPID);
```

4.3.2 Controladores de los motores

Para controlar los motores hemos utilizado dos controladores *Simple-H* de *robot power. A* cada uno de ellos está conectado un motor y gracias a estos controladores controlamos la velocidad y el sentido de cada motor.

Cada *Simple-H* de *robot power* recibe dos señales PWM del microcontrolador. Para uno de los motores, los pines OC1 y OC3 del microcontrolador están conectados a los pines PA y PB del controlador del motor, y para el otro motor los pines OC2 y OC4 del microcontrolador están conectados a los pines PA y PB del otro controlador del motor. La salida del *Simple-H* de *robot power* va conectado al motor.

Para que el motor se mueva, una de las dos señales debe estará activa (estado lógico alto), y la otra en baja. Para que esté parado ninguna de las dos señales debe estar activa. El sentido de giro del motor se controla mediante las dos señales de los pines PA y PB. Para que gire hacia un lado, la señal PA debe estar en estado lógico alto y la señal PB en estado lógico bajo y para que gire hacia el otro lado se intercambian los valores de PA y PB.

4.3.3 Controlador PID (PROPORCIONAL INTEGRAL DERIVATIVO)

Para conseguir que el robot se mantenga en equilibrio, hemos utilizado un controlador PID.

Un controlador PID es un mecanismo de control genérico sobre una realimentación de bucle cerrado. El funcionamiento del controlador PID es como sigue: al sistema, es decir, al controlador, le entra un error calculado a partir de la salida deseada y la salida obtenida. Ese error es utilizado como entrada en el sistema que queremos controlar. El controlador intenta minimizar el error ajustando la entrada del sistema.

Para el correcto funcionamiento de un controlador PID que regule un proceso o sistema se necesita, al menos:

- Un sensor, que determine el estado del sistema (termómetro, manómetro, acelerómetro…).
- Un controlador, que genere la señal que gobierna al actuador.
- Un actuador, que modifique al sistema de manera controlada (resistencia eléctrica, motor, válvula, bomba…).

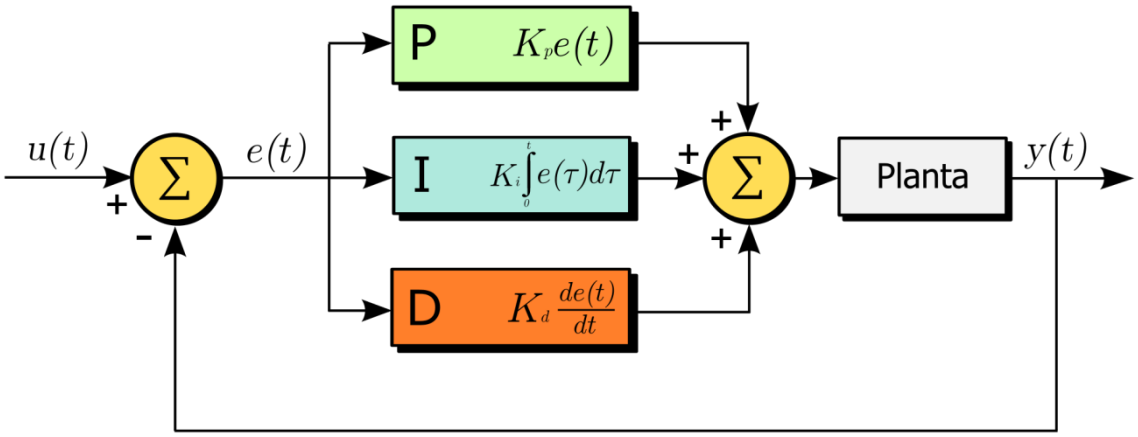


Figura 18: Diagrama de bloques de un controlador PID de lazo cerrado

u (t): Entrada (Salida que se desea obtener).
y(t): Salida.
e(t): Error.
Planta: Sistema.

El controlador PID se ajusta en base a tres parámetros: el proporcional, el integral y el derivativo. Dependiendo la modalidad del controlador alguno de estos parámetros pueden valer “0”. Podemos crear un controlador proporcional P, en el que el valor integral y el derivativo son “0”, o un controlador proporcional e integral PI, en el que el valor derivativo es “0”, o también un controlador proporcional derivativo PD en el que el factor integral es “0”. Finalmente el que hemos utilizado para el proyecto es el controlador PID en el que los tres parámetros son distintos de cero. Cada uno de estos parámetros influye sobre alguna característica de la salida: tiempo de establecimiento, sobre oscilación, etc. También influye, en mayor o menor medida sobre las demás, por lo que por mucho que ajustemos los parámetros no encontraríamos un PID que redujera el tiempo de establecimiento a “0”, la sobre oscilación a “0”, el error a “0”, etc… Se trata más de ajustarlo a un término medio cumpliendo las especificaciones requeridas.

Respuesta proporcional

La respuesta proporcional es la base de los tres modos de control, si los otros dos, integral y derivativo están presentes, éstos se suman a la respuesta proporcional. Se obtiene una respuesta proporcional, es decir se le aplica un factor multiplicador a la entrada del controlador. A este múltiplo se le llama "ganancia" del controlador.

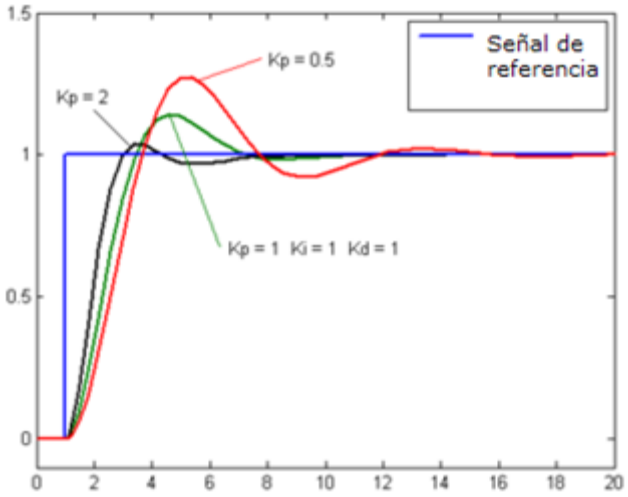


Figura 19: Respuesta proporcional

Respuesta integral

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

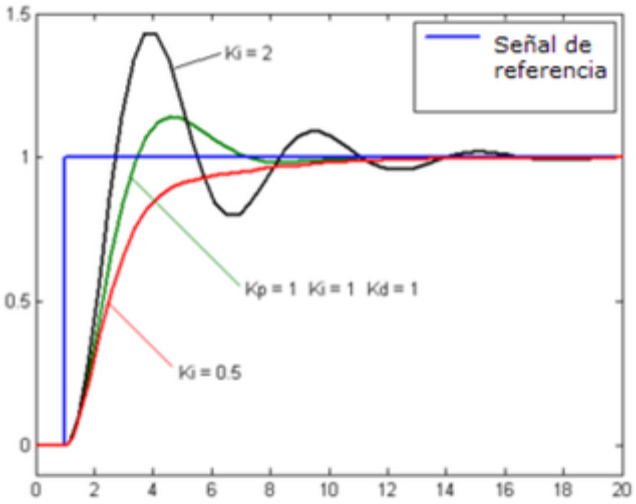


Figura 20: Respuesta integral

Respuesta derivativa

La acción derivativa da una respuesta proporcional a la derivada del error o dicho de otra forma, velocidad de cambio del error. Añadiendo esta acción de control a las anteriores se disminuye el exceso de sobre oscilaciones.

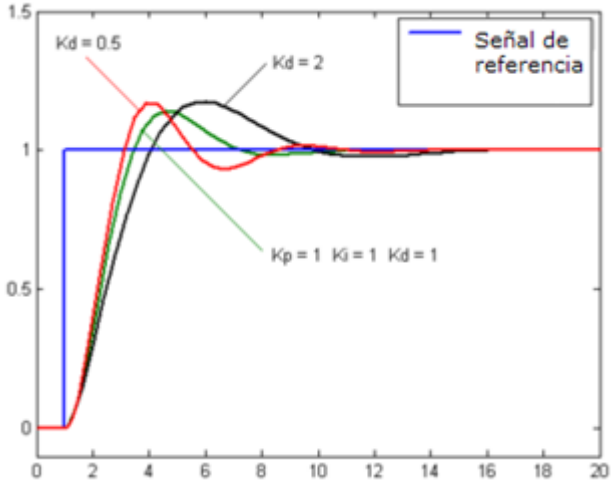


Figura 21: Respuesta derivativa

En resumen, un controlador PID, lo que hace es: a partir de una señal de referencia, intenta ajustar la salida a la señal de referencia aplicando una función al error obtenido, entre la salida y la señal de referencia. Esa función se ajusta mediante tres parámetros P, I, D (Kp, Ki, Kd) y en base a esto, el sistema obtiene una respuesta mejor o peor.

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

Para ajustar los parámetros existen distintos métodos: el método de Ziegler-Nichols, el método de Cohen-Coon, el método que sigue unas reglas heurísticas de ajuste y por último ajustándolo a ojo, que viene a ser algo parecido a seguir las reglas heurísticas de ajuste.

En nuestra solución, primero ajustamos el Kp, de forma que el robot se mantenga más o menos en equilibrio. Una vez encontrado ese valor, ajustamos el Kd, para que las oscilaciones disminuyan y finalmente ajustamos el Ki, para eliminar el error en régimen estacionario.

Tras diversas pruebas, los valores Kp, Kd y Ki con los que mejor respuesta hemos obtenido han sido los siguientes:

Kp = 330.0

Ki = 25.25

Kd= 1150.0

La señal de referencia para nuestro controlador PID es el ángulo de inclinación del robot, que idealmente debe tener siempre el valor 0°. Es decir, el robot se debe encontraren posición vertical. En caso de que se incline hacia un lado, tendrá un ángulo mayor a 0° y en caso de que se incline hacia el otro tendrá un ángulo menor a 0°. Como se ve en la siguiente figura el objetivo es que el robot mantenga en todo momento, respecto al eje vertical (Z), un ángulo (α) de 0°.

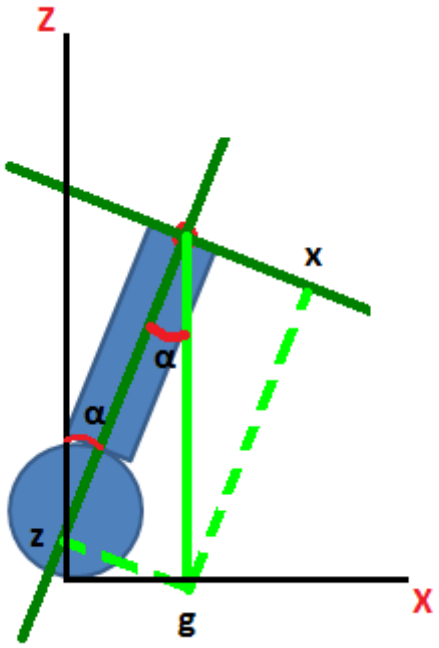


Figura 22 Fuerza de la gravedad sobre el robot, el ángulo α representa el ángulo que necesita corregirse para que el robot se quede en posición vertical

La entrada del controlador PID, es el propio ángulo del robot, que previamente ha sido procesado por el filtro de Kalman, y el valor de referencia objetivo o *setpoint* es el valor "0" (0°).

El control PID que hemos implementando funciona de la siguiente manera:

1- Se obtiene como entradas la señal de referencia, y la salida obtenida anteriormente del sistema, que en este caso es el ángulo que tiene el robot.
2- Se calcula el error entre el valor de referencia y el ángulo y se aplica el control PID.
3- Se obtiene como salida un valor, ese valor es el ciclo de trabajo del PWM para los motores del robot. Es decir, se regula la velocidad de las ruedas del robot en base al controlador PID, y se le aplica a los motores para corregir el ángulo.
4- Se vuelve al paso 1.

La implementación es la siguiente:

```
voidcalcularPID(){
    error=anguloFiltro-SETPOINT;
Proportional=Kp*error;
Derivative=error-prev_error;

if((abs(error)<=MARGEN_INTEGRADOR))
{
      Integral+=Ki*error;
if(Integral>LIMITE_PWM)Integral=LIMITE_PWM;
elseif(Integral <-LIMITE_PWM) Integral=-LIMITE_PWM;
}
//-----------------------------------------------------------------
k_k=Proportional+ Integral +Kd* Derivative;

//Limites de la salida en función de la configuración del timer del
PWM, para nuestro caso es 8000.
if(k_k<-LIMITE_PWM)
{
k_k=-LIMITE_PWM;
}elseif(k_k>=LIMITE_PWM)
{
k_k=LIMITE_PWM;
}

auxoutputPID=(int)k_k;
outputPID=auxoutputPID;
prev_error=error;
}
```

La función *calcularPID(),* se le llama desde la rutina de atención de interrupción del *timer7* que tiene la siguiente configuración:

```
void Inic_Timer_7(void)//Función de inicialización del TIMER7
{
// Inicializar y habilitar TIMER7
    T7CONbits.TON =0;// DisableTimer
T7CONbits.TCS =0;// Select internal instruction cycle clock
```

```
    T7CONbits.TGATE =0;// Disable Gated Timer mode
T7CONbits.TCKPS =0b10;// SelecionarPrescaler: 1:64
TMR7 =0x00;// Clear timer register
    PR7 =3125;// PeriodoGiroscopio: 10ms = 6250
//  _T7IP = 0x01; // Set Timer 7 Interrupt Priority Level
    _T7IP =0x05;// Set Timer 7 Interrupt Priority Level
    _T7IF =0;// Clear Timer 7 Interrupt Flag
    _T7IE =1;// Enable Timer 7 interrupt
    T7CONbits.TON =1;// Start Timer

estado_int=0;
t1=clock();
errorFlag=1;
}
```

Por lo tanto a esta función se le llama cada 10 milisegundos.

4.4 *ENCODERS*

Un *encoder* de motor es un codificador rotatorio, que habitualmente es un dispositivo electromecánico usado para convertir la posición angular de un eje a un código digital, lo que lo convierte en una clase de transductor. Estos dispositivos se utilizan en robótica, en lentes fotográficas de última generación, en dispositivos de entrada de ordenador (tales como el ratón y el *trackball*), y en plataformas de radar rotatorias. Hay dos tipos principales: absoluto e incremental (también llamado relativo).

En el *encoder* absoluto se produce un código digital único para cada ángulo distinto del eje. Se corta un patrón complejo en una hoja de metal y se pone en un disco aislador, que está fijado al eje. También se coloca una fila de contactos deslizantes a lo largo del radio del disco. Mientras que el disco rota con el eje, algunos de los contactos tocan el metal, mientras que otros caen en los huecos donde se ha cortado el metal. La hoja de metal está conectada con una fuente de corriente eléctrica, y cada contacto está conectado con un sensor eléctrico separado. Se diseña el patrón de metal de tal forma que cada posición posible del eje cree un código binario único en el cual algunos de los contactos esté conectado con la fuente de corriente (es decir encendido) y otros no (apagados). Este código se puede leer por un dispositivo controlador, tal como un microprocesador, para determinar el ángulo del eje.

Sector	Contacto 1	Contacto 2	Contacto 3	Ángulo
1	OFF	OFF	OFF	0º a 45º
2	OFF	OFF	ON	45º a 90º
3	OFF	ON	ON	90º a 135º
4	OFF	ON	OFF	135º a 180º
5	ON	ON	OFF	180º a 225º
6	ON	ON	ON	225º a 270º
7	ON	OFF	ON	270º a 315º
8	ON	OFF	OFF	315º a 360º

Figura 23: Tabla que podría interpretar las lecturas del *encoder* para convertirlo en ángulo

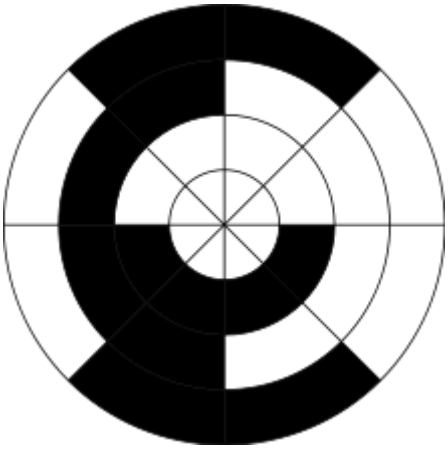


Figura 23: Codificador rotativo para dispositivos de medidas de ángulo

Generalmente, si hay "n" contactos, el número de posiciones distintas del eje es 2^n. En este ejemplo n es 3, es decir, 2^3 lo cual nos da 8 posiciones.

En nuestro caso no usamos un *encoder* absoluto, sino uno relativo o incremental. Este método también utiliza un disco unido al eje, pero este disco es mucho más pequeño marcado con una gran cantidad de líneas de la parte radial como los radios de una rueda. Un interruptor óptico, equivalente a un fotodiodo, genera un pulso eléctrico cada vez que una de las líneas pase a través de su campo visual. Un circuito de control electrónico cuenta los pulsos para determinar el ángulo con el cual el eje da vuelta.

Este sistema, en su forma más simple, no puede medir el ángulo absoluto del eje. Puede medir solamente el cambio en ángulo, tal como posición del eje cuando desde que empezamos a alimentar a los *encoder*. Cuando la posición absoluta debe ser conocida, un segundo sensor puede ser agregado que detecta que el eje pasa por su posición cero. Este problema no se daría en sistemas como ratones donde tener la posición absoluta no es importante, sino que nos importa el cambio que presenta el *encoder*.

El segundo problema de este sistema es que no puede decir qué dirección está rotando el eje. Para superar este problema, el sensor óptico se debe aumentar a dos sensores colocados en ángulos diferentes alrededor del eje. La dirección de rotación se puede deducir en orden en la

cual los dos sensores detecten cada línea radial. Este tipo de codificador se conoce como codificador de cuadratura.

En nuestro caso nuestro *encoder* es del tipo relativo magnético. Los sensores de este *encoder* se llaman sensores Hall y trabajan con el magnetismo. A diferencia de los otros *encoders* ópticos, éstos pueden trabajar en condiciones muy extremas, donde las temperaturas varían mucho y hay polvo en abundancia. Además, el precio no varía mucho respecto a los ópticos. Los motores que utilizamos disponen, en la parte posterior del eje, de un disco magnético con dos sensores que se encargan de leer y generar pulsos independientes, con los que podemos conocer el cambio de ángulo y su dirección. Este codificador tiene una resolución de 64 CPR (*counts per revolution*), es decir, es capaz de contar 64 pulsos en una revolución o giro completo del eje.

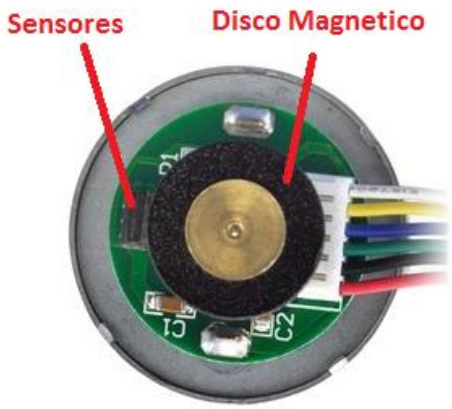


Figura 24: Vista posterior del motor donde se observa el *encoder*

Los sensores de los *encoders* requieren de un voltaje de entrada de entre 3.5V y 20V y pueden tener una corriente máxima de 10mA. Una vez leamos los sensores tendremos unos impulsos. La frecuencia entre las transacciones de los impulsos nos dirá la velocidad del motor y la dirección.

Figura 25: Output de los dos *encoder* de una rueda cuando esta avanza

Contando los picos y caídas de cada uno de los *encoders* A y B, es posible contar 64 veces por revolución del motor.

Una vez vista la teoría del *encoder*, se procede a leer los pulsos que genera. Para ello, lo primero es hallar la frecuencia a la que debemos leer los *encoders*. Partimos de las velocidades de los motores. La velocidad máxima de los motores es de 365 rpm, es decir que gira 365 veces en un minuto lo cual es casi equivalente a 6 vueltas por segundo.

$$365\frac{revoluciones}{minuto} * 1\frac{minuto}{60\ segundos} \sim 6.083\ \frac{revoluciones}{segundo}$$

Puesto que el *encoder* cuenta 64 veces por cada revolución quiere decir que puede leer hasta un máximo de aproximadamente 390 marcas por segundo.

$$6.083\ \frac{revoluciones}{segundo} * 64\frac{marcas}{revolucion} = 389.33\frac{marcas}{segundo}$$

Es decir, lee una marca cada 0.0026 segundos aproximadamente. Los segundos lo pasamos a milisegundos y obtenemos que aproximadamente cada 2.5 milisegundos será el tiempo al que deberíamos leer los *encoders* para ver si han cambiado e interpretar ese cambio. Con este

tiempo de lectura el *encoder* no debería saltarse ningún cambio a pesar de que el motor fuese a máxima velocidad.

La lectura del cambio no nos dice mucho con cada sensor individualmente, ya que son unos pulsos de subida y bajada que indican que las ruedas están en movimiento. Pero si la rueda está funcionando hacia delante y hacia detrás estas lecturas de poco servirían. Para ello usamos el otro *encoder* y diferenciamos 4 fases de los estados de *encoders*.

	estado 1	estado 2	estado 3	estado 4
encoder 1	Low	High	High	Low
encoder 2	Low	Low	High	High

En cada lectura el *encoder* se encontrará en uno de los cuatro estados, y si tras leer el *encoder* se determinara en qué estado se encuentra. Si el estado aumenta significará para nosotros que el robot está caminando hacia adelante y si se reduce un estado significara que el robot camina hacia atrás. Hay que tener en cuenta que si en la lectura anterior estábamos en el estado 1 y tras la siguiente lectura nos encontramos en el estado 4 se tomará como si fuésemos hacia atrás y lo contrario si estamos en el estado 4 y en la lectura anterior estábamos en el 1, es decir, que anda hacia adelante.

Una vez podemos interpretar los pulsos que el *encoder* lee, podemos conseguir la distancia recorrida. Para calcular la distancia recorrida tan solo debemos llevar una cuenta del número de marcas que una rueda ha girado en una dirección y restarle los de la otra dirección, consiguiendo finalmente una suma de marcas totales hacia uno u otro lado. Con ello tendríamos el número de marcas que avanza o se retrasa el robot. Para convertirlo en distancia primero debemos saber que distancia se avanza al girar la rueda una vuelta. Para ello tenemos la ecuación de perímetro de una circunferencia.

$$perimetro = \pi * diametroRueda$$

Una vez sabemos esto y el número de cuentas que el *encoder* realiza por vuelta, debemos dividir el número de cuentas que el *encoder* ha realizado por el de una vuelta:

$$numero\ Vueltas = \frac{lecturas\ realizadas\ por\ el\ encoder}{lecturas\ que\ el\ encoder\ detecta\ por\ vuelta}$$

Una vez tenemos el número de vueltas que se han realizado por rueda y el diámetro de estas multiplicamos los valores para obtener la distancia avanzada o retrocedida.

$$distancia = perimetro * numero\ Vueltas$$

No se ha encontrado mucha información sobre la programación y uso de *encoders* y lo encontrado era muy difícil de entender, por ello, todo el código desarrollado ha sido de cosecha propia siguiendo esta teoría.

También se ha realizado un cálculo de distancia total recorrida, es decir, la distancia total independientemente de si va para adelante o para atrás. Esto es más sencillo ya que una vez se detecta una transición de estado sumamos un valor a la lectura y luego aplicando las mismas fórmulas que para calcular la distancia avanzada o retrocedida, se calcula la distancia total recorrida.

Una vez tenemos estas, también es interesante calcular la velocidad a la que el robot se está moviendo. Para ello disponemos de la distancia que ha avanzado entre lectura y lectura y calculando el tiempo que ha pasado entre lectura y lectura del *encoder*, se puede calcular la velocidad con la siguiente formula:

$$velocidad = \frac{distancia}{tiempo}$$

Por último, también se dispone a que el robot vuelva al sitio donde se puso en marcha tras recibir una fuerza que lo desplace del sitio. Para ello se parte de la distancia avanzada o retrocedida del robot, puesto que con ello podemos saber si el robot necesita moverse para volver a la posición inicial. Una vez sabemos si el robot ha de avanzar o retroceder tan solo habría que darle potencia a los motores en dicha dirección. Pero esto ha supuesto un gran problema a la hora de implementarlo, puesto que si lo hacemos teniendo el robot quieto, es decir, desactivando el módulo correspondiente a los valores del PID que otorga valores a los motores, este funciona bien. Pero a la hora de activar el módulo del PID, dado que el hecho de enviarle potencia al motor directamente supone un desequilibrio, el robot automáticamente corrige la potencia extra que se envía y contrarresta la fuerza extra que se envían a los motores mediante el control PID, haciendo casi inútil dicho esfuerzo.

Con ello lo que se ha logrado es que el robot para corregirse no oscile tanto y que se mantenga más quieto en una posición, pero como he dicho antes, no funciona del todo correctamente y por ello si se le mueve al robot este tiende a quedarse quieto en la posición a la que se ha movido.

5 PROBLEMAS ENCONTRADOS

En este apartado se detallan los problemas que han ido surgiendo durante el desarrollo del proyecto, el retraso que nos han causado y la solución final que se le ha dado al problema.

Este apartado está dividido en dos partes, en la primera parte se describen los problemas encontrados en la primera fase (La fase que hemos hecho en conjunto Markel Picado y yo) y en la segunda parte los problemas relacionados con la segunda fase del proyecto.

Con este apartado lo que se pretende es que se comprenda los retrasos que hemos sufrido en la primera fase concretamente y que están reflejados en el diagrama de Gantt así como facilitar el trabajo a futuros estudiantes o ingenieros que quieran abordar un proyecto de estas características.

5.1 CONTROLADORES DE LOS MOTORES

Antes de acabar utilizando los controladores *Simple H robot power,* utilizamos otros modelos distintos, el TB6612FNG *Dual Motor Driver Carrier* y el L293D.

Para ambos modelos el problema era prácticamente el mismo, el sobrecalentamiento, debido a algún problema que desconocemos, y que no llegaban a entregar la potencia suficiente a los motores. El problema de la potencia lo descubrimos demasiado tarde, ya que aunque los motores funcionases hacia ambos sentidos como no teníamos otro robot con el que poder comprar no sabíamos si realmente funcionaban a su potencia máxima. Debido a esto el robot no llegaba a rectificar el ángulo y a mantenerse en equilibrio.

Con el TB6612FNG*Dual Motor Driver Carrier* mientras hacíamos las pruebas básicas para comprobar el comportamiento de los motores (moverse hacia delante y hacia detrás) el controlador se prendió fuego, como consecuencia directa el controlador dejo de funcionar y cambiamos al controlador L293D. Una vez hecho el cambio, nos dimos cuenta que aparte de quemar el controlador, también se quemaron algunas de las patas a las cuales estaba conectado el primer controlador, por lo que tuvimos que sustituir las conexiones.

Aun con el nuevo controlador, no conseguíamos que el robot se mantuviera en equilibrio, por lo que empezamos a sospechar que el problema podría ser de los motores.

Finalmente descubrimos que el problema era tanto del controlador porque no entregaba la potencia suficiente como de los motores que no funcionaban correctamente, y los sustituimos por las piezas que tenemos ahora.

La diferencia entre los controladores actuales respecto a los anteriores es que los anteriores cada uno podía controlar dos motores, mientras que los actuales solo pueden controlar un motor cada uno y es por eso por lo que tenemos dos controladores de los motores, uno para cada rueda.

5.2 MOTORES

Como se ha comentado en el apartado anterior, llegamos a las sospechas de que los motores no funcionaban correctamente. Para comprobar el funcionamiento de estos motores, volvimos a conectar la placa original que venía con el robot, la que compramos, para saber si aún seguía funcionando correctamente y así descartar el problema de los motores.
Para nuestra sorpresa, vimos que con la placa anterior, es decir, tal y como nos vino el robot según lo compramos, tampoco funcionaba. Los motores se comportaban de forma rara y no llegaba a mantenerse el robot en equilibrio.

Haciendo comprobaciones con distintas herramientas (osciloscopio, multímetro…) se descubrió que hacía falta un ciclo de trabajo del 50% para que los motores comenzaran a moverse, lo que nos limitaba a la hora de controlar el robot en inclinaciones cerca de los 0° y para cuando comenzaban a moverse los motores ya era demasiado tarde porque el robot se caía. Aparte de esto, la potencia que le llegaba a los motores desde el controlador no era la máxima como ya se ha comentado.

Finalmente decidimos comprar unos motores nuevos que tuvieran unas características adecuadas para este tipo de robot.

5.3 EXPLORER 16

La *Explorer 16* dispone de unos reguladores de voltaje los cuales impiden el paso de voltajes superiores de 5V y de 3.3V. Mientras trabajábamos nos dimos cuenta que estos reguladores se calentaban en exceso, hasta que un día se quemó uno de ellos. A pesar de ello la *Explorer 16*seguía funcionando correctamente y por ello seguimos usándola. Un mes después de que ocurriese esto y cuando ya se mantenía en equilibrio correctamente, la placa dejo de funcionar por motivos desconocidos, por lo que nuestras sospechan apuntan a que debió ser por el regulador roto.

Una vez cambiado el *Explorer 16*y volver a soldar todas las conexiones necesarias para que el robot funcionase correctamente, el comportamiento del robot cambió y ya no se mantenía solo en equilibrio. Debido a esto tuvimos que cambiar la configuración del robot. Ya que este problema ocurrió bastante tarde (Principios de Julio) no disponíamos del suficiente tiempo como para dejarlo tan bien como antes, y aunque el problema está solucionado parcialmente

porque el robot se mantiene en equilibrio, no se mantiene tanto tiempo como antes que prácticamente podía estar el tiempo que quisiese en equilibrio.

Creemos que este problema (al cambiar de una placa a otra placa) es debido a que los ajustes que teníamos realizados sobre la primera *Explorer 16* eran unos ajustes que ayudaban a corregir los defectos que esta placa tenia.

5.4 AJUSTE PID

Como ya se ha mencionado, el ajuste de los valores Kp, Kd y Ki los hacemos a ojo, por lo que cada vez que teníamos un problema de los mencionados anteriormente teníamos que reajustarlos, cosa que tomaba bastante tiempo.

Durante un largo tiempo, entre 2 y 3 meses, estuvimos trabajando en el ajuste del controlador PID sin resultados satisfactorios, hasta que descubrimos que teníamos problemas tanto con los motores como con los controladores.

5.5 CONVERSIÓN DE DATOS

En un principio el controlador PID nos proporcionaba valores en un formato *double*, los cuales eran muy grandes. Al realizar la conversión del tipo *double* al tipo entero (*int*), ya que el *double* es mucho más grande que el entero (8 bytes el *double* y 4 bytes el entero), este no cabía en el entero dándole un valor desconocido que no era el valor que verdaderamente esperábamos.

En un principio teníamos limitado el valor del PID (la variable k_k) después de la conversión (`auxoutputPID=(int)k_k`) como se puede ver en el siguiente código:

```
k_k=Proportional+ Integral +Kd*Derivative;
auxoutputPID=(int)k_k;

if(k_k<-LIMITE_PWM)
   {
       k_k=-LIMITE_PWM;
   }elseif(k_k>=LIMITE_PWM)
   {
       k_k=LIMITE_PWM;
```

Debido a ello los valores que obteníamos no eran los deseados y algunas veces incluso funcionaban hacia el lado contrario al deseado. Para solucionar esto se decidió limitar el valor que obteníamos del PID antes de realizar la conversión. Con ello acotamos el valor *double* en un rango de valores que al convertirlos a entero no da problemas de conversiones, con lo cual logramos el dato esperado.

```
k_k=Proportional+ Integral +Kd*Derivative;


if(k_k<-LIMITE_PWM)
{
k_k=-LIMITE_PWM;
}elseif(k_k>=LIMITE_PWM)
{
k_k=LIMITE_PWM;
}

auxoutputPID=(int)k_k;
```

5.6 *ENCODER*

Como se ha explicado anteriormente a la hora de querer que el robot vuelva a su posición, el robot no responde correctamente. Esto se debe a que lo que hacemos es darle una fuerza extra a los motores en la dirección adecuada para que el robot vuelva a la posición inicial. Esto a su vez supone un desequilibrio para el robot, por lo cual el control PID del robot automáticamente lo detecta y anula dicha fuerza. Debido al tiempo escaso tenido al final del proyecto y que no encontrábamos una solución para el problema sin tener que modificar todo el control PID, lo cual podía suponer otra vez tener que trabajar en el control de equilibrio y esto requiere mucho tiempo, dejamos este problema sin solucionar. A la hora de controlar por Bluetooth mi compañero Markel Picado ha tenido un problema similar, por el cual no ha podido hacer que el robot avance o retroceda en línea recta.

6 FUTUROS DESARROLLOS

A pesar de que la mayoría de objetivos han sido cumplidos, todavía han quedado algunos apartados sobre los que podrían trabajarse para que el proyecto mejorase. Tanto para mejorar la funcionalidad que posee como para aumentar dichas funcionalidades o incluso en la estética.

Para empezar, el equilibrio del robot no ha quedado totalmente resuelto. Se ha conseguido que el robot se mantenga en equilibrio durante más de 10 minutos, pero oscila demasiado y acaba moviéndose de su posición original. Además, el tiempo que tarda en corregirse frente a una fuerza externa podría considerarse demasiado alto, puesto que utiliza mucha distancia y bastante tiempo corregirse de un error.

Para mejorar esto se podría trabajar sobre diferentes partes:

- La unidad de medición inercial o IMU la cual podría ajustarse en una posición mejor y más fija, puesto que donde la tenemos ahora tiene un error que no hemos podido quitarle.
- También se podría trabajar en el filtro de Kalman. A pesar de que este filtro sea muy usado en este tipo de problemas y que es realmente útil, toma el valor de la posición "0" a partir del acelerómetro. Por tanto si éste no está bien ajustado el filtro desequilibra el ángulo obtenido. Además, tiene diferentes valores que apuntan a la predicción del futuro. Quizás jugando un poco con ellos se podría mejorar la respuesta de este filtro.
- Por último, tenemos el ajuste del PID. A pesar de que mucha parte del proyecto se ha dedicado a este apartado, se podría decir que ha sido el más desastroso. Debido a los problemas antes comentados, este ajuste lo hemos llevado a cabo con muchos errores encima y varias veces, y dado que es una labor bastante ardua no ha sido sencillo llevarla a cabo puesto que requería de mucho tiempo para ajustar estos valores. Para mejorar esto hemos usado diferentes tipos de ajustes no solo el PID, sino también el PD (hay proyectos que con un control PD consiguen unos resultados increíbles), haciendo que la parte derivativa se calcule también directamente con la lectura de la velocidad de giro del giroscopio o incluso también intentando un sistema de control PI o tan solo P. Dado que el comportamiento no es del todo satisfactorio se podría trabajar en este área y quizás conseguir que el robot se rectificase más eficazmente y se mantuviese quieto del todo.

Por último en el apartado de los *encoders*, se han logrado calcular diferentes datos. Uno de ellos representa la distancia total recorrida, es decir avanzando hacia adelante o hacia atrás la suma de ambas distancias. Esta ha sido calculada respecto a una rueda, la derecha. Por tanto si el robot girase sobre la rueda una vuelta, ésta contaría que ha avanzado una distancia cuando realmente no lo ha hecho. Este tipo de error podría ser más estudiado y mejorado.

En cuanto al apartado de *encoders* también podría mejorarse la parte en la que el robot vuelve a la posición inicial. Ya que el sistema PID automáticamente corrige la potencia enviada a los motores no he sido capaz de hacer que el robot vuelva a la posición de origen satisfactoriamente. Este error quizás podría solucionarse en vez de jugando con la potencia de los motores, jugando a modificar el ángulo del robot haciendo creer que está más inclinado hacia una zona y por tanto obligándole a corregirse dando potencia hacia el otro lado. Esta ocurrencia ha sido de última hora y no ha podido ser probada, pero estaría bien que se probase y ejecutase para ver si resulta efectiva.

Por último, dado que la fase de control vía Bluetooth y la de control de *encoders* ha sido desarrollada por diferentes personas (la de control vía Bluetooth por Markel Picado y la de los *encoders* por mí), esta no se ha puesto en común. Una gran mejoría del proyecto sería la de fusionar ambos códigos y lograr que ambos funcionasen a la vez pudiendo dotar al robot de ambos controles. Esto sería mucho más atrayente ya que hace al robot mucho más atractivo y divertido el poder controlarlo mediante un dispositivo.

7 CONCLUSIONES

Los objetivos de nuestro proyecto eran conseguir un robot balancín estable y capaz de mantenerse en auto equilibrio de forma dinámica y además de ello añadirle diferentes funcionalidades extra mediante el uso de *encoders*.

Para poder lograr esto se han dedicado largas horas a diferentes aspectos del proyecto, sobre todo de la primera fase con la cual quizás no hayamos quedado totalmente satisfechos. Esto se debe a que a pesar de que el robot se mantenga en equilibrio por más de 10 minutos (quizás mucho mas), este sigue avanzando muy poco a poco. Como hemos explicado en el apartado de problemas encontrados pasamos mucho tiempo ajustando valores PID y por diversos errores nuestros esfuerzos no dieron los frutos deseados. Por ello, al final tuvimos que aceptar un resultado satisfactorio sin ser realmente el óptimo para poder seguir avanzando con nuestro proyecto en individual.

En la parte de los *encoder* podemos decir que se han logrado añadir funcionalidades extra, como el de conseguir la distancia recorrida o velocidad a la que anda el robot. A pesar de ello no he sido capaz de que el robot vuelva al sitio de partida de forma autónoma lo cual podría mejorarse con un poco más de tiempo y esfuerzo.

Hemos tenido que superar grandes dificultades (y no han sido pocas además) y superarlas nos ha ayudado a aprender a mantener la calma, puesto que sin superarlas no podíamos hacer nada del proyecto y había momentos en que me/nos encontrábamos en una situación de impotencia.

Teniendo en cuenta todo lo anterior, puedo decir que aunque en algunos hayan sido muy desesperantes, en general, la experiencia ha sido muy satisfactoria. La gran mayoría de los objetivos han sido cumplidos y el poder trabajar en un tipo de proyecto que realmente te gusta y te motiva es un gran punto a favor, ya que todo ese mundo de la robótica y sistemas embebidos me atrae mucho.

8 APÉNDICE

En este apartado se profundizara en diferentes temas de los que hemos hablado en el proyecto, pero no hemos profundizado debido a que son temas que se han dado en clase o no veíamos convenientes introducirlos en la memoria.

8.1 PWM

El funcionamiento es simple, el ciclo de trabajo describe la cantidad de tiempo que la señal está en un estado lógico alto, se expresa como el porcentaje del tiempo total que esta para completar un ciclo completo. La frecuencia determina lo rápido que se completa un ciclo (por ejemplo: 1000 Hz corresponde a 1000 ciclos en un segundo). Al cambiar una señal del estado alto a bajo a una tasa lo suficientemente rápida y con un cierto ciclo de trabajo, la salida parecerá comportarse como una señal analógica constante cuando esta se aplica a algún dispositivo.

Ejemplo: Para crear una señal de 3V dada una fuente digital que puede ser alta (5V) o baja (0V), se puede utilizar un PWM con un ciclo de trabajo del 60%. El cual generaría una señal de 5V el 60% del tiempo. Si la señal es conmutada lo suficientemente rápido, el voltaje visto en las terminales del dispositivo parecerá ser el valor promedio de la señal. Si el estado lógico bajo es 0V (que es el caso más común) entonces el voltaje promedio puede ser calculado multiplicando el voltaje que represente el estado lógico alto por el ciclo de trabajo, o 5V x 0.6 = 3V. Seleccionar un ciclo de trabajo del 80% sería equivalente a 4V.

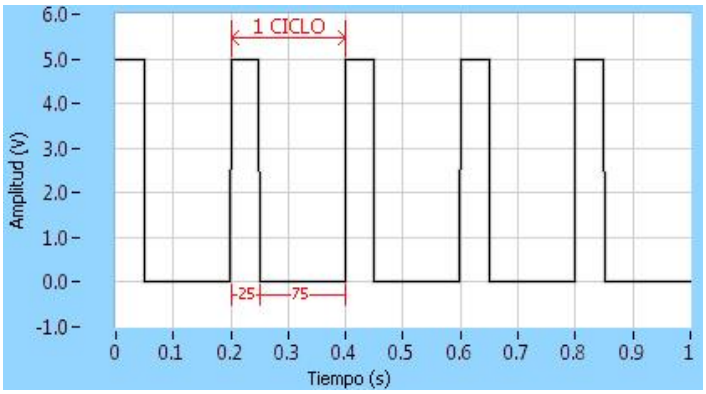


Figura 26: Ciclo de trabajo del 25%.

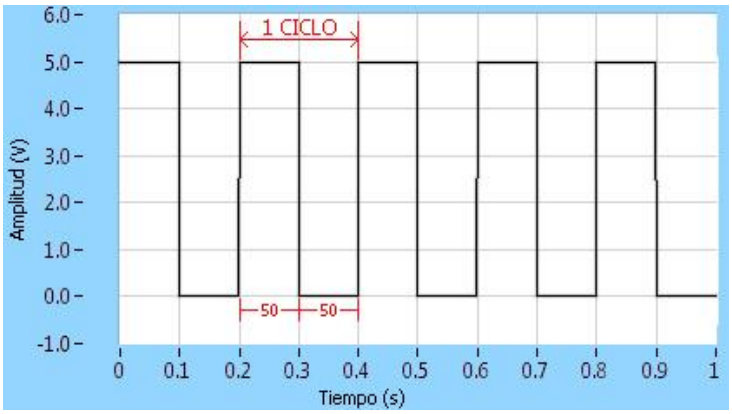


Figura 27: Ciclo de trabajo del 50%.

8.2 I^2C

La principal característica del I^2C es que tan solo usa dos líneas para transmitir la información: una para los datos y otra para la señal del reloj. También hace falta una tercera línea, pero esta sólo es la referencia (masa). Estas tres líneas suelen usar las siguientes nomenclaturas:

- SCL (SystemClock) es la línea de los pulsos de reloj que sincronizan el sistema.
- SDA (System Data) es la línea por la que se mueven los datos entre los dispositivos.
- GND (Masa) común de la interconexión entre todos los dispositivos "enganchados" al bus.

Los dispositivos conectados al bus I^2C tienen una dirección única para cada uno.

Para que la comunicación tenga éxito debe seguir un protocolo de comunicación. En una conexión entre varios dispositivos a través del I^2C existen dos tipos de dispositivos: los dispositivos maestros y los dispositivos esclavos. El maestro es el Dispositivo que determina los tiempos y la dirección del tráfico en el bus. Es el único que aplica los pulsos de reloj en la línea SCL. Cuando se conectan varios dispositivos maestros a un mismo bus la configuración obtenida se denomina "multi-maestro". En cambio, el dispositivo esclavo no tiene la capacidad de generar pulsos de reloj. Los dispositivos esclavos reciben señales de comando y de reloj generados desde el maestro.

Para que el maestro comience una comunicación el bus tiene que estar libre, esto quiere decir que tanto la línea SDA como la SCL tienen que estar inactivas, es decir, presentar un estado lógico alto. Si no se da el caso de que el bus está libre el maestro no puede comenzar comunicación alguna con los dispositivos conectados mediante ese bus. Cuando el bus está libre, cualquier dispositivo maestro puede ocuparlo, estableciendo la condición de inicio, en la cual la línea de datos, SDA, toma un estado bajo mientras que la línea de reloj, SCL, permanece alta.

El primer byte que se transmite luego de la condición de inicio contiene siete bits que componen la dirección del dispositivo que se desea seleccionar, y un octavo bit que corresponde a la operación que se quiere realizar con él (lectura o escritura), aunque en nuestro caso siempre pretenderemos realizar la escritura (la operación de escritura se indica con el bit a nivel lógico bajo). Cada dispositivo tiene una dirección única, en nuestro caso la IMU tiene la dirección: 0xA6 en caso del acelerómetro y 0xD0 en caso del giroscopio, las cuales hemos averiguado gracias al proyecto que Omar Luque Rodríguez presento sobre los cuadrucopteros en donde uso estos mismos dispositivos, pero aun así podríamos haberlos descubierto leyendo la ficha técnica del dispositivo IMU.

Tras enviar la condición *start* y los 8 bits si el dispositivo cuya dirección corresponde a la que se indica en los siete bits está presente en el bus, éste contesta con un bit en bajo, ubicado inmediatamente luego del octavo bit que ha enviado el dispositivo maestro. Este bit de reconocimiento (ACK) en bajo le indica al dispositivo maestro que el esclavo reconoce la solicitud y está en condiciones de comunicarse. Aquí la comunicación se establece en firme y comienza el intercambio de información entre los dispositivos.

Puesto que indicamos que se desea realizar la operación de escritura el dispositivo maestro envía datos al dispositivo esclavo. Esto se mantiene mientras continúe recibiendo señales de reconocimiento, y el contacto concluye cuando se hayan transmitido todos los datos. El dispositivo maestro puede dejar libre el bus generando una condición de parada (stop en inglés).

La secuencia completa de comunicación seria la siguiente:

1. Enviar una secuencia de inicio
2. Enviar la dirección de dispositivo con el bit de lectura/escritura en bajo
3. Enviar el número de registro interno en el que se desea escribir
4. Enviar el byte de dato
5. [Opcionalmente, enviar más bytes de dato]
6. Enviar la secuencia de parada

En este código se puede apreciar que se pasan como parámetros *"ControlByte"* y *"Address"*. El primero corresponde a los 8 bits que se mandan donde se indica la dirección del dispositivo y el bit con lectura o escritura. En cuando al *"Address"* indica la dirección del registro en la que se desea escribir la información para que luego podamos coger de ahí los bits que nos interesan o nos aportan la información para saber el estado del robot.

8.3 FILTRO DE KALMAN

El objetivo de este filtro será la obtención de un estimador óptimo de un sistema dinámico, basado en observaciones ruidosas y en un modelo de la incertidumbre de la dinámica del sistema. Para ello haremos uso de dos ingredientes fundamentales: los conceptos de modelo del sistema y modelo de medida o de observación.

Para ir entendiendo la formulación vamos a considerar paralelamente la estimación de una constante (por ejemplo un voltaje) cuya medida lleva asociado un ruido con una desviación típica de 0.1 voltios. El voltaje será una constante y tenemos observaciones secuenciales ruidosas.

8.3.1 Modelo del sistema

El sistema físico se modeliza por un vector de estados x, llamado simplemente el estado, y un conjunto de ecuaciones llamado el modelo del sistema. El modelo del sistema es una ecuación de vectores que describe la evolución del estado con el tiempo. El tiempo de observación tiene la forma $t_k=t_0+k\Delta T$, k=0,1..., ΔT es el intervalo de muestreo y x_k el estado x (t_k). Vamos a suponer que ΔT es pequeño y que por tanto podemos utilizar un modelo del sistema lineal, es decir,

$$x_k = \varphi_{k-1} x_{k-1} + \xi_{k-1}$$

donde ξ_{k-1} es un vector aleatorio que modeliza el ruido aditivo. El subíndice k-1 en φ indica que la matriz de transición φ es (puede ser) una función del tiempo. Volvamos ahora a nuestro ejemplo. Como el voltaje es una constante podemos utilizar como modelo del sistema

$$x_k = x_{k-1} + \xi_{k-1}$$

Observemos que cuanto mayor sea k, en principio, más fiabilidad tendrá la estimación. Es importante analizar las características de ξ_{k-1} pero esto lo haremos con posterioridad. Casi siempre el ruido ξ_{k-1} se supone normal de media cero.

8.3.2 Modelo de medida

El segundo ingrediente en la teoría de la estimación es el modelo de medida. Suponemos que en cada instante t_k tenemos una observación ruidosa del vector de estados o al menos alguna de sus componentes mediante la siguiente relación

$$z_k = \mathrm{H}_k x_k + \mu_k$$

z_k es el vector de medidas tomadas en el instante t_k. H k es la llamada matriz de medidas y μ_k es un vector aleatorio que modeliza la incertidumbre asociada a las medidas. En nuestro caso tendríamos μ_k normales independientes de media 0 y desviación 0.1.

8.3.3 Formulación de la teoría de estimación lineal óptima del filtro de Kalman.

Esta sección es una recopilación de las dos anteriores.

El estado de un sistema dinámico en el instante t_k está descrito por un vector n-dimensional x k llamado vector de estados. La evolución del sistema se modeliza mediante:

$$z_k = \varphi_{k-1} \mathrm{x}_{k-1} + \xi_{k-1}$$

Donde φ_{k-1} es una matriz de tamaño *nxn* llamada matriz de transición de estados y ξ_k es un *n*-vector que modeliza el ruido asociado al modelo del sistema.

En cualquier instante t_k se obtiene un vector m-dimensional de medidas z_k. La relación entre el estado y las medidas es lineal y se modeliza mediante

$$z_k = \mathrm{H}_k x_k + \mu_k$$

H_k es una matriz dependiente del tiempo de tamaño *mxn* y μ_k es un m-vector aleatorio que modeliza la incertidumbre asociada a las medidas.

Los términos ξ_k y μ_k son vectores aleatorios gaussianos blancos de media cero y matrices de covarianza Q_k y R_k respectivamente.

Para el problema del ángulo R_k=0.5 y la varianza del modelo de estados la fijamos nosotros. En nuestro caso hemos supuesto Q_k=0.001 aunque también hemos realizado pruebas con Q_k=0.003, pero vimos mejor resultado con el primero.

Supongamos que tenemos x_{k-1} y P_{k-1}. En el primer instante hemos de tener x_0 y P_0. Los valores iniciales x_0 y P_0 los introducimos a mano. ¿Cómo calculamos los nuevos estimadores iterativamente?

Tenemos un estimador x_{k-1} y su matriz de covarianzas P_{k-1}. Primero calculamos, antes de que llegue la observación z_k

$$P_k{}' = \phi_{k-1}\, \mathrm{P}_{k-1}\phi_{k-1}{}^{t} + \varrho_{k-1}$$

Después la ganancia

$$K_k = \mathrm{P}_k'\mathrm{H}_k{}^{t}\left(\mathrm{H}_k{}^{t}P_k{}'^{H_k} + R_k\right)^{-1}$$

A continuación el estimador del estado k óptimo (aquí metemos la observación)

$$x_k = \phi_{k-1}x_{k-1} + P_k{}'\mathrm{H}_k{}^{t} * \left(\mathrm{H}_k{}^{t} * P_k{}' * H_k + R_k\right)^{-1} * \\ * \left(z_k - \mathrm{H}_k\phi_{k-1}x_{k-1}\right)$$

Y por último la matriz de covarianzas de este estimador

$$P_k = P_k' - P_k{}'\mathrm{H}_k{}^{t} * \left(\mathrm{H}_k{}^{t} * P_k' * H_k + R_k\right)^{-1}\mathrm{H}_k P_k{}' =$$

$$P_k' - K_k H_k P_k' = (I - K_k H_k)P_k' =$$

$$(I - K_k)P_k'(I - K_k)^{t} - K_k R_k K_k{}^{t}$$

9 REFERENCIAS BIBLIOGRÁFICAS

En este apartado se mostraran algunos de los enlaces que se han obtenido para informarnos sobre el proyecto y su funcionamiento:

Relacionados con el Robot Balancín y/o proyectos similares:

http://www.elecrow.com/freescale-mc9s12xs128mal-2wheel-selfbalancing-robot-p-878.html

http://www.elecrow.com/freescale-mc9s12xs128mal-2wheel-selfbalancing-robot-p-878.html

https://github.com/fasaxc/ArduRoller

http://www.bajdi.com/building-a-self-balancing-bot/

http://www.instructables.com/id/Self-Balancing-Segway-Instructabot/

https://github.com/jjulio/b-robot

http://www.instructables.com/id/2-Wheel-Self-Balancing-Robot-by-using-Arduino-and-/

http://www.balanduino.net/

Segway:

http://www.segway.es/sostenibilidad/sostenibilidad.pdf

http://touristear.com/en-segway-por-madrid-te-animas/

https://en.wikipedia.org/wiki/Segway_PT

https://es.wikipedia.org/wiki/Segway

Controlador de motor:

http://www.robotpower.com/products/simple-h_info.html

I^2C:

http://robots-argentina.com.ar/Comunicacion_busI2C.htm

https://es.wikipedia.org/wiki/I%C2%B2C

Encoders y motores:

http://www.sainsmart.com/29-1-metal-gearmotor-37dx52l-mm-with-64-cpr-encoder-12v-365rpm.html

https://www.pololu.com/product/1443

http://bibing.us.es/proyectos/abreproy/11452/fichero/PFC.pdf

https://en.wikipedia.org/wiki/Least_significant_bit

IMU:

https://www.sparkfun.com/products/10724

https://es.wikipedia.org/wiki/Aceler%C3%B3metro

http://www.alldatasheet.com/view.jsp?Searchword=Adxl345%20datasheet&gclid=COXq9rnLoMcCFUbkwgod5JMD-w

Filtro de Kalman:

https://es.wikipedia.org/wiki/Filtro_de_Kalman

http://robotsforroboticists.com/kalman-filtering/

http://forum.arduino.cc/index.php?topic=58048.0

Control PID:

https://es.wikipedia.org/wiki/Controlador_PID

http://www.lra.unileon.es/es/book/export/html/268