E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

# Diseño y fabricación de una tarjeta PCB ("Shield") para el control de un robot autobalanceado basado en Arduino

Grado en Ingeniería
en Tecnologías Industriales

Trabajo Fin de Grado

Luis Fernando Basarte Bozal

Javier Goicoechea Fernández

Pamplona, Junio de 2014

# Índice

# 1. Introducción y objetivos.

## 1.1. Introducción.

Este proyecto comprende el desarrollo e implementación de un robot autobalanceado basado en la plataforma Arduino. Se utilizará una placa Arduino y se diseñará y fabricará con un shield o tarjeta (PCB), donde se incluirán los elementos hardware que se consideren necesarios.

Abarca el estudio y montaje del chasis y los sistemas de sensado, control digital, alimentación y motores.

Un robot autobalanceado es un dispositivo que, aún teniendo su centro de masas por encima del eje de giro, consigue mantener el equilibrio. Se basa o aproxima al problema del péndulo invertido, uno de los temas clave, que se verá más adelante.

Para mantener el equilibrio, el robot utilizará sensores que le informen acerca de su posición y orientación con respecto al suelo. Estos datos serán gestionados por un microcontrolador, integrado en nuestro caso en una placa Arduino. Tras el procesado de la información, también se encargará de generar la señal de control que, a través de un dispositivo de potencia, transmita a los motores la consigna de actuación adecuada.

Como reto cabría destacar que es un proyecto que debe conjugar diversas materias o disciplinas propias de la ingeniería como:

- Mecánica, ya que deberá analizarse el sistema que conforma el robot desde este punto de vista, considerando la estructura o la disposición de las masas entre otros factores.
- Instrumentación: será necesaria la obtención de información que nos indique el estado de nuestro robot con respecto a un sistema de referencia.
- Microcontroladores y programación: incluye la elección y programación de los dispositivos que tratarán la información obtenida por los sensores, y que la transformaran en señales de control.

- Control: algoritmo que será incluido en la programación del microcontrolador y que se encargara de analizar la información y elaborar la respuesta apropiada.
- Potencia: comprende la alimentación de los circuitos y la transmisión de la señal de control a los actuadores.

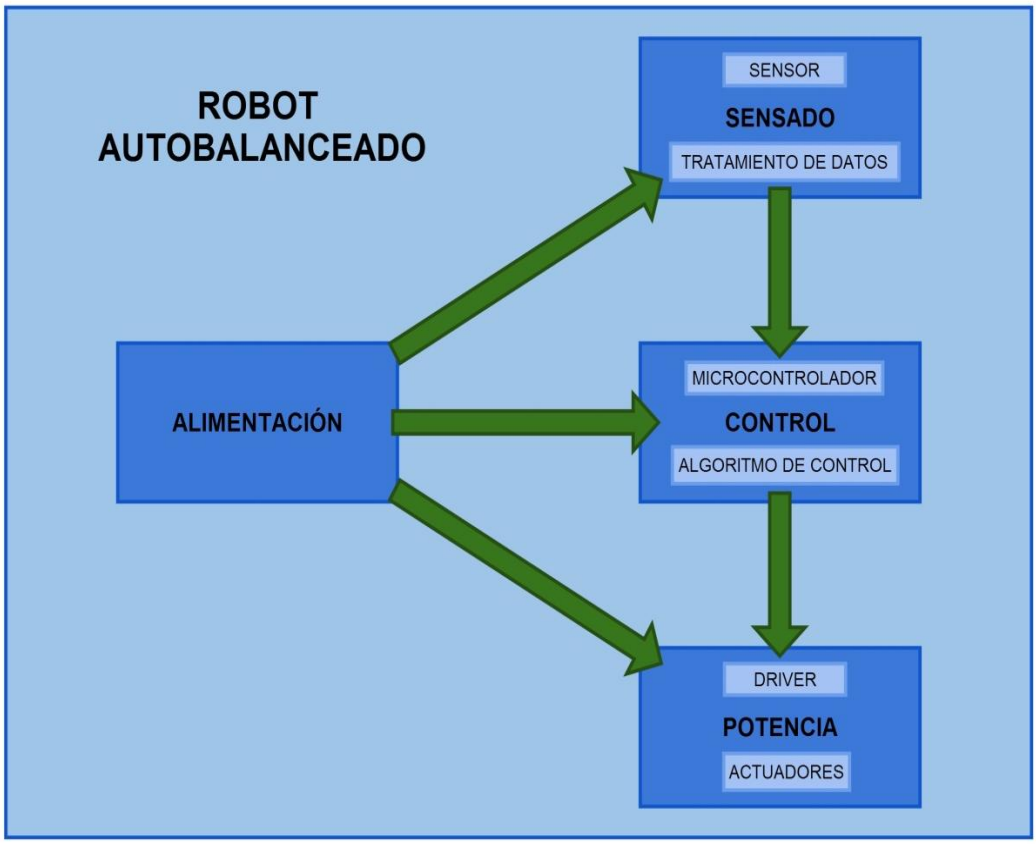



Figura 1. Diagrama del sistema.

## 1.2. Objetivos.

La meta principal del proyecto es conseguir el equilibrio de un robot autobalanceado. Para alcanzar tal fin, deberán conseguirse otros objetivos que se complementen y en conjunto se logre el fin último. Serían los siguientes:

- Análisis de necesidades.
- Elección adecuada de los componentes.
- Obtención de datos a través de sensores.
- Tratamiento de la información obtenida.
- Implementación de un algoritmo de control.
- Programación del microcontrolador.
- Transmisión del control y la potencia.
- Realización de ensayos.
- Generación de conclusiones.

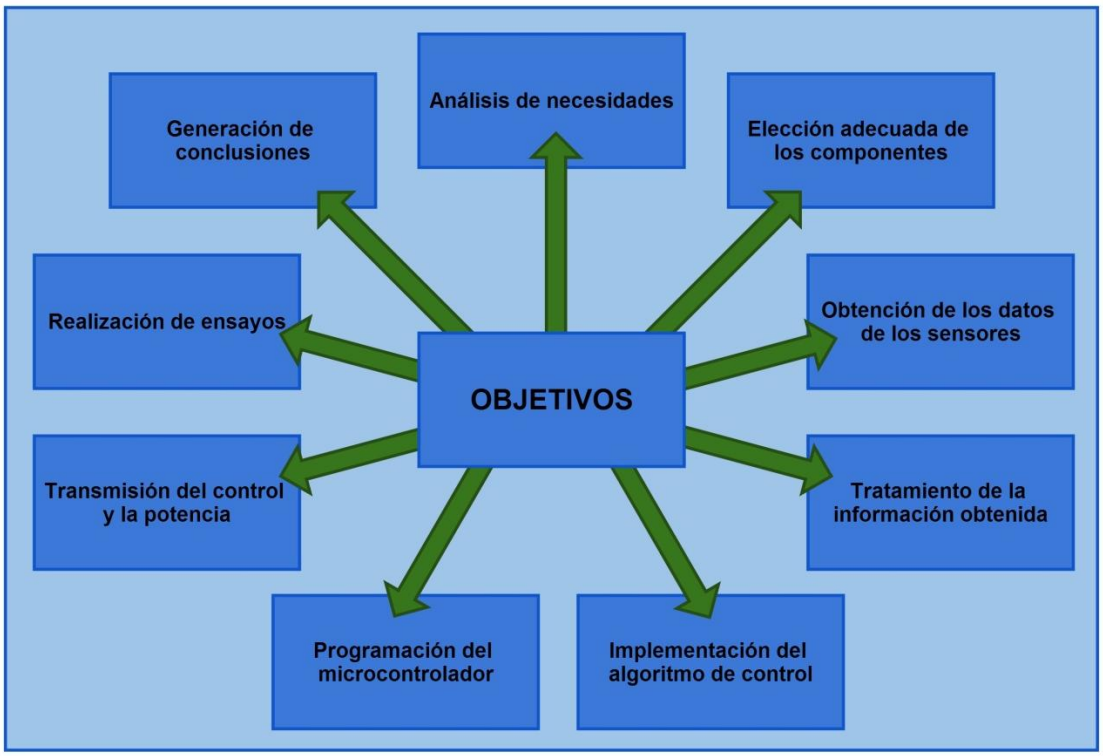



Figura 2. Diagrama de objetivos.

# 2. Análisis del robot autobalanceado.

Este aparatado muestra el análisis de las necesidades que deberemos cubrir o resolver a la hora de avanzar en la realización del robot.

En primer lugar abordaremos el estudio de la parte mecánica, asemejando nuestra estructura y disposición de masas al concepto de péndulo invertido. Posteriormente, se buscará la manera de obtener la información de interés mediante la selección de los sensores adecuados.

Escogeremos el microcontrolador que constituirá el sistema de captación y canalización de dicha información, y que será además el encargado de generar las respuestas adecuadas para lograr el equilibrio.

Se estudiará también la manera de intervenir sobre nuestro sistema seleccionando los actuadores pertinentes y la manera de transmitir la potencia.

Finalmente se realizará la programación, el ensayo con el sistema completo, y la corrección de los fallos o deficiencias que pudieran surgir.

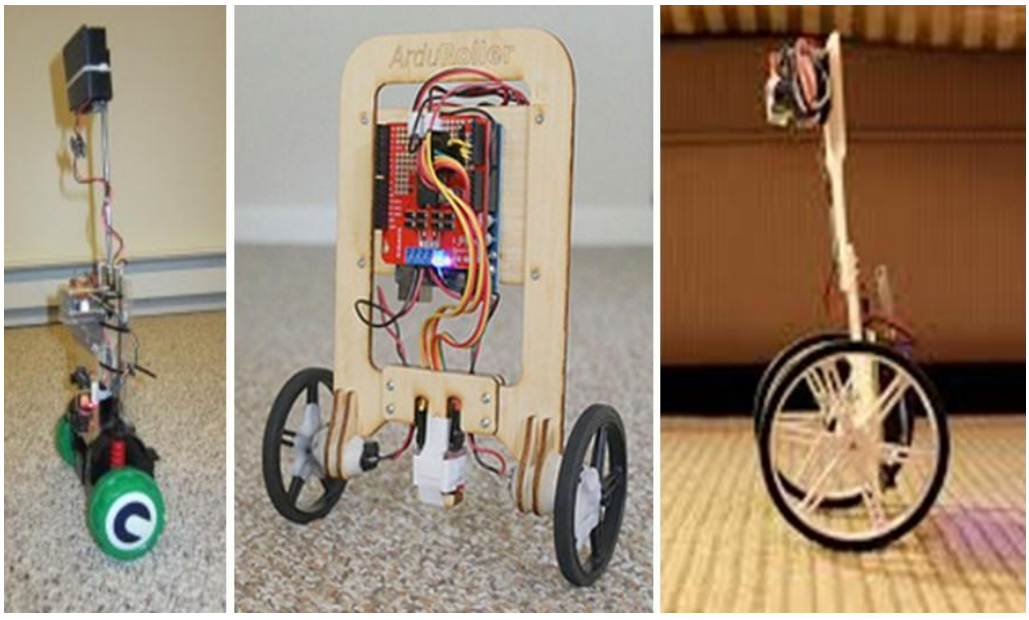


Figura 3. a)BalancingRobot1[1]. b) Arduroller. [2] c) Vertibot[3].
(Fuente: Vertibot: http://www.unocero.com/2013/05/14/vertibot
Arduroller: http://sugru.com/blog/the-arduroller-a-self-balancing-robot.
BalancingRobot1:http://www.kerrywong.com/2012/03/21/a-self-balancing-robot-iii/).

## 2.1. El péndulo invertido.

Como se menciona en la introducción, se comenzará analizando el problema físico del equilibrio de un péndulo invertido, que consiste en un péndulo cuyo centro de masas se encuentra situado por encima del punto o eje de balanceo. Esta disposición dota al sistema de una inestabilidad estática, que puede ser compensada aplicando un momento o par de características apropiadas en el eje de giro, o mediante un movimiento en el plano horizontal [4].

El fundamento de este proyecto consiste de forma simplificada en tratar de conseguir y controlar dicho equilibrio, basándose en el ángulo formado con la vertical, también deberá ser tenida en cuenta, aunque en menor medida, las aceleraciones que actúen en el robot.

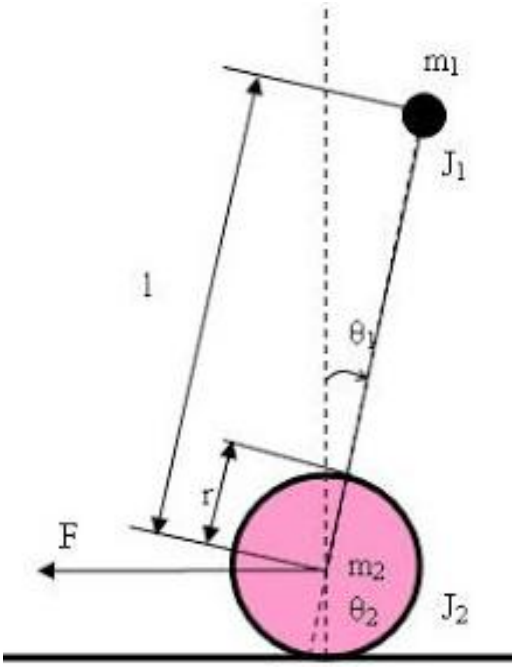



Figura 4. Péndulo invertido. (Fuente: http://nxttwowheels.blogspot.com.es/).

El análisis de estos sistemas constituye uno de los clásicos problemas abordados en la teoría de control y en la dinámica de sistemas, es por ello tratado en diversos textos. También se suele utilizar este problema como banco de pruebas para ensayos con diferentes tipos de controles como pueden ser PID (Proporcional Integral Derivativo), representación de espacio de estados, o control difuso entre otros [5] [6] [7].

Considerando el problema como un péndulo invertido y aproximándolo mediante consideraciones como punto de balanceo fijo, y sistema de movimiento en 2 dimensiones, obtenemos como ecuación principal del comportamiento de nuestro sistema, una función en la que la aceleración ($\ddot{\theta}$) es dependiente de la gravedad (g), la longitud del péndulo (l) y el seno del ángulo formado (θ) [8].

$$\ddot{\theta} = \frac{g}{\ell} \sin\theta$$

Figura 5. Dinámica péndulo invertido. (Fuente: Wikipedia).

En conclusión, se deduce que deberemos centrarnos en el estudio del ángulo, $\theta_1$ en la imagen, y en su control mediante los métodos descritos de actuación con par y movimiento en el eje horizontal.

## 2.2. Sensor.

### 2.2.1. Tipos de sensores.

La existencia de múltiples sensores empleados en la medición de ángulos o inclinación llevo al análisis de varias opciones [9], algunas de las cuales son:

- Potenciómetros rotativos: se trata de una resistencia variable cuya salida de tensión varía en función del ángulo [10]. Esta opción fue rechazada ya que se buscaba un sistema totalmente autónomo y móvil, sin ningún tipo de conexión con el suelo o que debiera incluir algún tipo de modificación.

Figura 6. Potenciómetro rotativo.(Fuente: Wikipedia).

- RVDT ("Rotary Variable Differential Transformer"): tipo de transformador eléctrico empleado en la medición de ángulos. Su funcionamiento se basa en

un devanado primario que induce mayor o menor voltaje en un bobinado secundario en función del ángulo formado [11]. Fue rechazado al igual que el anterior porque su inclusión conllevaba la modificación de la estructura y ligarlo a alguna referencia, eliminando por completo la autonomía.

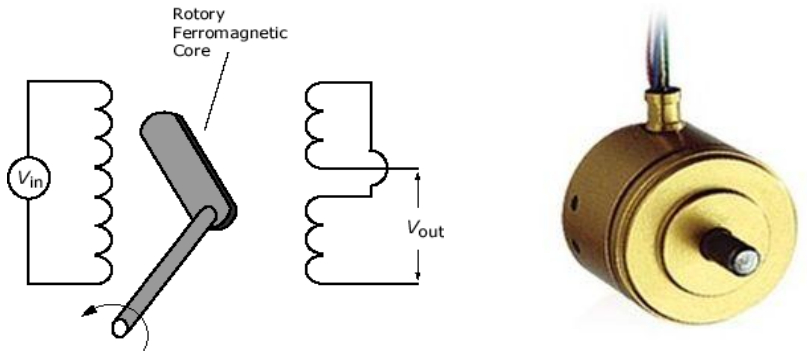



Figura 7. RVDT. (Fuente: http://www.efunda.com/designstandards/sensors/lvdt/rvdt_intro.cfm http://www.directindustry.es/prod/meggitt-sensing-systems-measurement-group/transductores-desplazamiento-rvdt-5413-447606.html).

- Encoders: Este tipo de sensores proporciona una lectura digital del ángulo. Físicamente, están formados por un dispositivo optoelectrónico, o electromecánico que convierte la posición angular o ángulo en un determinado código [12]. Este código puede ser tan simple como un contador de pulsos sencillos, pulsos en cuadratura, o datos binarios en código de Jonhnson. Como principal contra presentan que la resolución es más limitada que en el resto de sensores considerados. Además adolecen de la misma desventaja que los sensores anteriores, requieren de una parte rotatoria y una estática, y al querer que el sistema sea completamente móvil no resultan apropiados.

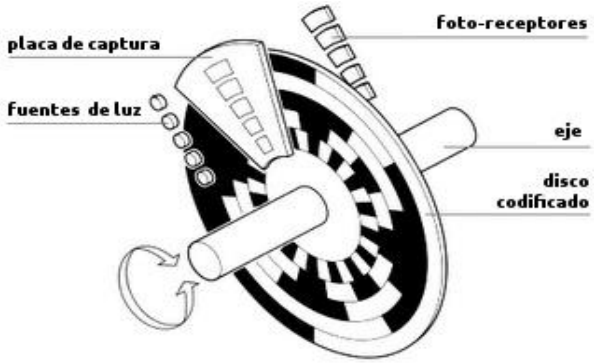



Figura 8. Encoder. (Fuente: http://www.lbaindustrial.com.mx/que-es-un-encoder/).

- Acelerómetro: Este tipo de sensor mide aceleraciones, y puede utilizarse para la medición de ángulos a partir de las señales que genera en ellos la gravedad [13]. Existen varios tipos de acelerómetros. Los más empleados para la medición de vibraciones son los piezoeléctricos y los ópticos. Sin embargo para aplicaciones más sencillas, con un ancho de banda más limitado, los acelerómetros MEMS ("Microelectromechanical Systems") están siendo cada vez más empleados ya que son muy competitivos en cuanto a coste. Como factores favorables cabe destacar que son sensores rápidos y estables, además de que no presentan derivas. Por otro lado, tienen como inconveniente que la señal obtenida es altamente ruidosa.

Figura 9. Acelerómetro piezoeléctrico de cuarzo. (Fuente: Wikipedia).

- Giróscopo: Estos sensores proporcionan información acerca de la velocidad angular de un móvil respecto de un sistema de referencia estático.

  La aproximación clásica a los giroscopios consiste en un disco con una gran inercia que rota en torno a un eje. Debido a la ley de conservación del momento angular, este móvil rotativo mantiene su orientación respecto del sistema de referencia en el que fue impulsado, aunque el móvil dentro del cual se aloja cambie de orientación. Estos dispositivos han sido ampliamente utilizados en la tecnología aeronáutica, donde es vital la estimación de la orientación de las aeronaves respecto del suelo [14].

En los últimos años esta aproximación clásica ha sido sustituida por otras más modernas y precisas. Por ejemplo en aeronáutica los giroscopios ópticos basados en interferómetros se tipo Sagnac. Estos dispositivos proporcionan una señal óptica, que es convertible a electrónica, siendo además proporcional a la velocidad angular, con lo cual puede obtenerse el ángulo integrando el valor de dicha velocidad.

Hoy en día, para aplicaciones menos exigentes, existen también otras tecnologías para fabricar giroscopios, como por ejemplo los dispositivos MEMS de los que se hablará más adelante en el trabajo.

En general los giróscopos son sensores poco sensibles a giros lentos, pero de los que se obtiene una señal más limpia. Una gran desventaja es que debido a la necesidad de integrar la señal del giroscopio, pueden obtenerse derivas en las mediciones.

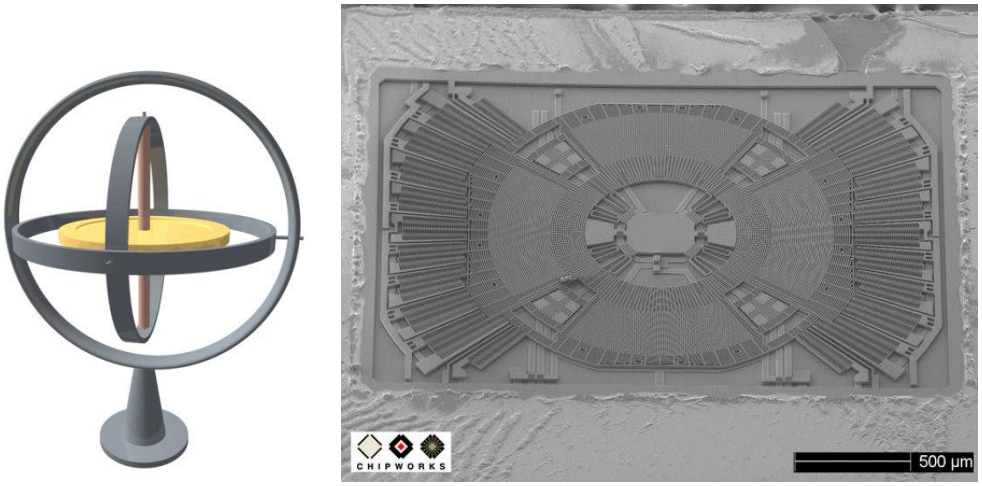



Figura 10. Izda. Giróscopio clásico. Dcha. Gisroscopio MEMS. (Fuente: Wikipedia http://elcoyotequesuelda.blogspot.com.es/2010/08/microfotografias-de-giroscopios-mems.html ).

De todas las propuestas planteadas, fue seleccionada una solución conjunta formada por las 2 últimas opciones, ya que resultaba la forma menos invasiva en cuanto a montaje y acondicionamiento de la estructura. Además, el tamaño de estos sensores es muchísimo menor y pueden encontrarse ambos implementados en un mismo encapsulado en los dispositivos electrónicos conocidos como IMU ("Inertial Measurement Unit").

### 2.2.2. IMU.

La Unidad de Medición Inercial o IMU es un dispositivo electrónico que mide y registra información obtenida acerca de la velocidad, la orientación y los efectos de las fuerzas gravitatorias. Utiliza una combinación de acelerómetro y giróscopos para obtener sistemas de 6 grados de libertad, que pueden ser ampliados hasta los 9 incluyendo magnetómetros [15] [16].

Para los requerimientos de nuestro robot será suficiente con emplear el sistema de 6 grados de libertad que incluye 3 acelerómetros dispuestos de forma ortogonal y 3 giróscopos dispuestos también de forma ortogonal. La implementación de los sensores en este formato IMU difiere de su forma habitual y se trata a continuación.

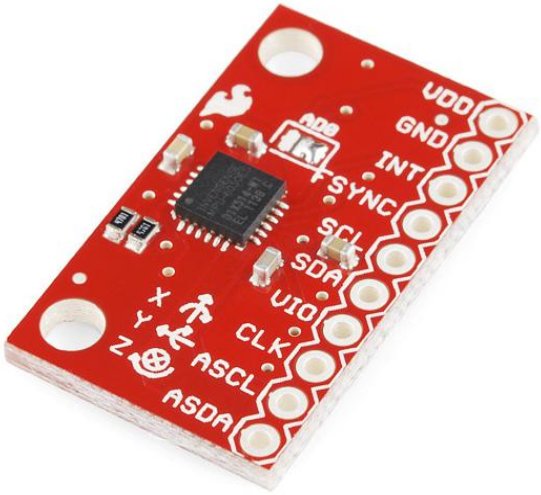



Figura 11. IMU MPU6050 Sparkfun.(Fuente: https://www.sparkfun.com/products/11028).

### 2.2.3. Acelerómetro.

Los acelerómetros MEMS (“Microelectromechanical Systems”) son de tamaño reducido y pueden estar incluidos en las IMU anteriormente descritas. La principal ventaja de estos dispositivos MEMS es que pueden ser creados mediante técnicas de fabricación microelectrónica en un chip de silicio, a la vez que toda la electrónica necesaria para el sistema de acondicionamiento, adquisición y comunicación a un precio y un tamaño muy reducidos [17].

La estructura de estos dispositivos puede apreciarse en la imagen, contienen por lo general placas capacitivas internas, algunas fijas y otras móviles. Las fuerzas de aceleración que actúan sobre el sensor variarán la disposición de unas con respecto a las otras modificando la capacitancia existente entre ambas. Estos cambios serán traducidos posteriormente en señales que podremos utilizar.

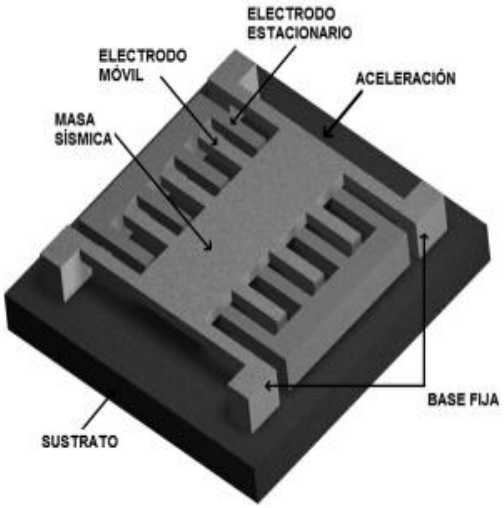



Figura 12. Acelerómetro MEMS. (Fuente: http://5hertz.com/tutoriales/?p=228).

### 2.2.4. Giróscopo.

Los sensores giroscópicos incluidos en el IMU son también MEMS, su funcionamiento está basado en una pequeña masa que varía su posición al variar la velocidad angular, el dispositivo convierte estos cambios en una señal medible. Gracias a este diseño, ha sido posible reducir de forma notable el tamaño de estos dispositivos y poder así incluirlos en un circuito integrado [18].

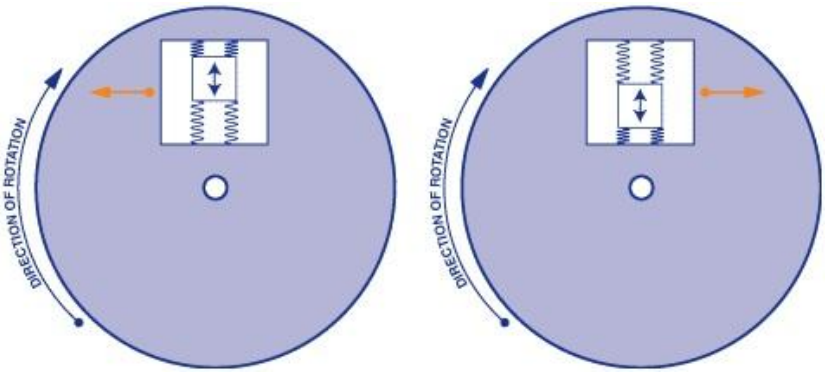



Figura 13. Giróscopo MEMS. (Fuente: http://5hertz.com/tutoriales/?p=431).

### 2.2.5. Conclusiones.

La solución que se dispuso para el sensor fue el uso de un IMU de 6 grados de libertad (en adelante g.d.l). Se estudiaron como posibilidades el MPU6050 y el ITG3200/ADXL345, ambos comunicados por el protocolo de comunicaciones digitales $I^2C$ (Inter-Integrated Circuit) y de precios similares. Finalmente se adoptó el MPU6050 que fue elegido por su capacidad de ser ampliado a 9 g.d.l. y por incluir un procesador propio que pudiera descargar de trabajo al procesador principal en caso de ser necesario.

## 2.3. Microcontrolador.

### 2.3.1. Análisis y conclusiones.

Un microcontrolador es un circuito integrado que aúna varios bloques funcionales. Los principales órganos que forman el microcontrolador son la Unidad Central de Procesos o CPU, la memoria, y los periféricos. Se trata además, de un dispositivo de bajo costo, versátil, programable, que puede ejecutar funciones y dotar de inteligencia a sistemas más complejos [19] [20].

El siguiente paso fue la selección del microcontrolador, se partía desde el conocimiento de 2 plataformas: PIC de Microchip y AVR de Atmel, implementado este último en las placas de desarrollo Arduino (ATmega).

En primer lugar se analizó el aspecto hardware, en el que ambas familias comparten las características más determinantes, algunas de las cuales son:

- Microprocesador tipo RISC, con un juego de instrucciones reducidas.
- Arquitectura Harvard, buses separados para las instrucciones y los datos.
- Existencia de modelos con núcleos de 8/16/32 bits.
- Memoria Flash, ROM y EPROM.
- Puertos de entrada/salida (E/S).
- Temporizadores
- Conversores analógico digital (CAD)
- Módulos CCP (Captura Compara y PWM (Pulse Width Modulation))
- Puertos de comunicación serie UART(“Universal Asynchronous Receiver Transmitter”), I2C (“Inter Integrated Circuit”) e SPI (“Serial Peripheral Interface”).

Posteriormente se comparó la parte no física, que incluye aspectos como el software y la programación, la cantidad de información y su accesibilidad, existencia de foros o los precios de los componentes entre otros. Al igual que en el apartado anterior compartían la mayoría de los aspectos, con la posibilidad de ser programados en C, multitud de información en páginas y foros de internet, siendo los precios algo inferiores en el caso de los ATmega. Ante todas estas similitudes, el principal elemento que desequilibró la balanza fue Arduino y su plataforma de desarrollo [21] [22].

### 2.3.2. Arduino.

Arduino es una plataforma de prototipado en código abierto. Fue creado para el desarrollo de proyectos electrónicos de bajo costo, como alternativa a las costosas placas de desarrollo existentes hasta su aparición.


Figura 14. Arduino Logo. (Fuente: http://arduino.cc/en).

Constituye una herramienta útil en aplicaciones de:

- Sensado, gracias a sus entradas digitales y analógicas.
- Comunicación, pudiendo incluir I2C, TWI ("Two Wire Interface"), SPI o UART.
- Control de dispositivos, mediante las salidas digitales o de PWM ("Pulse Width Modulation")

Existe una amplia gama de placas disponibles, la mayoría cuentan con microcontroladores Atmel, como en el modelo Uno, Mega o Nano que incluyen el ATmega328 o el ATmega2560, otro podría ser el SAM3X8E ARM Cortex-M3, incluida en el Due, aunque poco a poco se van introduciendo otros de casas como Intel, con el modelo Galileo.

La programación del microcontrolador presente en estos dispositivos se hace mediante un lenguaje propio, Arduino, sencillo e intuitivo favorece un rápido aprendizaje. Este está basado en el lenguaje de alto nivel Processing y en C, por lo que soporta muchas de sus funciones.

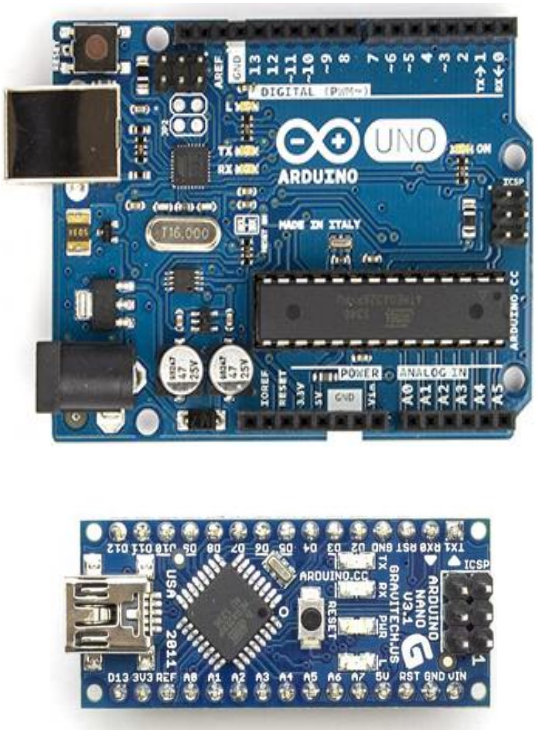



Figura 15. Arriba Arduino Uno R3. Abajo Arduino Nano. (Fuente: http://arduino.cc/en).

Otras ventajas que proporciona son:

- Código abierto: permite que el software sea desarrollado y distribuido libremente.
- Bajo costo: ya que nació con esta idea como fundamento, hace que la electrónica sea asequible a un mayor número de personas, lo que favorece su expansión.
- Conjugado con el punto anterior, se obtiene una amplia comunidad que participa en el desarrollo de nuevos programas y de nuevo hardware compatible, esto favorece un *feedback* o realimentación continua, ya que cada vez la cantidad de información y de dispositivos es mayor, adaptándose mejor a las necesidades de potenciales nuevos usuarios, y cada uno de ellos aportando nuevamente información o desarrollo de dispositivos. Esto es

fácilmente visible navegando por internet, dónde se encuentran gran número de páginas y blogs relacionadas con Arduino.

- Alta compatibilidad: puede adaptarse para ser usado en proyectos electrónicos de casi cualquier tipo, ya sea a través de hardware desarrollado específicamente para Arduino, como utilizando hardware más general.
- Página Web propia, http://arduino.cc/, disponible en varios idiomas. Contiene gran cantidad de información que facilita una inmersión rápida.

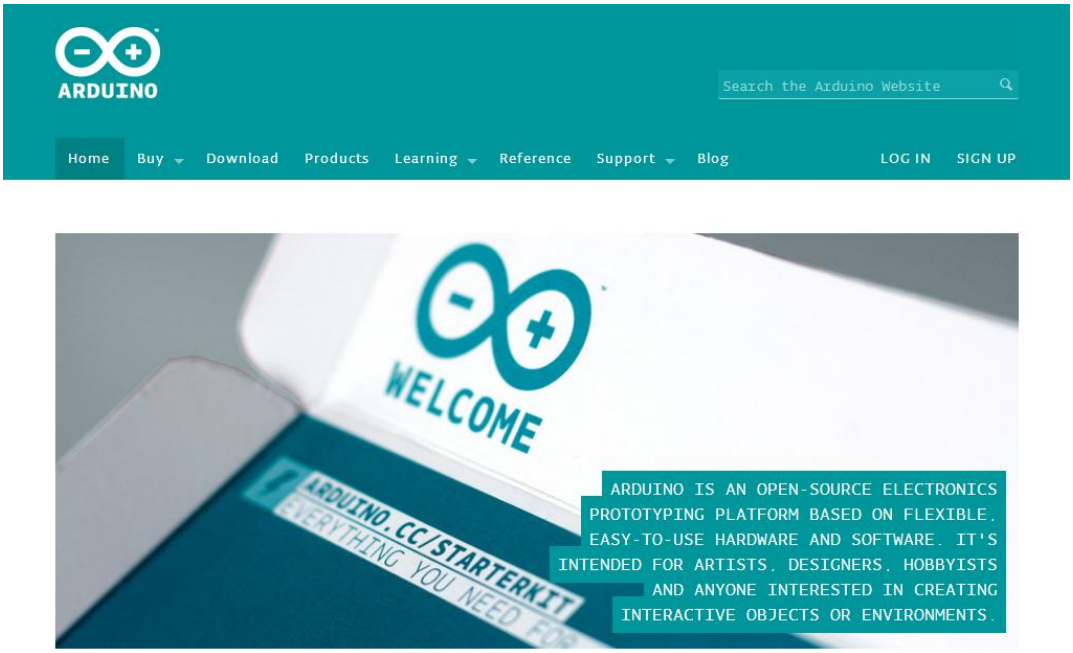



Figura 16. Pagina web Arduino. (Fuente: http://arduino.cc/en)

- *Homepage*: con información acerca del producto y la empresa.
- *Buy*: incluye los canales para adquirir sus productos, con enlaces a tiendas o distribuidores.
- *Download*: sección desde la que descargar el software necesario para la puesta en marcha.
- *Products*: dispositivos como placas, shields o kits desarrollados por ellos, hojas de documentación, descripción del producto, etc.
- *Learning*: guía con los primeros pasos, numerosos ejemplos y el Playground, una sección en la que compartir información y programas.

- *Reference*: contiene todo lo relacionado acerca del lenguaje como funciones, estructuras de control, tipos de datos o creación de librerías.
- *Support*: sección con las preguntas más frecuentes y el foro.

### 2.3.3. Selección de placa.

Una vez seleccionada la plataforma de programación, es necesario elegir la placa. A la hora de escoger, se tuvieron en cuenta los siguientes requisitos:

- Tamaño reducido.
- Mínimo 10 entradas/salidas digitales, para control del driver y alguna función extra que pudiera ser necesaria.
- Posibilidad de comunicación I2C, necesaria para comunicar con el sensor MPU.
- Mínimo de 2 puertos PWM, con los que controlar la velocidad de los motores.

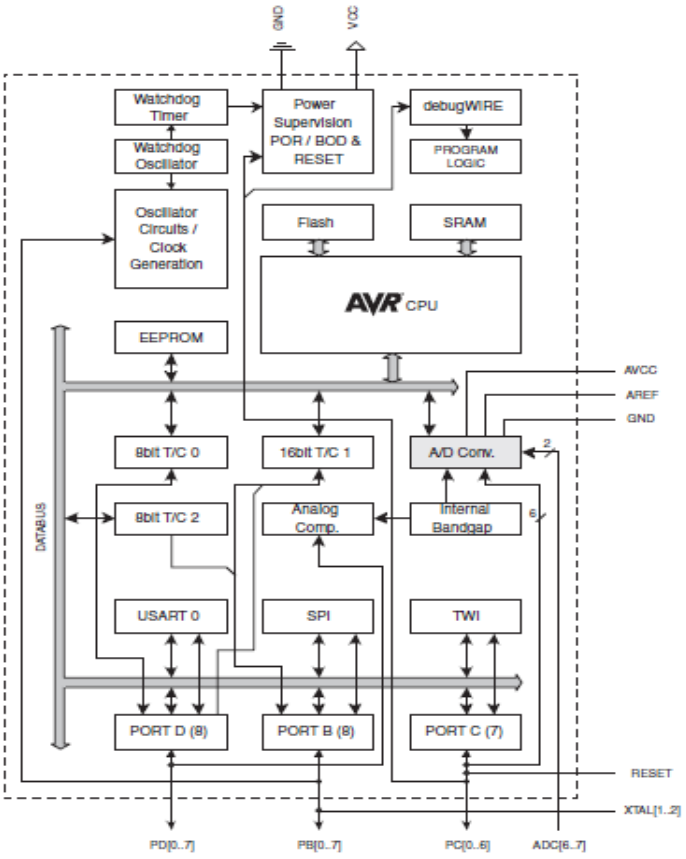



Figura 17. Diagrama de bloques del ATmega328. (Fuente: Atmega328 Datasheet).

Se estudiaron 3 opciones del amplio catalogo existente en Arduino:

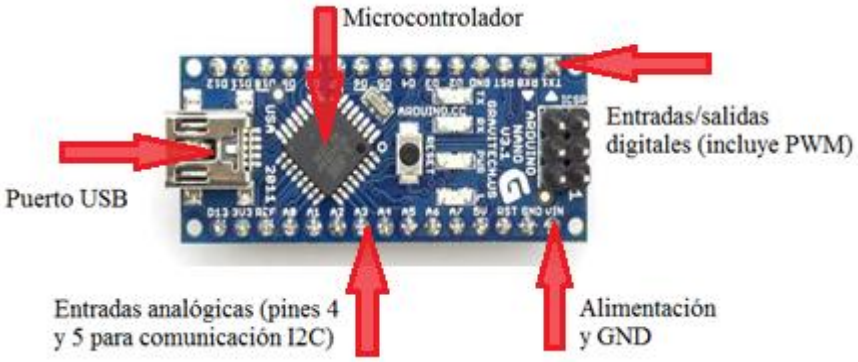
- Arduino Mega: basada en el microcontrolador ATmega2560, funciona con un oscilador de cristal de 16MHz y dispone de 54 entradas/salidas digitales, 15 de ellas con posibilidad de PWM. Presenta también 16 entradas analógicas, 4 puertos de comunicación serie UART, conexión USB e ICSP (In Circuit Serial Programming). Permite comunicación serie, SPI y TWI (I2C). Esta opción fue descartada por el tamaño de la placa, la mayor de las 3, y por estar sobredimensionada en todos los requisitos.


Figura 18. Arduino Mega.(Fuente: http://arduino.cc/en).

- Arduino Uno: microcontrolador ATmega328 y oscilador de cristal a 16 MHz Presenta 14 pines de entrada/salida digitales, 6 de los cuales pueden ser usados como PWM. Dispone también de 6 entradas analógicas, conexión USB e ICSP. Permite, al igual que la placa anterior comunicación serie, SPI y TWI. Esta fue la opción elegida para comenzar a trabajar.


Figura 19. Elementos destacados del Arduino Uno. (Fuente: http://arduino.cc/en).

- Arduino Nano: de características análogas a la placa anterior ya que incorpora el mismo microcontrolador aunque en otro encapsulado. Las diferencias más notables son su menor tamaño y su conexión vía mini-USB. Fue la opción utilizada finalmente y la incluida en el diseño de la placa de circuito impreso.




-

Figura 20. Elementos destacados del Arduino Nano. (Fuente: http://arduino.cc/en).

## 2.4. Sistema de control.

Un sistema de control constituye un elemento de regulación para sí mismo u otros sistemas. Pueden ser en lazo abierto si no tienen en consideración la salida del sistema, o en lazo cerrado cuando sí lo hacen.

Debido a la inestabilidad del sistema a controlar, necesitaremos un control en lazo cerrado, es decir, aquel en el cual es tenida en cuenta la señal de salida mediante una realimentación. Esta característica proporciona una mayor estabilidad.

Los sistemas de control en lazo cerrado se encargan de actuar en función de una entrada de referencia o estado deseado. Para ello la comparan con la señal de salida del sistema, que en nuestro caso llegará recogida por los sensores, obteniendo el error. En función de esa señal de error se elabora una acción que enviar a los actuadores, todo esto con el fin de obtener la respuesta adecuada, lograr el ángulo 0 o de equilibrio del robot [23].

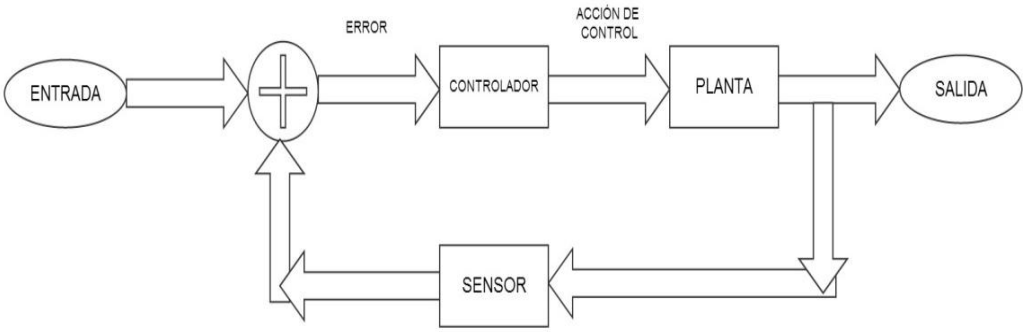



Figura 21. Diagrama de control.

Entre los controladores en lazo cerrado nos encontramos con:

- Control on/off (todo o nada): siempre aplica la máxima acción correctiva, produce mucha oscilación y es muy sensible al ruido. Al actuar siempre de forma extrema, puede provocar desgaste en la mecánica de los actuadores.

- Control proporcional: tiene una pendiente lineal en la banda proporcional, fuera de ella satura y se comporta como el anterior. Este tipo de control hace que siempre haya error en estado estacionario, es decir, nuestro robot nunca se detendría por completo.

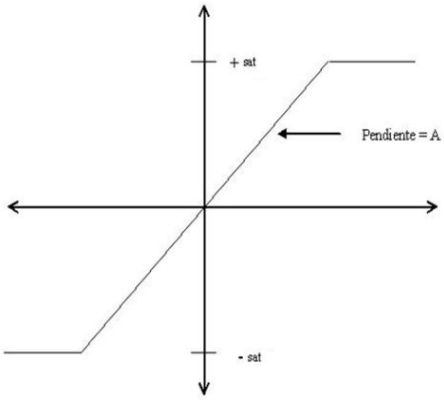



Figura 22. Control proporcional. (Fuente: Wikipedia).

- Control PID (proporcional integral derivativo): son los más ampliamente utilizados. Incorpora al control proporcional 2 términos más.
    - El integral tiene un efecto memoria, al integrar el error, cuanto más tiempo se mantenga este sin hacerse nulo, mayor será la señal de actuación. Influye en el término proporcional y no responde a cambios bruscos de la señal de error.

$$I_{\text{sal}} = K_i \int_0^t e(\tau)\, d\tau$$

Figura 23. Término integral. (Fuente: Wikipedia).

- El derivativo puede interpretarse como un efecto anticipativo, además mejora la estabilidad del sistema a costa de introducir algo de retardo. No afecta a errores estáticos o que cambian de forma muy lenta y es muy sensible al ruido.

$$D_{\text{sal}} = K_d \frac{de}{dt}$$

Figura 24. Término derivativo. (Fuente: Wikipedia).

Con el control PID obtenemos finalmente un sistema que actuará sobre errores actuales (parte proporcional), errores pasados (parte integral), y errores futuros (parte derivativa) [24].

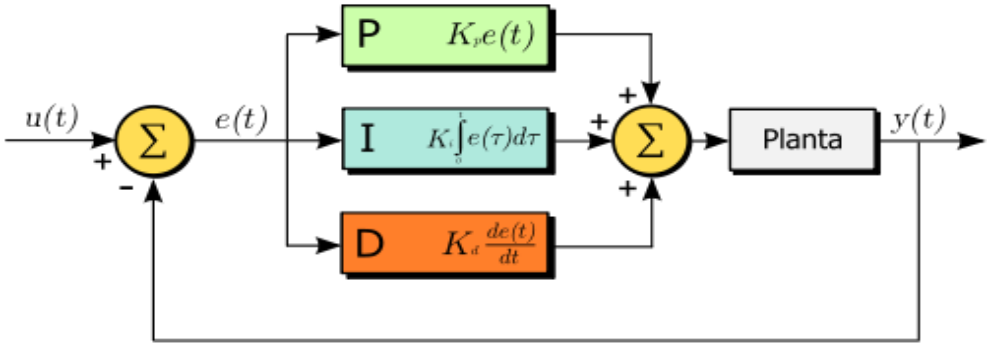



Figura 25. Diagrama de bloques del sistema (planta) compensado en lazo cerrado mediante un controlador PID. (Fuente: Wikipedia).

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

Figura 26. Salida del sistema. (Fuente: Wikipedia).

Como se ha visto a la hora de abordar el péndulo invertido, para este tipo de robot podrían implementarse varios tipos de sistemas de control. Por ejemplos vistos en otros robots de este tipo, con este tipo de control debería ser suficiente, aunque tampoco quedaba descartado que fuera necesario el empleo de otros.

## 2.5. Sistema de potencia.

### 2.5.1. Actuadores.

Para corregir la posición del robot y la desviación del ángulo, necesitaremos ejercer un par adecuado en el punto de balanceo, esto se logrará utilizando motores.

Un motor es un dispositivo que transforma energía eléctrica en energía cinética o mecánica. El movimiento generado suele ser circular, y producido gracias a la acción de un campo magnético generado.

Se emplearán motores de corriente continua ya que el objetivo es un robot autónomo alimentado mediante baterías. Los más típicos en aplicaciones de electrónica son:

- Servomotor: motor de corriente continua que destaca por la capacidad de orientarse y mantener estable una posición o ángulo, siempre dentro de los límites de su movimiento. Además de la posición, también puede ser controlada su velocidad. Formados por un motor, una reductora y un circuito de control, presentan la gran desventaja de tener una potencia de trabajo reducida [25] [26].

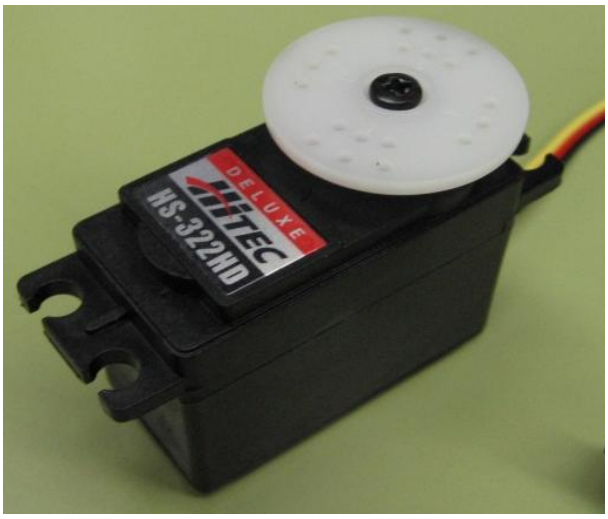



Figura 27. Servomotor. (Fuente: Wikipedia).

- Motor paso a paso: dispositivo que convierte una secuencia determinada de pulsos eléctricos en un movimiento circular discreto, de un número determinado de grados, varía en función de la construcción del motor.

Presenta la ventaja de tener una alta precisión en sus desplazamientos. Por el contrario resaltar que las posiciones, o ángulos formables, que pueden alcanzar estos motores son finitas, por lo que no podrían ser lo suficientemente ajustables a nuestras necesidades [27] [28].

Figura 28. Motor paso a paso. (Fuente: http://www.bricogeek.com).

- Motor de corriente continua: tiene un par de giro elevado, y prácticamente constante, que es regulado mediante corriente. Puede regularse su velocidad y par generados actuando sobre su alimentación y mediante PWM. El sentido de giro de estos motores es también controlable según la disposición de la alimentación, cambiando el sentido al invertir la polaridad de sus terminales como se aprecia en la imagen [29] [30].

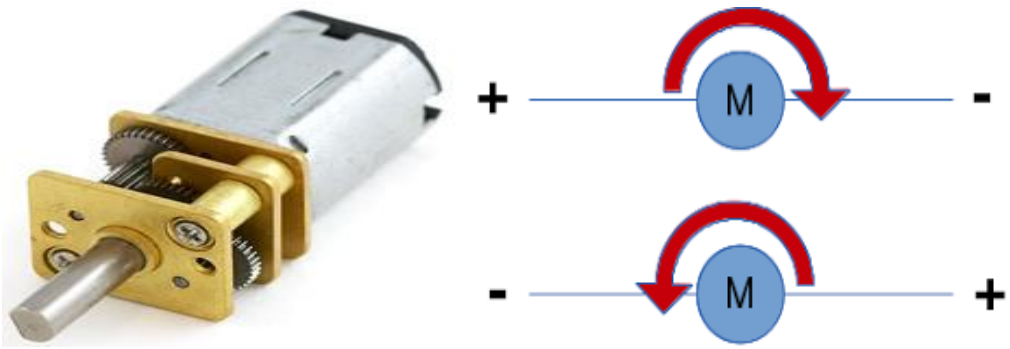



Figura 29. Izda. Motor DC. Dcha.relación sentido de giro/polaridad. (Fuente: http://www.bricogeek.com/shop/motores/112-motor-micro-metal-dc-con-reductora-5-1.html http://fuenteabierta.teubi.co/2013/08/control-de-motores-dc-con-arduino-nano.html)

Este será el tipo de actuador seleccionado ya que es el más sencillo y el que mejor se ajusta a las necesidades. De entre la variedad de motores existentes de este tipo, se escogió un motor con reductora como el de la imagen, lo cual incrementará el par obtenido en perjuicio de la velocidad.

### 2.5.2. Transmisión de la potencia.

La limitación de los microcontroladores a la hora de entregar corriente, necesaria para alimentar los motores, hace necesaria la inclusión de un dispositivo que sea capaz de suministrarla. Este es el lugar ocupado por los drivers, circuitos integrados encargados de suministrar la información del microcontrolador con la potencia adecuada.

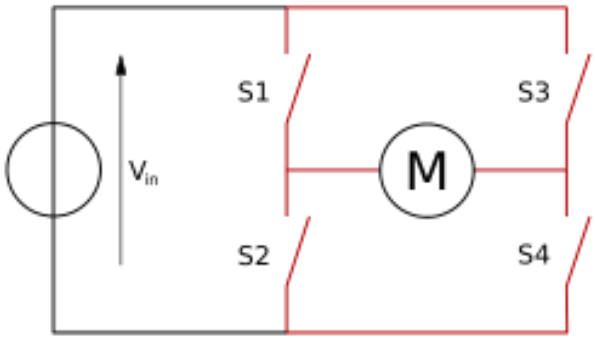



Figura 30. Puente en H. (Fuente: Wikipedia).

Existe dentro de este tipo de integrados, unos que podrían ser de especial interés, los puentes en H. Estos circuitos permiten el cambio del sentido del giro del motor de una forma sencilla, además, son totalmente compatibles con el control de velocidad por PWM. Conseguiríamos así la posibilidad de modificar sentido y velocidad a conveniencia [31].

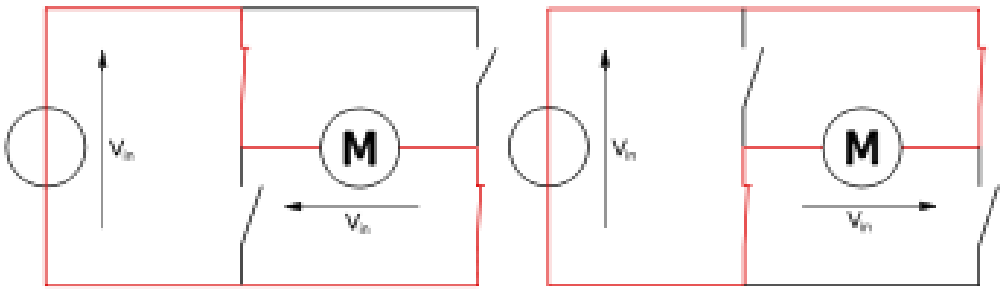



Figura 31. Sentidos de giro 1 y 2. (Fuente: Wikipedia).

Tras realizar una búsqueda de opciones en varias páginas web de componentes, se consiguen 3 candidatos:

- L6206: contiene dos puentes completos, la intensidad proporcionada a la salida puede llegar a los 2.8A de forma continuada. El montaje de los puentes está realizado con tecnología MOS y contiene diodos de protección. El encapsulado tipo PDIP, que no cuenta con la opción de añadir un radiador, hace presuponer que no será un dispositivo que se caliente en exceso. El principal punto en contra lo presenta el precio, puesto que es el mayor de los 3 de forma notable (9.5 €) [32].

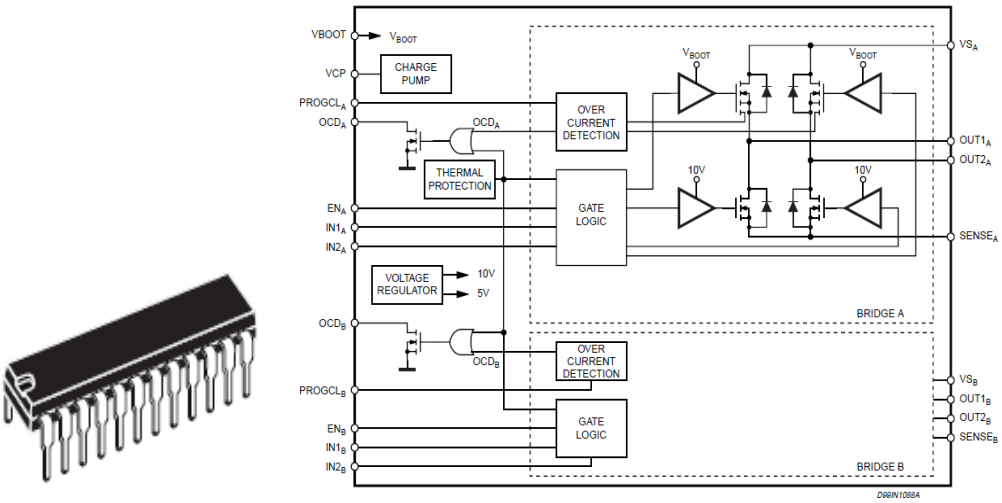



Figura 32. L6206 encapsulado y diagrama. (Fuente: L6206 Datasheet).

- L298: al igual que el anterior también presenta un puente completo doble. Proporciona 2A de corriente de salida nominal, esta implementado en tecnología BJT y no incluye diodos de protección. El encapsulado en el que se presenta este componente ya da a entender que podría generar mucho calor, por lo que podría ser necesaria la instalación de un disipador, algo que es confirmado en el proyecto "DISEÑO Y CONSTRUCCIÓN DE UN ROBOT SIGUE LÍNEAS CONTROLADO A DISTANCIA CON INTERFAZ ANDROID". A favor estaría el ser una opción más asequible (su precio ronda los 3.3€) [33].

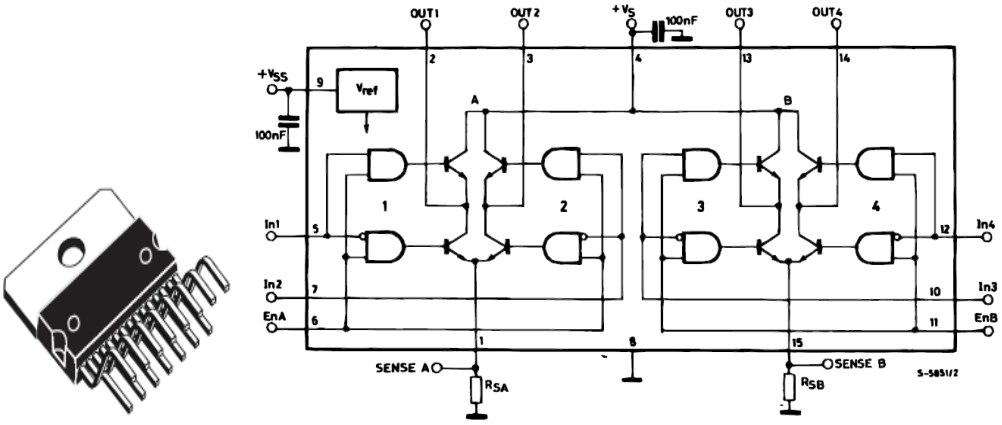



Figura 33. L298 encapsulado y diagrama. (Fuente: L298 Datasheet).

- L293D: se trata de un medio puente en H cuádruple, está separado en 2 puentes independientes que juntos forman un puente completo doble. Proporciona 600mA de corriente de salida nominal pudiendo llegar a picos de 1.2A. Implementado en tecnología compatible TTL. El encapsulado PDIP hace prever que tampoco se trata de un componente que se caliente en exceso, algo que es confirmado en varios foros de electrónica y Arduino. El precio también es reducido (2.6€) [34].

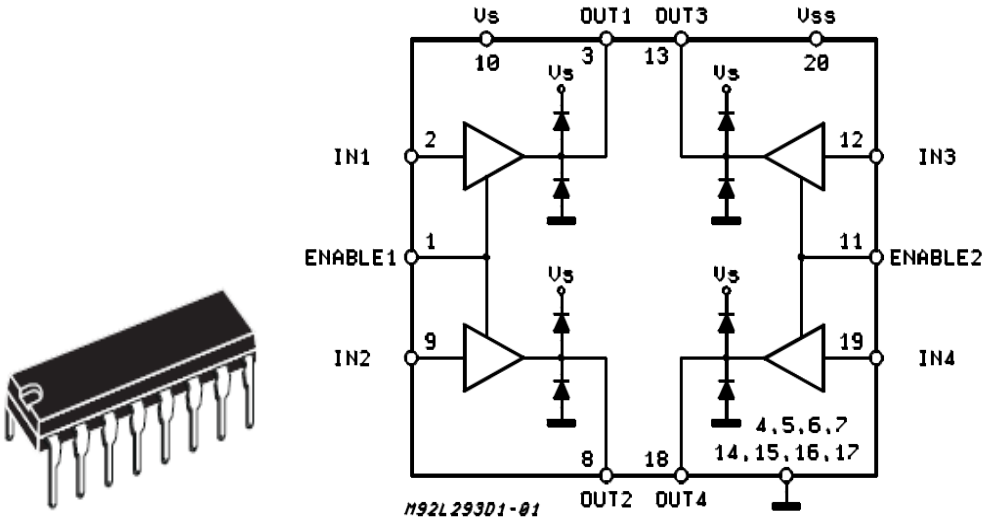



Figura 34. L293D encapsulado y diagrama. (Fuente: L293D Datasheet).

## 2.6. PCB.

El objetivo principal del proyecto era la creación de una PCB (Printed Circuit Board), una placa en la que se agruparían los elementos anteriormente citados y que componen el robot.

Las PCB o placas de circuito impreso están compuestas por una base no conductora, suelen ser un laminado de fibra de vidrio reforzada o plástico, sobre la que están incluidos unas pistas o caminos de material conductor, generalmente cobre. Son utilizados para sostener y conectar elementos electrónicos discretos.

A continuación se muestra el proceso seguido durante la realización y los resultados finales obtenidos.

- Montaje en protoboard: para una primera toma de contacto con los componentes, y localizarlos en el espacio primero se realizó el montaje en una protoboard. Este hecho resulto de gran utilidad para conocer las necesidades que podrían surgir en cuanto a espacio, conexión con otros elementos, etc.

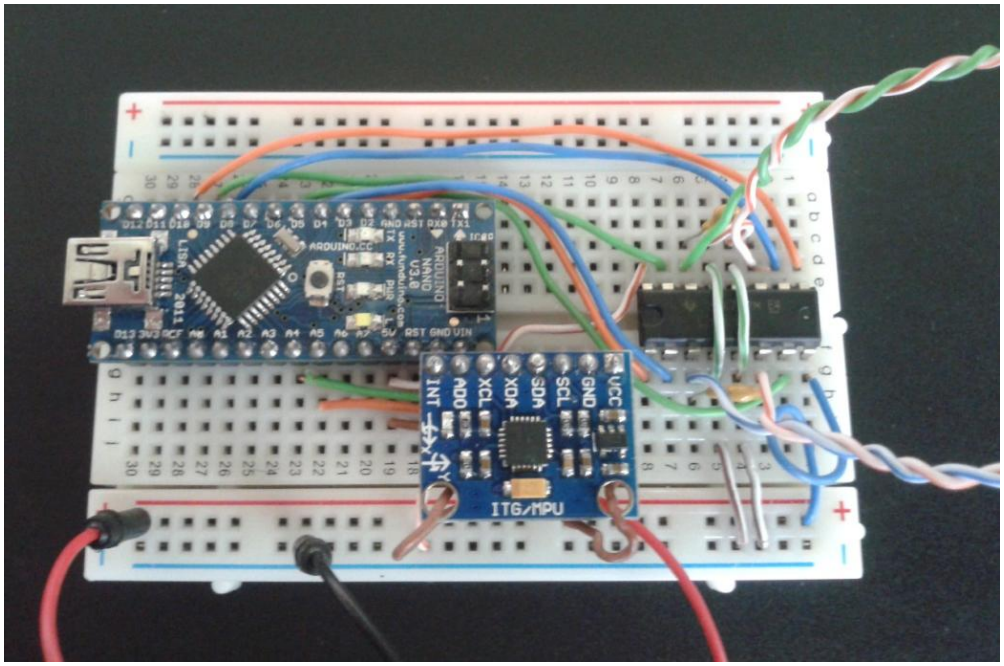



Figura 35. Montaje en protoboard.

- Posteriormente se procedió a implementar la placa con el software visto en la asignatura de Fabricación y Ensayo de Equipos Electrónicos [35], *Design Spark*. Este programa requiere crear un archivo que incluya el esquemático del sistema con las conexiones entre los distintos elementos. Puede verse en la siguiente imagen, que incluye el Arduino Nano, el driver y los distintos conectores para baterías y motores.

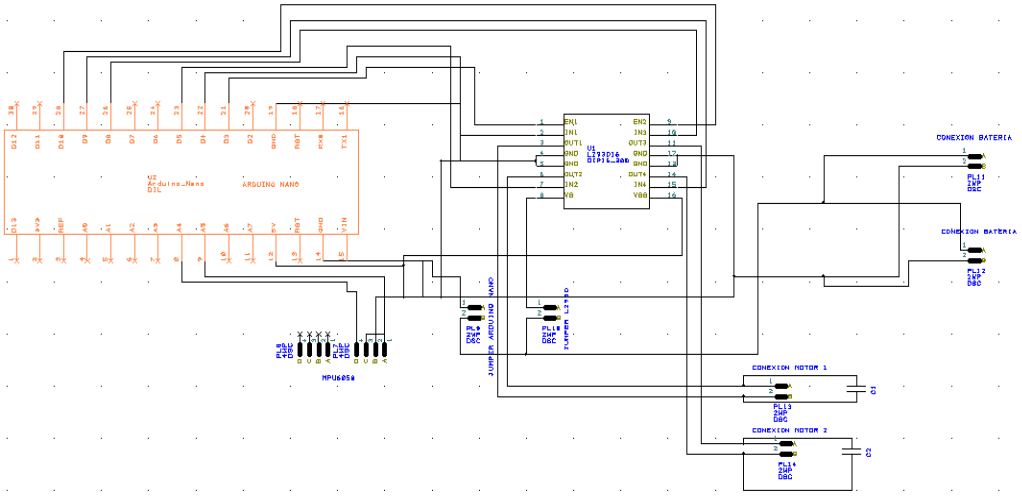



Figura 36. Esquemático del sistema.

- Tras crear el archivo con el conexionado, se volcó en uno tipo pcb para, desde este, comenzar a crear la placa propiamente dicha. Además de la disposición de los elementos, para optimizar el funcionamiento de la placa, se tuvieron en cuenta algunas condiciones:
    - Longitud de las pistas: para evitar caída de tensión en la pista, lo que conllevaría una peor transmisión de señal, estas deben ser lo más cortas posibles. En nuestro diseño esto no fue del todo posible ya que también se ve comprometido este apartado por la disposición de los elementos.
    - Disposición de las pistas: tiene que permitir la máxima funcionalidad del sistema, algunos de los detalles más tenidos en cuenta fue la colocación del Arduino de forma que el conector USB del mismo quedara accesible, que el sensor se encontrará lejos de las fuentes de

potencia, y que entre las vías de señal y de alimentación hubiera pistas de tierra que las protegieran.

- Minimización del espacio utilizado: para que se desperdicie la menor cantidad de material posible, se disminuye dentro de un margen adecuado el tamaño del mismo, se eliminan espacios vacíos entre pistas y se disponen los elementos de forma adecuada. En nuestro caso se ha considerado además utilizar *jumpers* en lugar de interruptores
- Otros: se han eliminado islas de los planos de masa, ya que pueden provocar interferencias. También para reducir interferencias se han intercalado pistas de tierra entre aquellas que transmiten señal y potencia.

Todo esto puede verse de una forma más detallada a continuación:

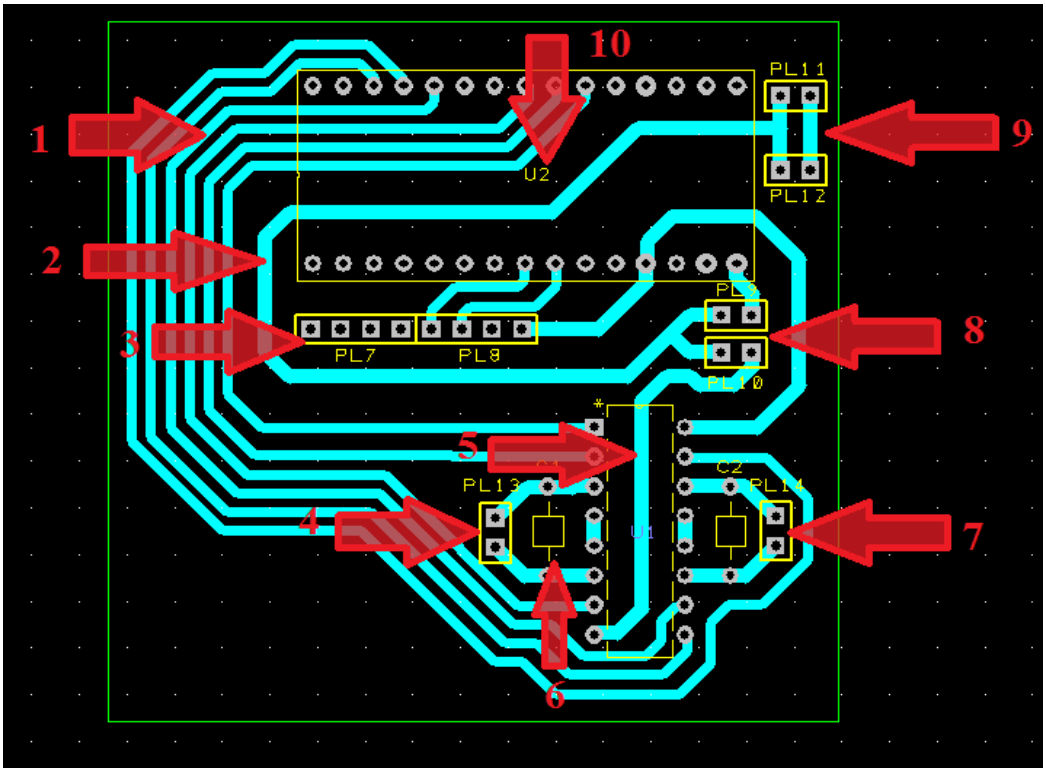



Figura 37. Detalle de la PCB.

1. Pistas correspondientes al conexionado entre el Arduino Nano y el driver. Al estar destinadas para enviar datos transmitirán poca potencia, por ello su anchura se ha considerado como oportuna en 0.8 mm.

2. Pista de alimentación. Será la encargada de suministrar la energía al sistema, parte de los conectores para las baterías (ref.9), y llega hasta los jumpers que conectarán con Arduino y el driver.
3. Zócalos para colocar el sensor, en ellos se conectará el MPU6050.
4. Corresponden a los conectores para uno de los motores, serán del tipo de la imagen, con tornillo que permita un mejor ajuste.

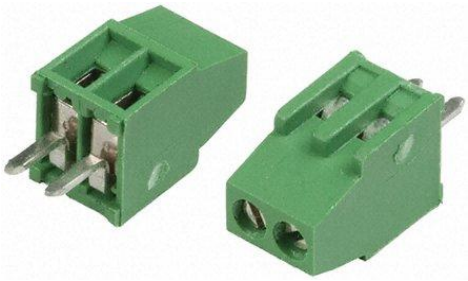


Figura 38. Conectores PCB. (Fuente: http://es.rs-online.com/web/).

5. Driver L293D, suministrara la energía necesaria a los motores.
6. Condensadores, para filtrar posibles ruidos en la alimentación de los motores.
7. Corresponden a los conectores para el segundo de los motores.
8. Jumpers, se han incluido estos conectores para diferenciar la alimentación de la placa Arduino de la del driver, esto se ha hecho así con el objetivo de que pudiera, si surgía la necesidad, utilizar el Arduino sin que el movimiento de los motores molestara. Destacar que se han utilizado estos elementos para minimizar lo máximo posible el espacio de placa, algo que no habría sido posible usando interruptores.

Figura 39. Jumper. (Fuente: http://es.rs-online.com/web/).

9. Conectores para las baterías. En ese lugar se colocarán unos conectores como los que se utilizan para los motores (refs. 4 y 7).
10. Emplazamiento y conectores para el Arduino Nano.

Para finalizar se incluye el resultado final, que incluye además de los elementos vistos anteriormente, las conexiones y planos de masa

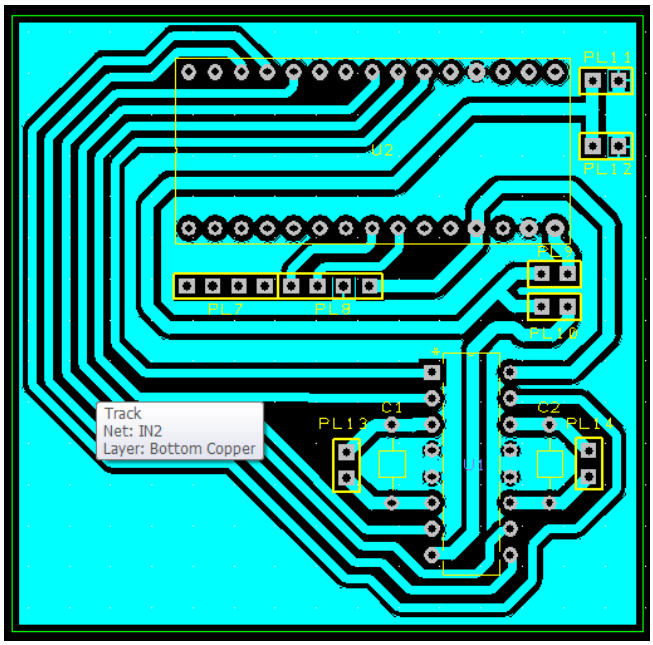



Figura 40. Diseño final del PCB.

Otra herramienta incluida en Design Spark y que resulta de mucha utilidad a la hora de visualizar el posible acabado final, es la generación de modelos 3D. Los correspondientes a nuestra placa ya acabada son los siguientes:

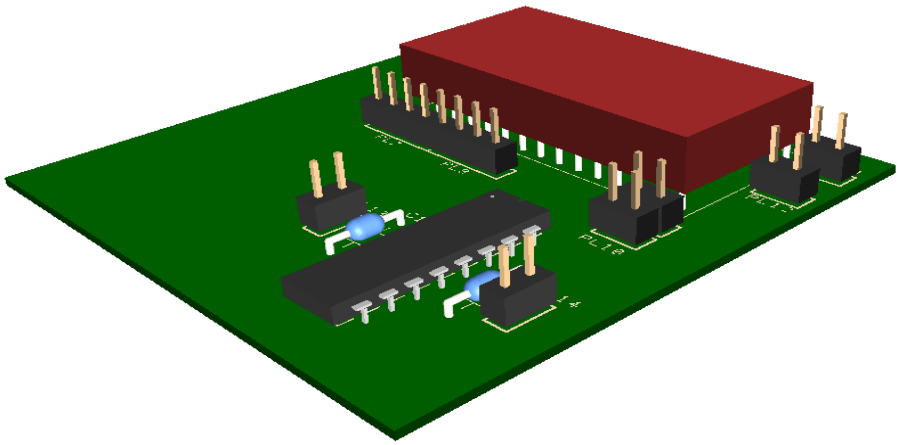


Figura 41. Vista 3D superior.

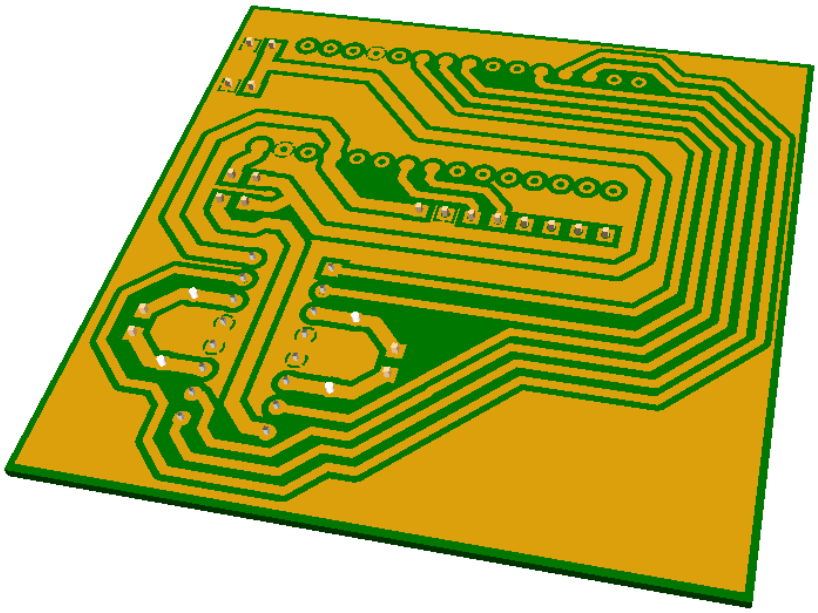


Figura 42. Vista 3D inferior.

# 3. Desarrollo.

Una vez establecida la base del proyecto que queda expuesta en el apartado anterior, comenzó el desarrollo del mismo, lo que requería concretar ciertos aspectos que aun no habían sido estudiados. En este capítulo se expondrán los pasos seguidos en orden cronológico, dejando para el próximo apartado las modificaciones, ensayos y pruebas que se realizaron hasta llegar al prototipo final.

## 3.1. Primeros pasos con Arduino.

Una vez seleccionada la placa se comenzó a trabajar con ella. Al ser Arduino un lenguaje no muy conocido hasta el momento se empezó por lo más básico, familiarizarse con el entorno y el lenguaje de programación. Esta tarea se vio muy favorecida, como ya se ha mencionado con anterioridad, por toda la información accesible a través de su página web y el foro relacionado [36]. Merecen especial interés:

- Reference: contiene el Language Reference, donde se explican las estructuras de programa y de control, sintaxis relacionada con dichas estructuras, tipos de variables (constantes, tipos de datos, conversión entre tipos) y algunas funciones de utilidad, todo ello acompañado de ejemplos.
- Learning: dentro de esta sección encontramos Examples, que incluye gran cantidad de programas. Se encuentran como los anteriores comentados.

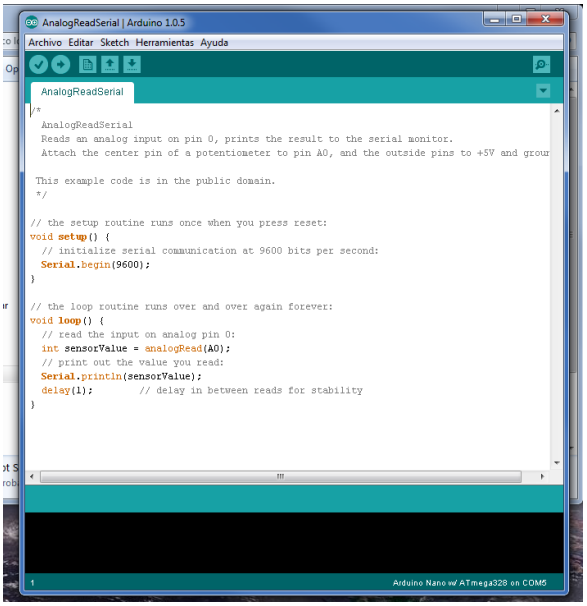



Figura 43. Interfaz de programación Arduino.

## 3.2. Obtención de datos MPU6050.

### 3.2.1. Especificaciones técnicas del sensor.

El sensor escogido, el IMU MPU6050 del fabricante Invensense [37], presenta las siguientes características:

- Giróscopo:
  - Sistema de giróscopo en 3 ejes fabricado con tecnología MEMS.
  - Puede ser ajustado a valores de ±250, ±500, ± 1000, y ±2000 °/s.
  - Integra 3 conversores analógico digital de 16 bits, uno por cada eje.
  - Dispone de un filtro paso bajo programable
  - Consume 3.6mA.
  - Dispone de self-test programable.
- Acelerómetro:
  - Sistema de acelerómetro en 3 ejes fabricado con tecnología MEMS.
  - Ajustable a valores de sensibilidad de ±2*g*, ±4*g*, ±8*g* y±16*g*.
  - Integra 3 conversores analógico digital de 16 bits, uno por cada eje.
  - Consumo de 500µA.
  - Dispone de interrupciones programables.
  - Detección de caída libre, vibraciones y movimiento 0.
  - Dispone de self-test programable.

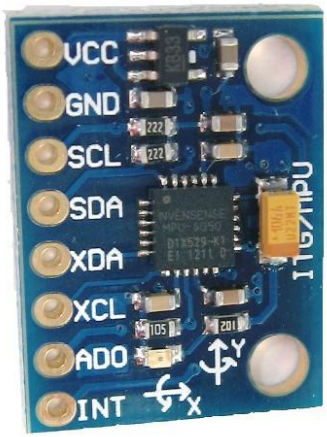



Figura 44. MPU6050. (Fuente: http://www.arduino.cc/).

- Otras:
    - Posibilidad de ser ampliado con un magnetómetro a un sistema de 9 grados de libertad.
    - Master auxiliar para comunicación I2C.
    - Regulador de tensión propio.
    - Buffer o memoria de 1024 bytes FIFO.
    - Sensor de temperatura.
    - Filtros programables por el usuario para los acelerómetros, los giróscopos y el sensor de temperatura.
    - Dispone de un DMP (Digital Motion Processor), un procesador disponible para el usuario que permitiría descargar de trabajo al microcontrolador principal.
    - Interfaces de comunicación SPI e I2C primaria y secundaria.

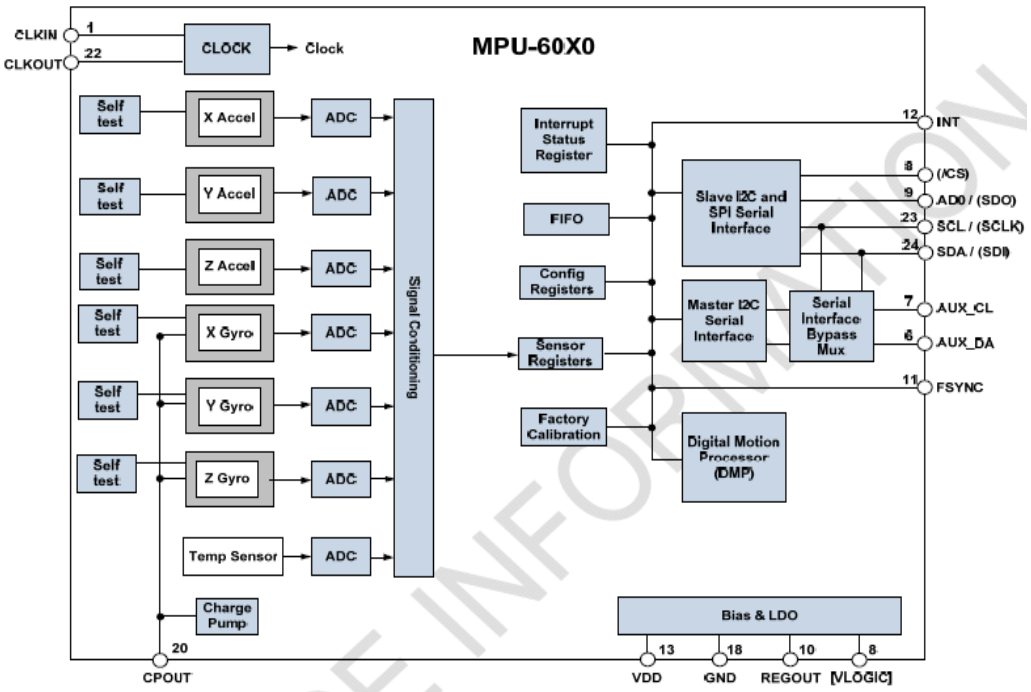



Figura 45. Diagrama de bloques MPU 6050. (Fuente: MPU6050 Datasheet).

### 3.2.2. Comunicación I2C.

I2C es un bus de transmisión de datos en serie creado en 1992 por Philips. La velocidad de comunicación es de 100 kbit/s aunque puede aumentar hasta 3.4 Mbit/s. Es un sistema muy extendido en la comunicación entre microcontroladores y sus periféricos.

Utiliza 2 líneas de transmisión, SDA (Serial Data Line) por la que circulan los datos, y SCL (Serial Clock Line) en la que se envía la señal de reloj. Cada dispositivo conectado al bus tiene una dirección asignada [38] [39] [40].

La transmisión es iniciada por el dispositivo maestro, que además proporciona la señal de reloj. Tiene la forma:

1→ START /Bits dirección de esclavo/Bit RW/Bit Ack/ Byte dirección de memoria / Bit ACK / Byte de datos / Bit ACK /Bit ACK / STOP /

2→ Master / Master /Master/ Slave / Master Slave / Master-Slave /( / Slave-Master /Master/

Bit de RW, 0 es Leer y 1 Escribir.

Bit de ACK (Acknowledge).

Master – Slave según sea lectura-escritura.

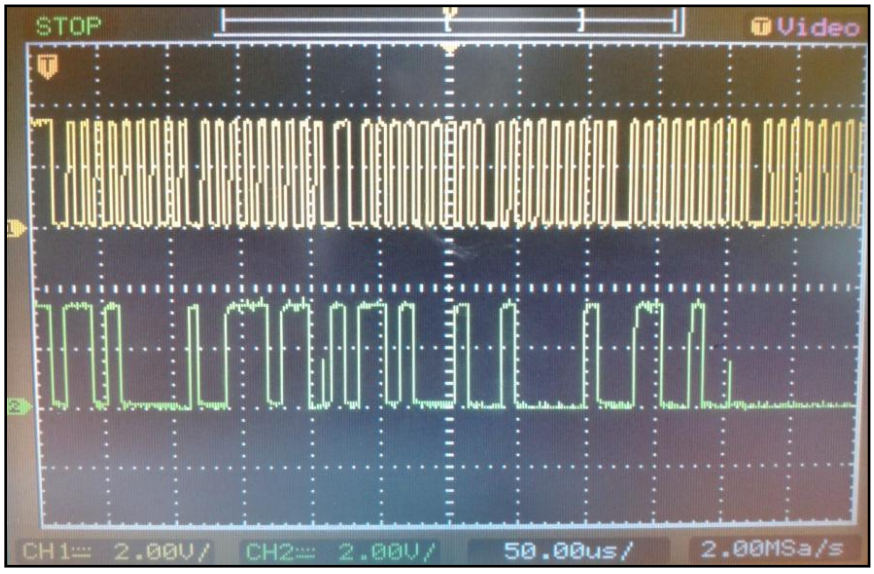



Figura 46. Trama I2C.

Una ventaja de este método de conexión con el sensor es la existencia para Arduino de la librería Wire, en la que se incluyen funciones para este tipo de comunicación.

### 3.2.3. Obtención de datos.

Tras realizar la conexión que se ve en la imagen, se cargó el primer programa que consistía en el establecimiento de la comunicación con el sensor, activación del mismo, y lectura de los registros en los que se almacenan los datos obtenidos. El código puede encontrarse en el Anexo 1 así como los primeros resultados obtenidos.

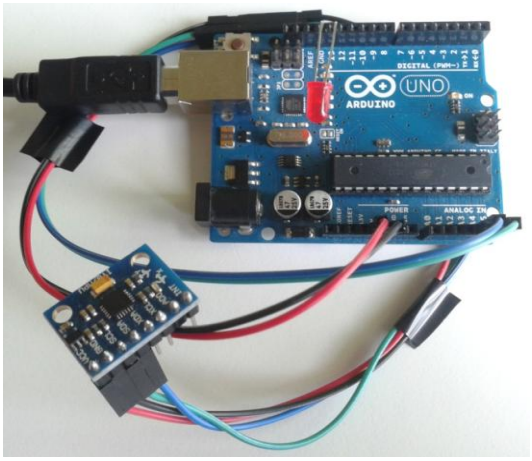



Figura 47. Conexionado del sensor.

Posteriormente se incluyeron funciones que habilitaban el uso del software Parallax-DAQ, que permite enviar datos al programa Excel, lo cual es muy útil para tratar los datos y obtener gráficas. El código de estas funciones se encuentra en el Anexo 2.

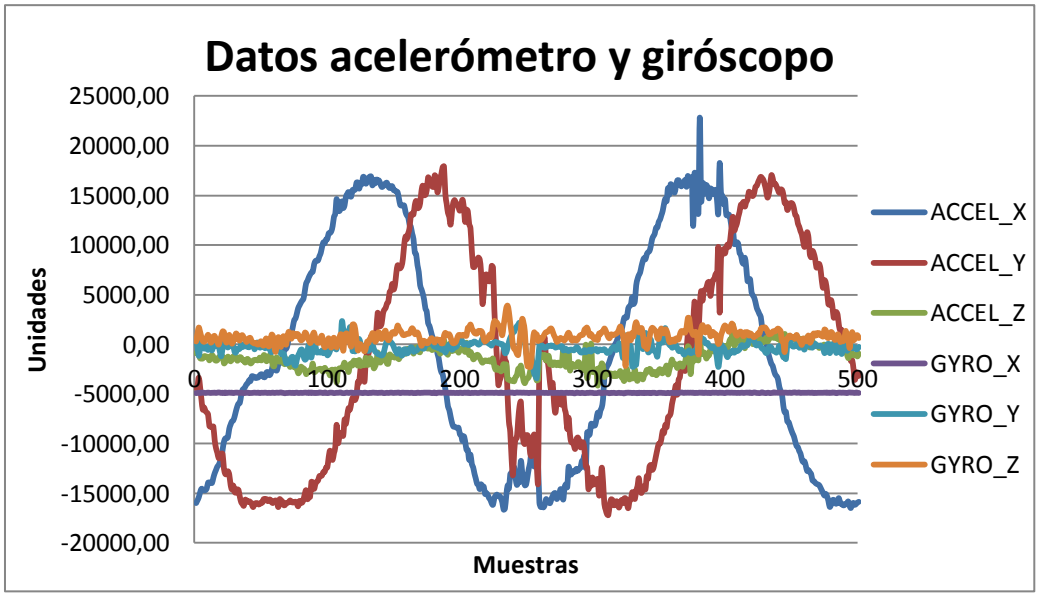



Figura 48. Toma de datos con el sensor en movimiento 1.

### 3.2.4. Obtención del ángulo.

Para la obtención del ángulo se desarrollaron 2 nuevas funciones, una para transformar los datos obtenidos por el acelerómetro y otra para los del giróscopo.

- Acelerómetro: de este sensor obtenemos información de cómo afecta la gravedad a los ejes, el ángulo formado se calculará a partir de una relación trigonométrica entre los valores obtenidos. Teniendo en cuenta esto se desarrolló la función incluida en el Anexo 3.
- Giróscopo: nos proporciona medidas de velocidad angular, para la obtención del ángulo será necesaria una función que integre dichos valores. El código se encuentra en el Anexo 4.

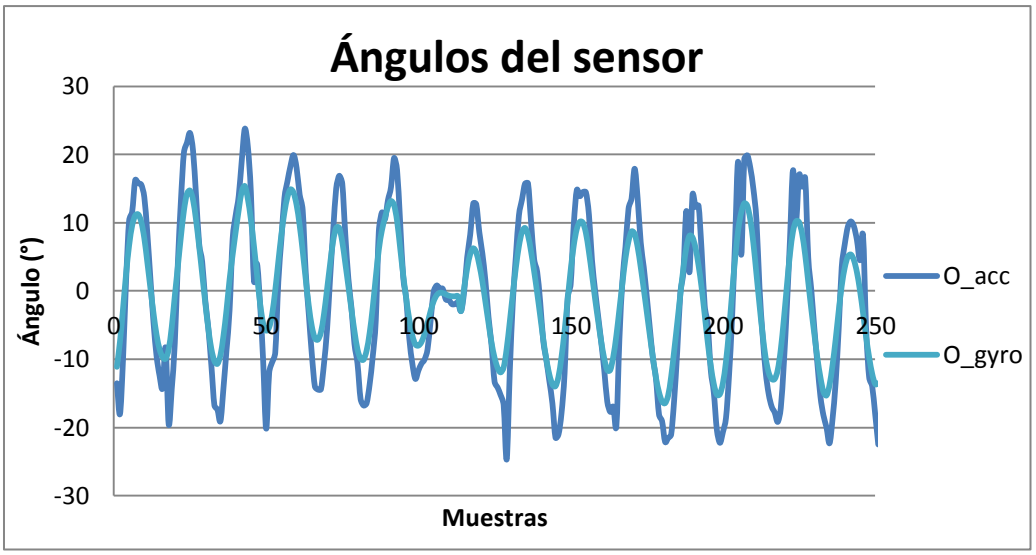



Figura 49. Ángulos obtenidos con acelerómetro (O_acc) y .giróscopo (O_gyro).

De los resultados obtenidos cabría destacar que el ángulo obtenido mediante los acelerómetros varía con mayor rapidez siendo también más ruidoso. Por el contrario, el generado por los giróscopos varía de forma más lenta aunque contiene menor cantidad de ruido.

La combinación de ambos para obtener una estimación del ángulo que aúne las mejores propiedades de una y otra medida se verá en el siguiente apartado.

### 3.2.5. Filtro complementario.

El uso de los ángulos obtenidos mediante las funciones del apartado anterior puede presentar problemas o inconvenientes como offsets, los ruidos de las medidas con los acelerómetros, que pueden indicar valores falsos, o derivas producidas al tener que integrar los valores del giróscopo. Para poder obtener medidas que se ajusten lo máximo posible a la realidad del sistema, necesitaremos de algún otro medio como el filtro complementario.

El uso de este método requiere de un acondicionamiento previo de los valores obtenidos, en primer lugar deben ser eliminados los offsets de los sensores, algo que es implementado con la función del Anexo 5. También es necesario ajustar o escalar los datos obtenidos por los sensores de una forma adecuada, esto se hará teniendo en cuenta la sensibilidad, se encuentra ya incluido en los anexos 3 y 4 de obtención de ángulos. Obtendremos así los datos de ángulo deseados.

El filtro complementario basa su actuación en los siguientes puntos:

- Fórmula:
  Ángulo = (valor filtro paso alto)*(ángulo + valor integrado del giróscopo) + (valor filtro paso bajo)*(valor del acelerómetro);
- Filtro paso bajo: este tipo de filtro no deja pasar cambios rápidos, como en la medida obtenida de los acelerómetros. De esta manera eliminamos el ruido propio de este sensor.
- Integración: como se ha comentado previamente, este elemento es necesario para obtener el ángulo a partir de medidas de velocidad angular.
  Ángulo giróscopo = ángulo previo +valor del giróscopo * tiempo
- Filtro paso alto: es el método empleado para eliminar la deriva de los giróscopos. Este tipo de filtro funciona de manera contraria al paso bajo, permitiendo el paso de cambios rápidos y evitando el de cambios lentos (deriva).
- Constante de tiempo: a la hora de obtener los valores para los filtros, es tenido en cuenta la velocidad a la que se muestrea el sistema o se realiza el bucle de medida, así como hasta que punto actuará en las señales de los sensores.

Constante de tiempo = (valor filtro paso alto * tiempo de muestreo)/(1 - valor filtro paso alto).

- Complementario: indica que las 2 partes que componen el filtro suman 1. Apreciable en los términos (valor filtro paso alto) y (1 - valor filtro paso alto).

El esquema obtenido por tanto sería:

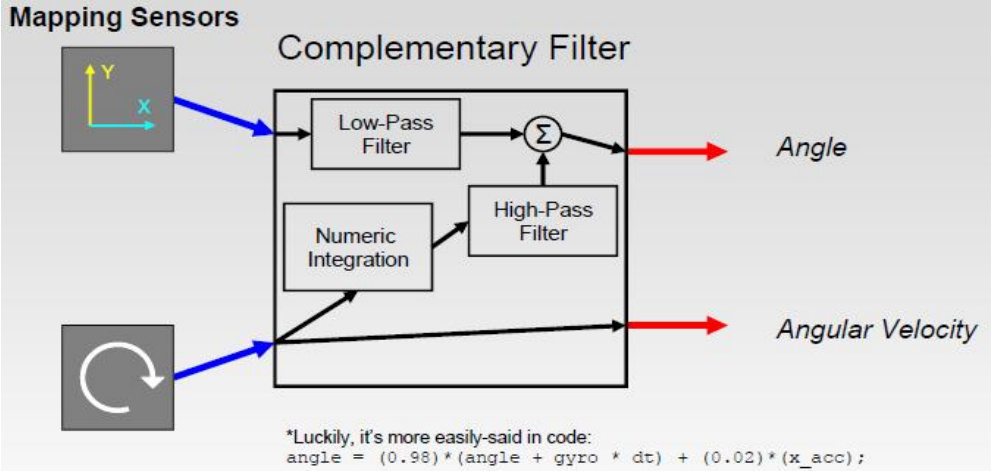



Figura 50. Diagrama filtro complementario [41]. (Fuente: Shane Colton. A Simple solution for Balance Filter. MIT. June 2007).

El resultado que se obtuvo del empleo del filtro:

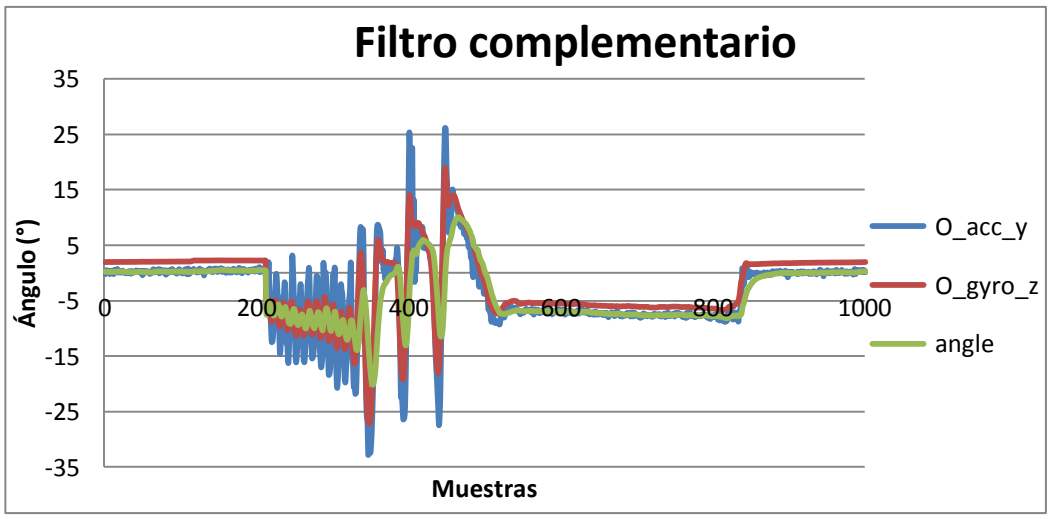



Figura 51. Datos obtenidos con el uso del filtro (tiempo de muestreo de 20 ms).

La imagen superior corresponde a un ajuste del filtro con valor de filtro paso alto 0.90. Se observan los valores arrojados por el acelerómetro en azul, el giróscopo en

rojo, y en verde la estimación del ángulo realizada. Son destacables los efectos que han sido anteriormente descritos y que pueden apreciarse:

- Seguimiento de los valores como referencia del acelerómetro y eliminación de esta forma de la deriva, se observa en los tramos de 0-200, 500-800 y 850-1000.
- Seguimiento del valor del giróscopo cuando el sensor se mueve, evitando todo el ruido propio de los acelerómetros. Tramo 200-500.
- La estimación introduce un retraso, es decir, es más lenta. Esto puede verse en los cambios bruscos. Tramo 350-500.

Comparación entre distintos valores de filtrado:

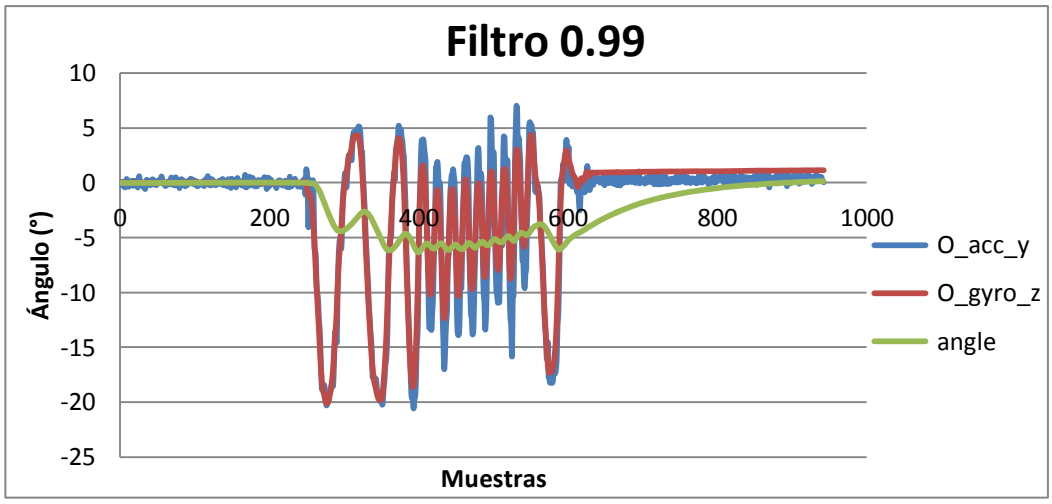



Figura 52. Filtro complementario 0.99 (tiempo de muestreo de 20 ms).

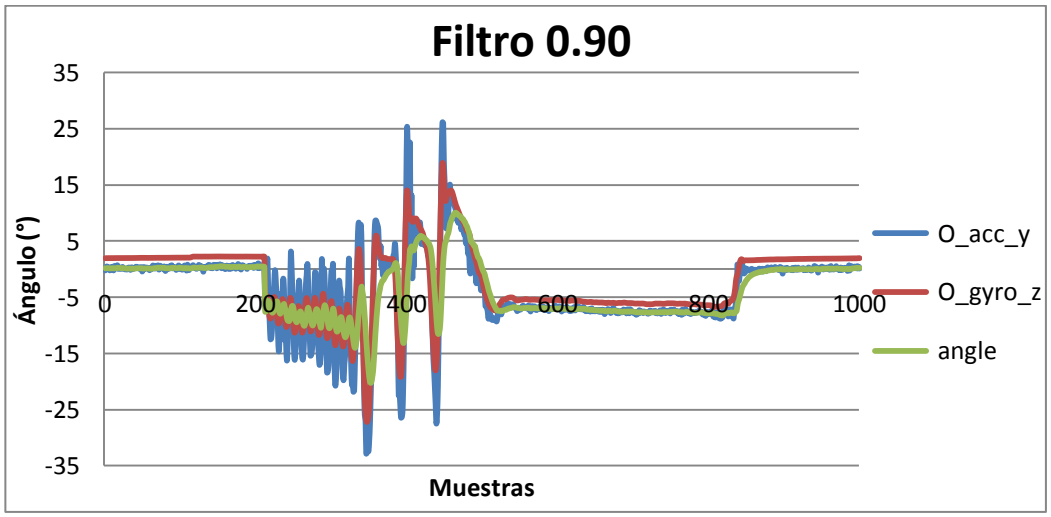



Figura 53. Filtro complementario 0.90 (tiempo de muestreo de 20 ms).

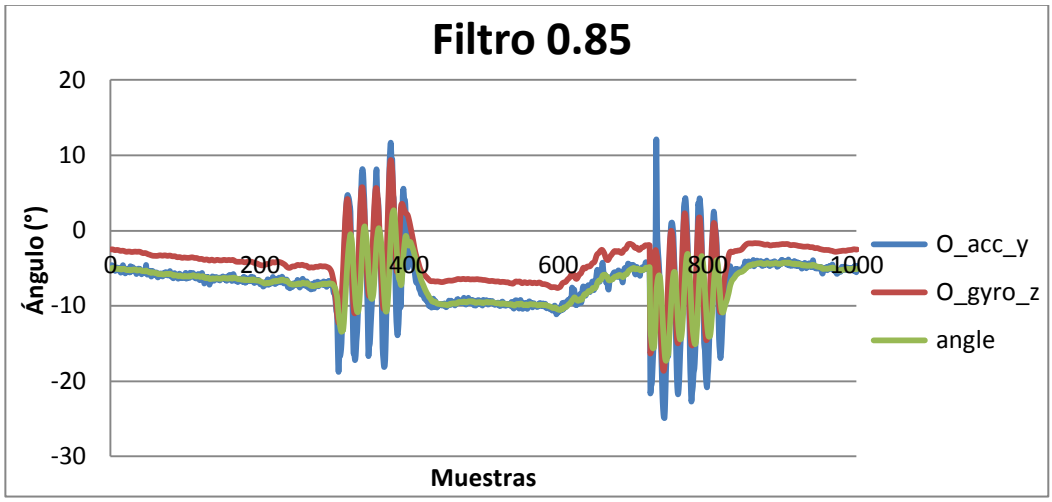



Figura 54. Filtro complementario 0.85 (tiempo de muestreo de 20 ms).

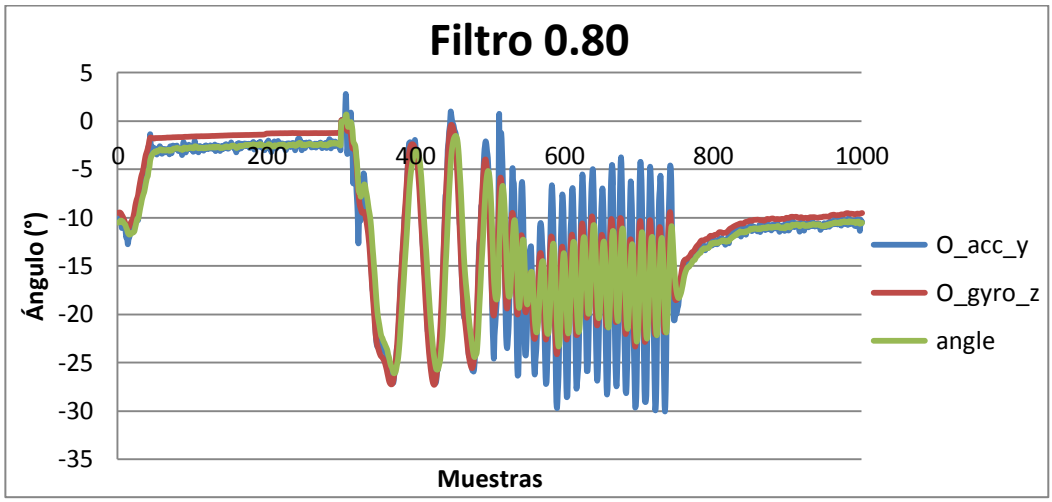



Figura 55. Filtro complementario 0.80 (tiempo de muestreo de 20 ms).

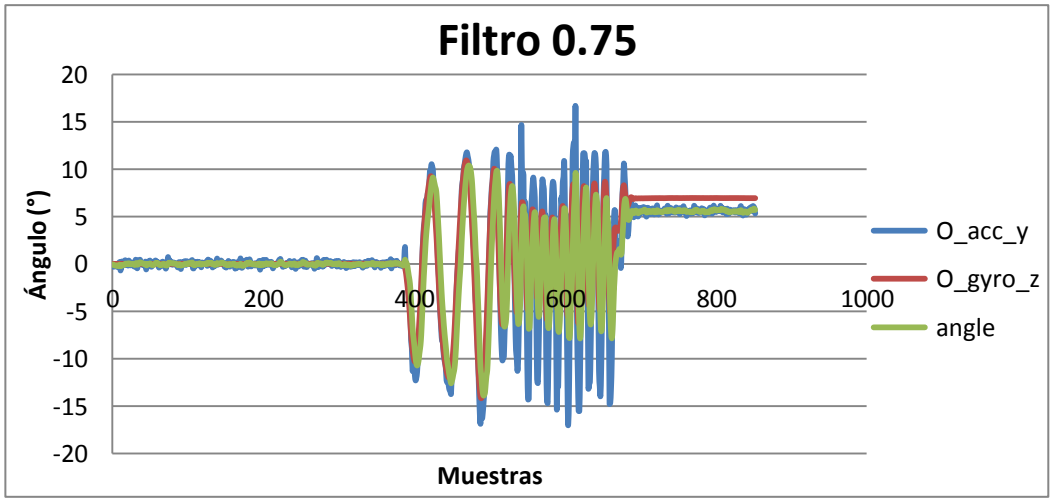



Figura 56. Filtro complementario 0.75 (tiempo de muestreo de 20 ms).

Conclusiones:

- Mientras que la corrección de la deriva se mantiene, y se evita la introducción de ruido, el descenso del valor del filtro paso alto tiene como consecuencia la aceptación, cada vez, de cambios más rápidos.
- La respuesta es muy lenta para valores altos, ya que prima la información proveniente del giróscopo, esto puede apreciarse de forma notable en la primera de las imágenes con un valor de 0.99.
- Conforme se desciende, la respuesta se agiliza debido al mayor peso de la información del acelerómetro. Un descenso excesivo puede conllevar que la señal que obtengamos sea de peor calidad al pasar por el filtro algo más de ruido de este sensor.
- El ajuste final de estos valores de realizará de forma experimental.

El código y uso de este algoritmo puede verse en el Anexo 6. También se han incluido las graficas a un tamaño mayor.

## 3.3. Algoritmo de control

Para la implementación del algoritmo del PID, no podía utilizarse simplemente la estructura teórica vista en la sección 2, puesto que nuestro sistema es altamente inestable debía ser un PID más avanzado [42] [43]. Para ello se adentró un poco más en el análisis de las debilidades de este tipo de esquema, y se observaron 2 puntos de actuación de especial interés:

- Corrección Windup: Surge en la parte integral del algoritmo PID, este efecto tiene como origen la limitación de los actuadores o su saturación. El denominado Windup se produce cuando el término integral, por su característico efecto memoria, alcanza valores muy altos debido a errores prolongados en el tiempo, que no pueden ser corregidos por las limitaciones de los actuadores del sistema. Conlleva que cuando el sistema vuelve a estar en valores de error menores, la actuación es muchísimo mayor a la debida por todo el error almacenado en la parte integral.

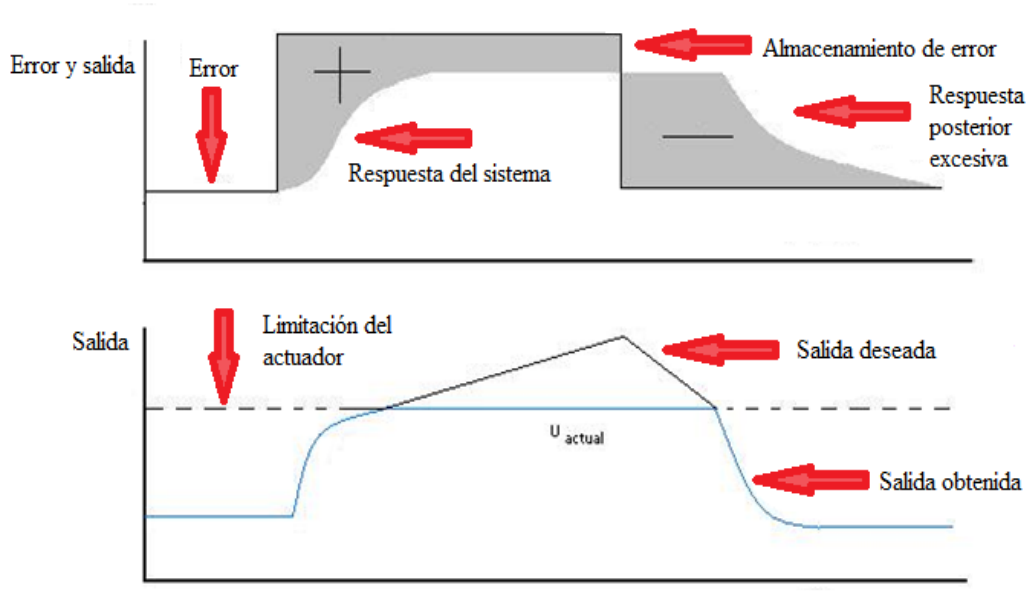



Figura 57. Windup. (Fuente: https://controls.engin.umich.edu/wiki/index.php/PIDDownside).

A la hora de corregir este efecto indeseado, se ha incluido en el código correspondiente a la implementación del PID, un par de condiciones con la que se eliminarían sus consecuencias. La primera afecta al error integral, que estará limitado a los valores de saturación de los actuadores, más concretamente al valor máximo del PWM. La segunda actúa en la salida de control, que también estará limitada al valor de saturación de los motores.

- Corrección derivative kick: este efecto proviene de la parte derivativa. La señal de error se obtiene a partir de la resta de la salida obtenida a la señal de referencia (error = referencia - salida). Esto puede acarrear problemas para el cálculo de la acción derivativa cuando el sistema se ve sometido a cambios bruscos en la referencia, como entradas de tipo escalón, ya que genera una señal de corrección muy fuerte. Un error grande, entre un intervalo de tiempo muy pequeño hace que tienda a infinito durante un muy breve periodo de tiempo. Estos picos producidos, tienen efectos muy negativos ya que provocan desgaste en los componentes mecánicos.

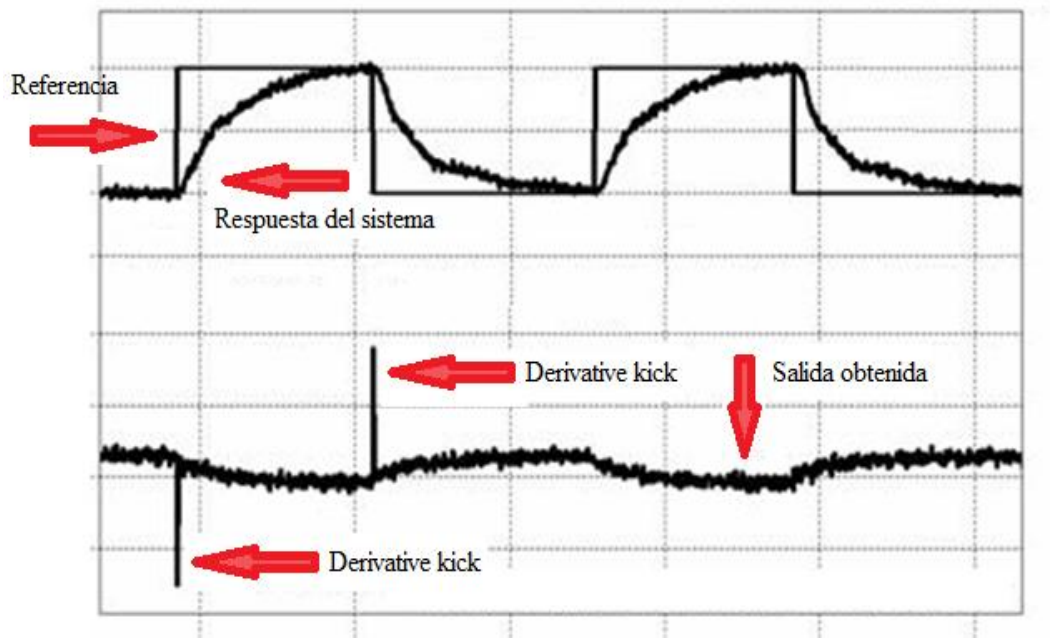



Figura 58. Derivative kick. (Fuente: http://www.controlguru.com/wp/p76.html).

Para eliminar el Derivative kick es suficiente con, a la hora de calcular el término derivativo, no utilizar la expresión típica que incluye referencia y salida, sino únicamente la salida. De esta forma se eliminan los cambios bruscos que introduce la referencia y se consigue mitigar este efecto.

El código de cómo quedo finalmente implementado el controlador PID puede verse en el Anexo 6.

## 3.4. Sistema de potencia.

El sistema de potencia está compuesto, como ya se ha visto, por las baterías, el driver y los motores. Para su inclusión en el sistema, primero se realizaron una serie de conexionados y pruebas.

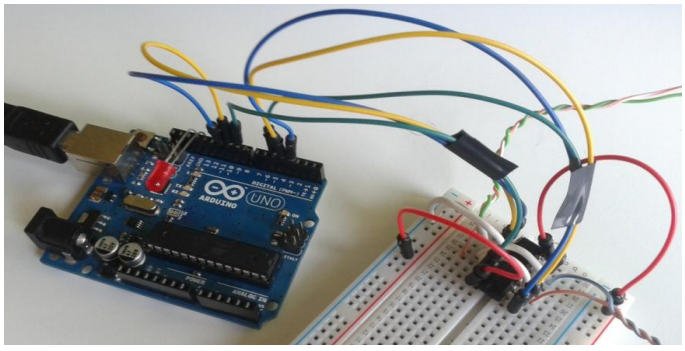


Figura 59. Conexionado del driver.

El centro de este sistema está en el driver, el encargado de suministrar la orden de actuación y la potencia a los motores. Se configuró de tal forma que el giro de motor pudiera ser invertible, y le es suministrada la señal de control, en nuestro caso PWM, por la patilla "enable". Así se consigue que cada puente sea gobernado de forma independiente, obteniendo un sistema diferencial. Esto se hizo con la idea de dotarlo de mayor versatilidad y, en un futuro, poder dirigir el robot y que pueda girar o pivotar sobre el eje vertical.

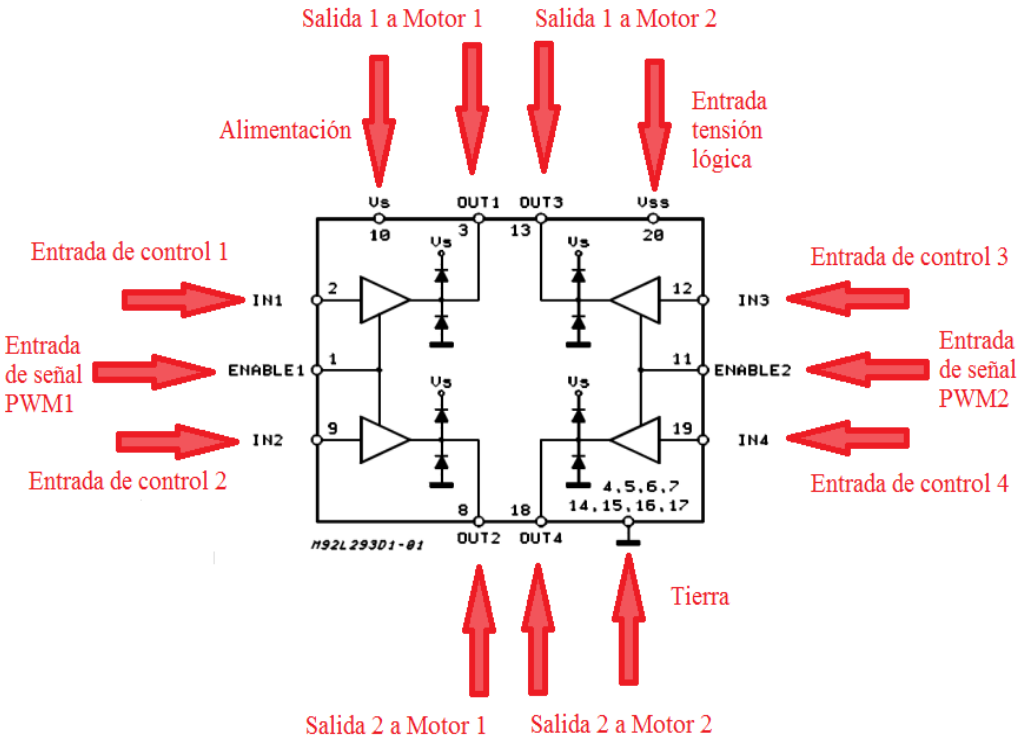



Figura 60. Patillaje del driver. (Fuente: L293D Datasheet).

Una versión final del programa que fue utilizado puede verse en el Anexo 8.

## 3.5. Diseño y elaboración del chasis.

Para la realización del soporte en el que incluir todos los componentes se siguió el siguiente proceso:

Se crearon dos tipos de modelo posible, que pueden verse en las imágenes inferiores. Las opciones provienen del estudio de la forma de varios robots autobalanceados existentes en Internet. Predominan las siguientes disposiciones:

- Estructura paralela: Los robots de este tipo presentan 2 piezas a los lados que actúan como soportes, están dispuestos de forma paralela al eje vertical y unidos mediante otras láminas perpendiculares. Los motores van situados junto con las ruedas en la parte inferior. Queda un espacio por encima de ellos en el que se situarían el resto de los componentes del robot. Generalmente este espacio superior suele estar compartimentado en distintos niveles con otras láminas más anchas y paralelas al plano horizontal, aunque en el modelo de la figura, esto fue modificado con el fin de reducir el peso de la estructura.

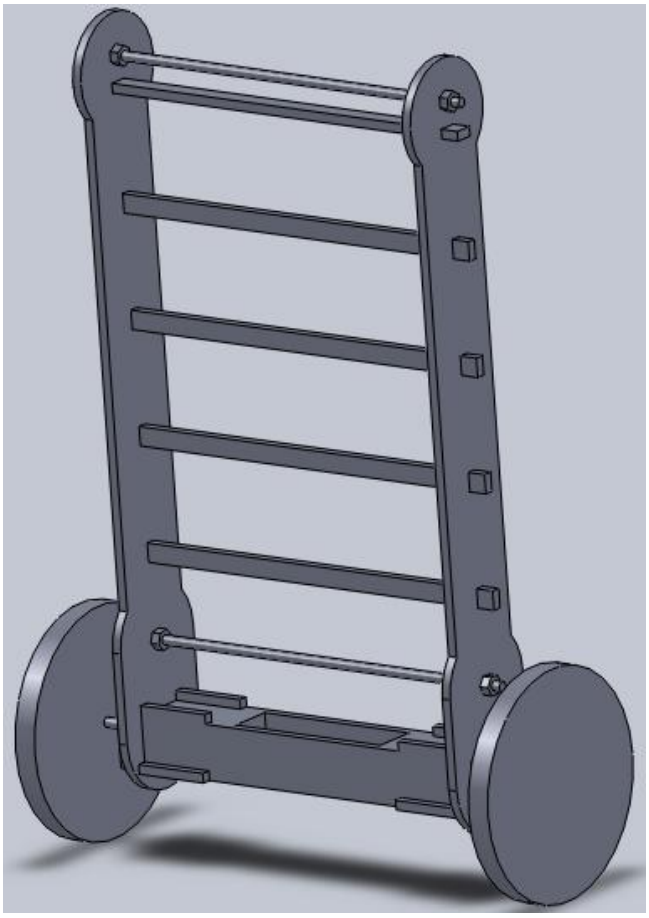


Figura 61. Boceto de robot con estructura paralela.

- Estructura planar: Presenta la misma configuración para motores y ruedas que quedan en la parte inferior. El resto del robot está dispuesto en una lámina paralela al eje vertical y al eje de balanceo. El resto de componentes se situarían anclados o adheridos a la superficie de dicho plano. Para dotar al

robot de una mayor versatilidad a la hora de situar las masas y conseguir que el centro de gravedad estuviera lo más cercano posible al eje vertical se hicieron los orificios que se aprecian. Esto también favoreció la ligereza de este tipo de chasis.

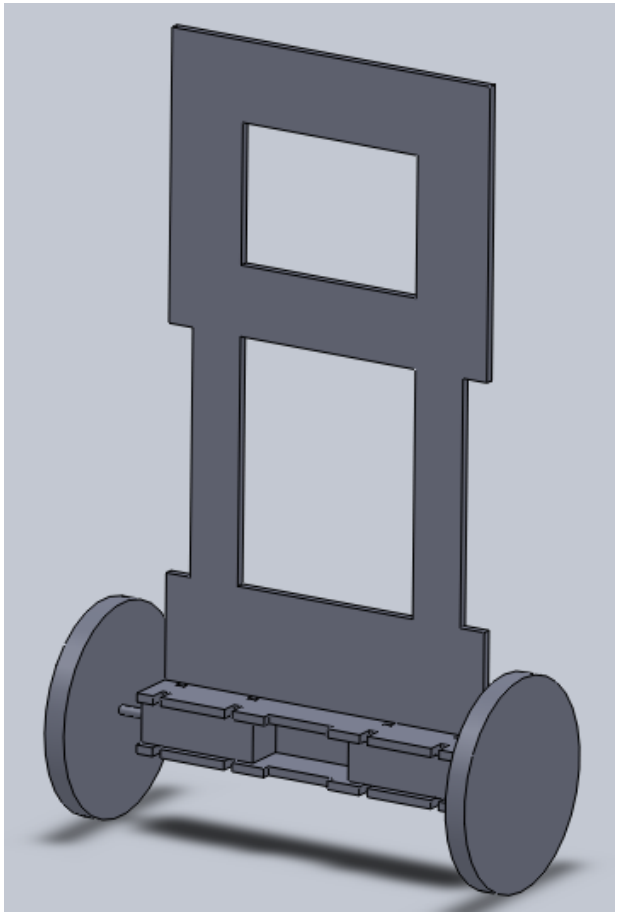


Figura 62. Boceto de robot con estructura planar.

Finalmente este fue el modelo elegido por su diseño sencillo, por presentar la posibilidad de ser aligerado de una forma fácil (haciendo los agujeros de mayor tamaño), y presentar la posibilidad de mover el centro de masas y poder obtener así un chasis más equilibrado.

Una vez establecido la forma que tendría nuestro robot, pasamos a la selección de materiales. Para primar la ligereza se pensó en elementos como la madera o algún tipo de polímero o plástico. Se realizaron diversas pruebas llegando a los siguientes resultados:

- Metacrilato: Se consideró este material ya que podría presentar unos resultados adecuados tanto en la parte mecánica como estética. Su trabajo conllevó grandes dificultades. En primer lugar si la lámina que se iba a emplear era demasiado fina, la estructura que se obtenía era endeble y con un pandeo notable, lo cual no era en absoluto aceptable. También se probó una lámina de grosor notablemente mayor, de esta se obtuvo una estructura que era lo suficientemente rígida, pero quebradiza a la hora de realizar taladros o aprietes. Además el trabajo de este tipo de material era problemático por la viruta generada, y porque, si el corte no era lo suficientemente cuidadoso, el compuesto terminaba por fundirse atrapando la herramienta. Finalmente la idea de emplear este material quedó rechazada.

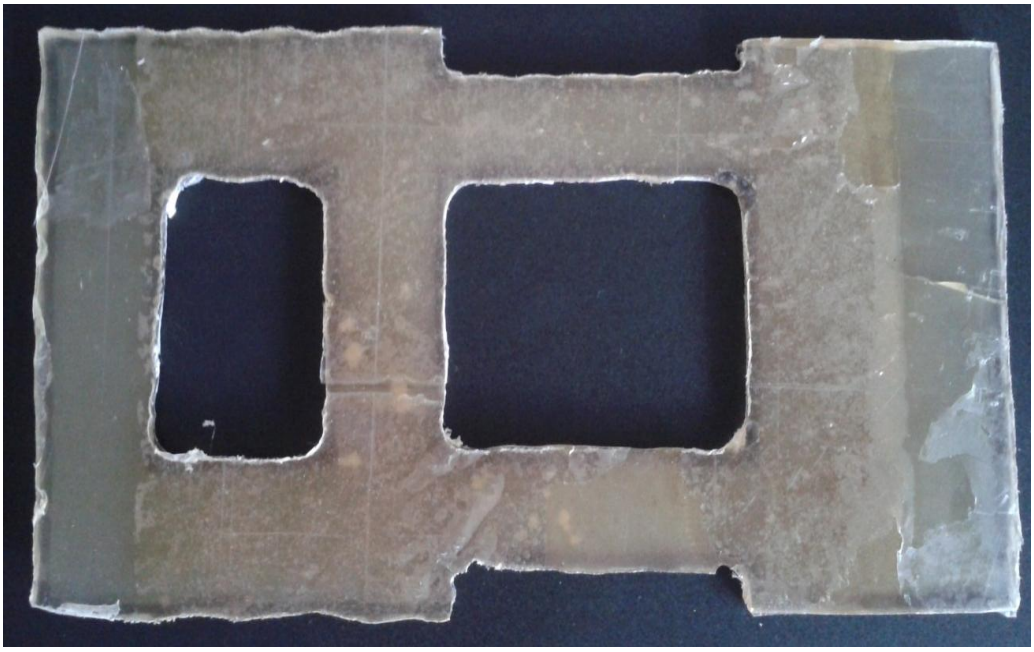


Figura 63. Estructura de metacrilato.

Figura 64. Detalle de las grietas.

- Chapa de madera Ocume: visto los problemas de excesiva flexibilidad del caso anterior, se optó por una chapa de 5 mm de grosor. Resultó ser un material más sencillo de trabajar y del que se obtuvieron buenos resultados, no presentando grietas a la hora de hacer taladros o agujeros. Aunque el resultado final no fue tan estéticamente llamativo como podría haber sido al utilizar el metacrilato, resultó ser aceptable y suficiente. Puede apreciarse en la imagen siguiente.

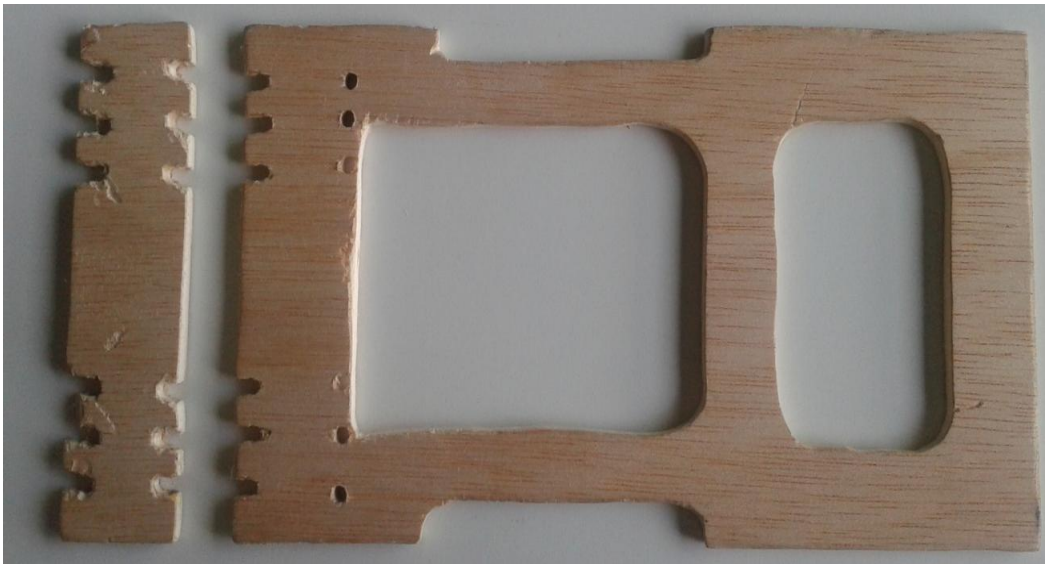


Figura 65. Estructura de chapa ocume.

### 3.6. Ensamblaje.

Una vez resueltos cada uno de los apartados que necesitaba el robot, fue el momento de ensamblarlo por completo. En la imagen inferior puede verse el resultado final.

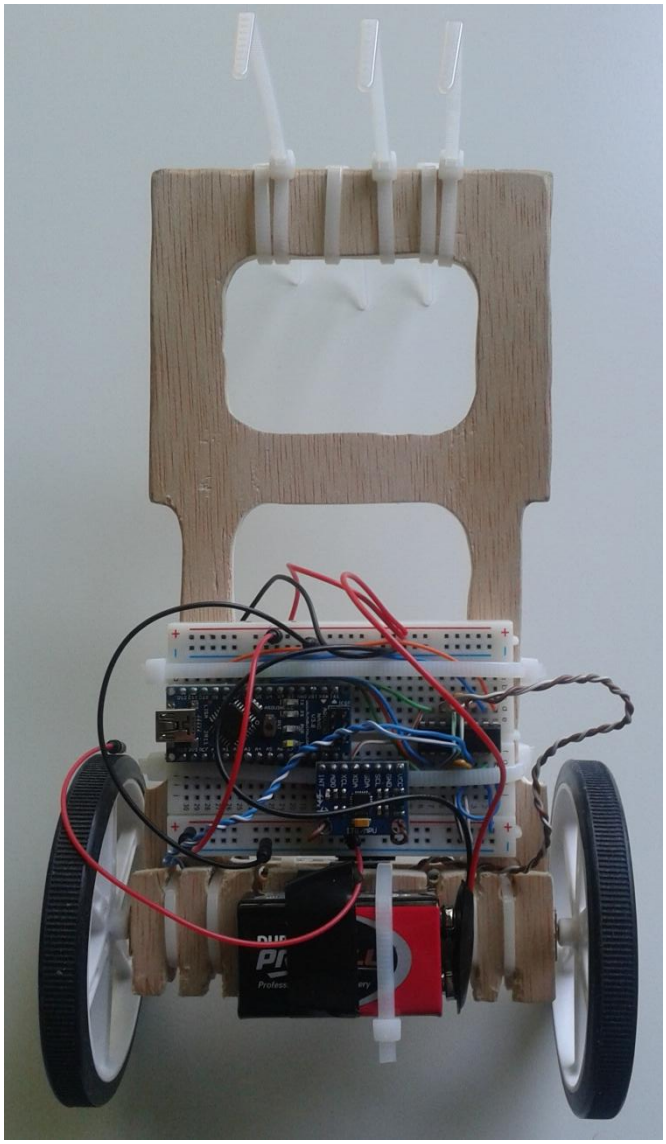


Figura 66. Robot completo.

Se ha incluido en el Anexo 9 una tabla con los pesos y las dimensiones de los componentes empleados.

También se ha incluido en el Anexo 10 el cálculo del centro de masas en las dos disposiciones diferentes que se utilizaron.

# 4. Ensayos.

## 4.1. Corrección en la estimación del ángulo.

Aunque en las pruebas hechas al sensor acelerómetro, se obtuvo como valores apropiados un rango que estaba entre 0.85 y 0.88, a la hora de buscar las constantes para sintonizar el PID, se comprobó que la respuesta que se obtenía era muy brusca e irregular. Esto no era algo que pudiera provenir del PID puesto que estaba todavía configurado simplemente como un control proporcional, con lo que la respuesta debiera haber sido lineal. Se pasó entonces a comprobar la configuración del filtro complementario.

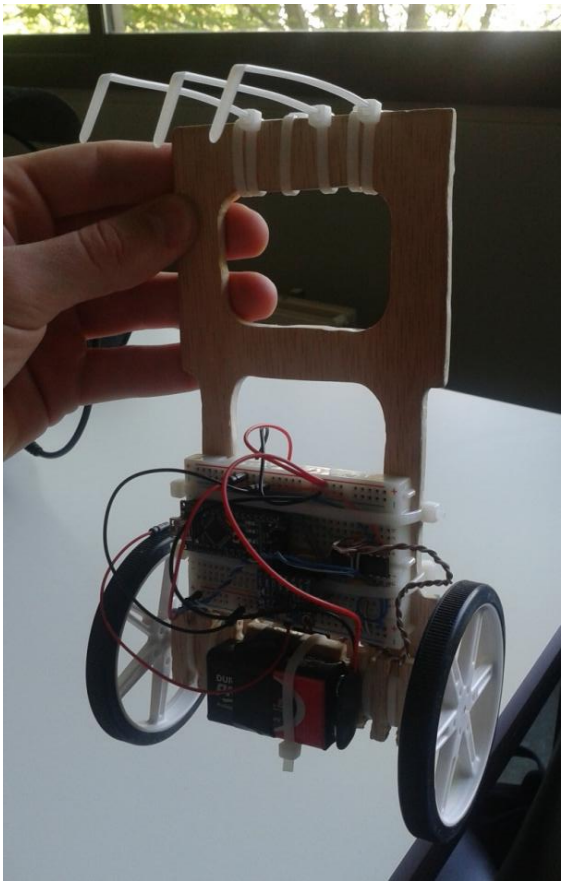


Figura 67. Test 1.

Se testeó y vio que los valores obtenidos no eran adecuados, ya que se introducía mucho ruido procedente del sensor acelerómetro, que era el causante de esa actuación tan irregular. Experimentalmente se comprobó que el valor apropiado era de 0.92, ya que con él desaparecían los cambios bruscos y la respuesta seguía siendo rápida.

Se terminó de comprobar que el ángulo obtenido era adecuado haciendo que parara al llegar a los 90º, con este ultimo complemento se dio por validos los datos obtenidos y fue esta la configuración que finalmente se utilizó.

## 4.2. Sintonización del PID.

La primera opción, a tenor de lo visto en otros casos de robots autobalanceados, fue intentar realizar el ensayo de "Ziegler - Nichols". Este método se basa en un ajuste empírico que parte de la búsqueda de la constante proporcional crítica, esta se caracteriza por generar un estado de oscilación constante en el sistema ante una entrada de tipo escalón. A partir de los datos obtenidos, pueden hallarse los valores para el resto de constantes ya que estos se encuentran tabulados.

Las siguientes imágenes corresponden a los resultados del ensayo de "Ziegler – Nichols" ante una entrada escalón de 5 grados.

En el primer caso, para K = 25, se observa como la respuesta es estable, se logra mantener el péndulo en los 5 grados que corresponden al valor de la entrada escalón.

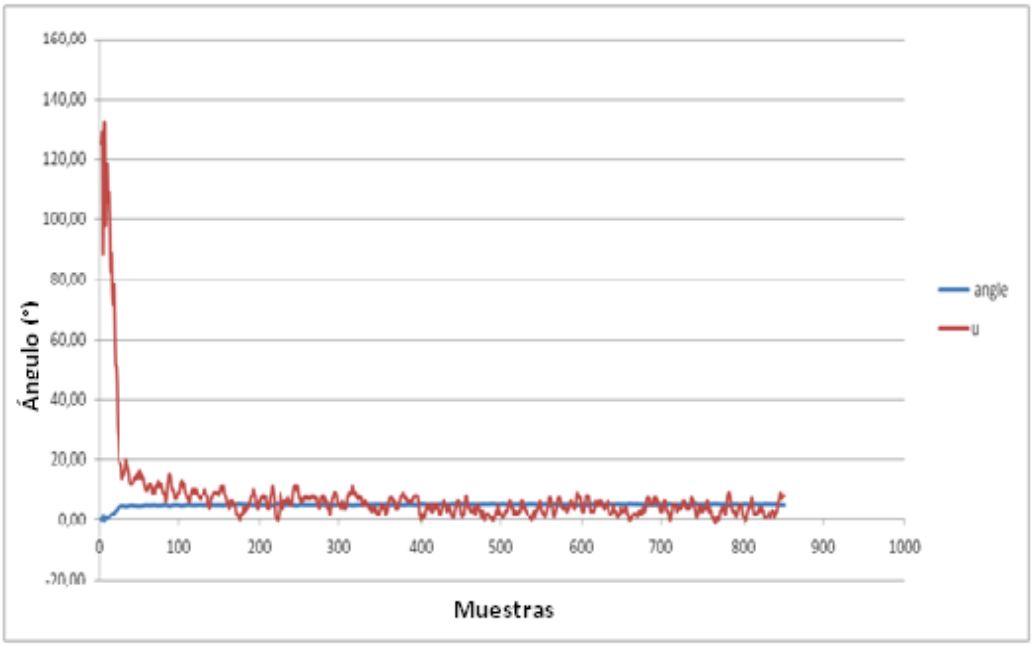



Figura 68. Resultado del ensayo para K = 25.

El siguiente caso, con K = 26, se observa una respuesta totalmente inestable, en la que se aprecia claramente la saturación de los motores además de que no se alcanza el ángulo deseado.

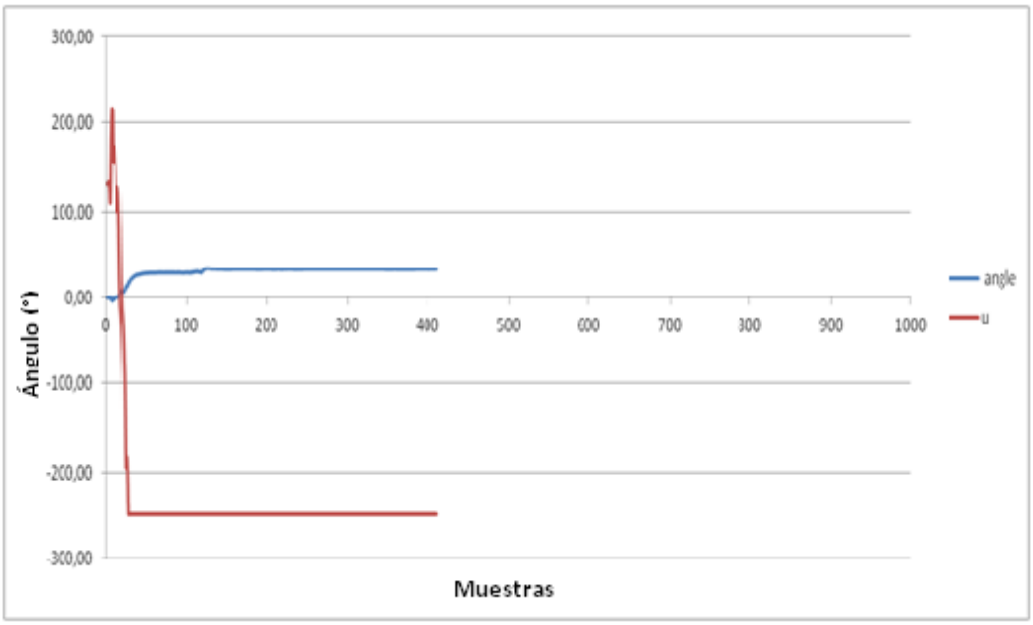



Figura 69. Resultado del ensayo para K = 26.

Con K = 25.5, se observa también una respuesta inestable, en la que se encuentra como en el caso anterior, no linealidades propias de la saturación de los motores.

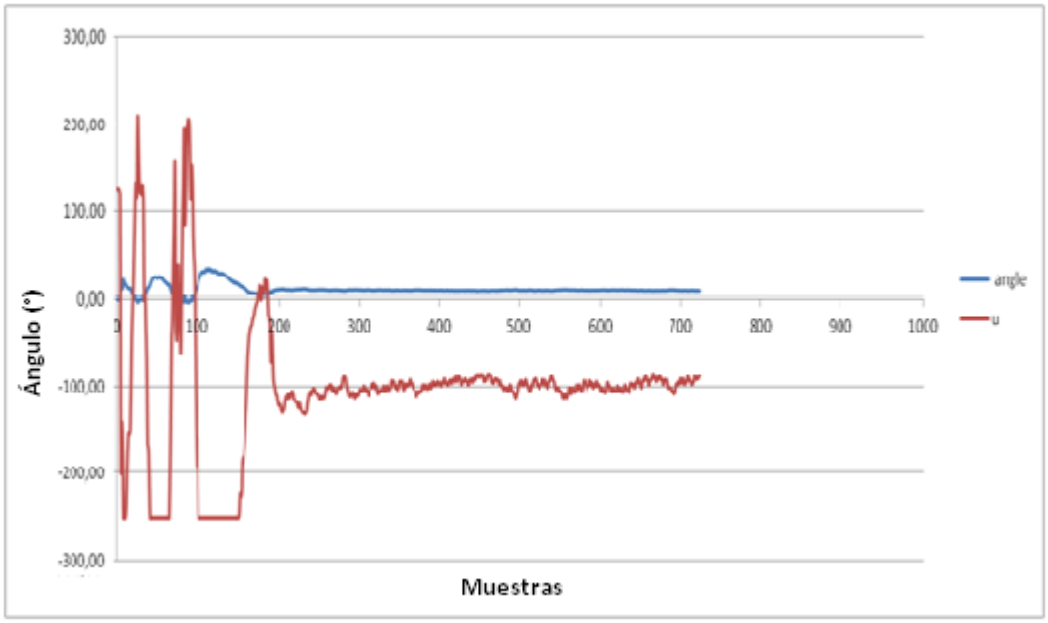



Figura 70. Resultado del ensayo para K = 25.5.

Tras la realización de muchas pruebas con distintos valores, fue imposible localizar el valor de dicha constante ya que, o la acción no era lo suficiente fuerte, o el

sistema se hacía completamente inestable. Valores cercanos o intercalados entre estos dos comentados tampoco arrojaron resultados útiles.

Consultando los ensayos hechos por otros autores, se vio que el ajuste que ellos realizaban era el mismo, ajustando de forma experimental los valores finales. De todo esto se obtuvo por lo tanto una respuesta nada concluyente.

Como lo que se busca es una combinación de tres términos, el intentar conseguirlo por simple ensayo y error habría supuesto una enorme inversión de tiempo, por lo que se intentó el control de un sistema similar pero estable, el péndulo normal.

Se valoró positivamente la realización de este ensayo además, porque permitiría de una forma sencilla comprobar la correcta implementación del algoritmo de control.

Con el péndulo preparado para medir en posición normal, es decir, con el centro de masas por debajo del punto de balanceo, se procedió nuevamente a realizar el ensayo de "Ziegler – Nichols". Las siguientes imágenes muestran las respuestas obtenidas en la búsqueda de la constante de oscilación crítica.

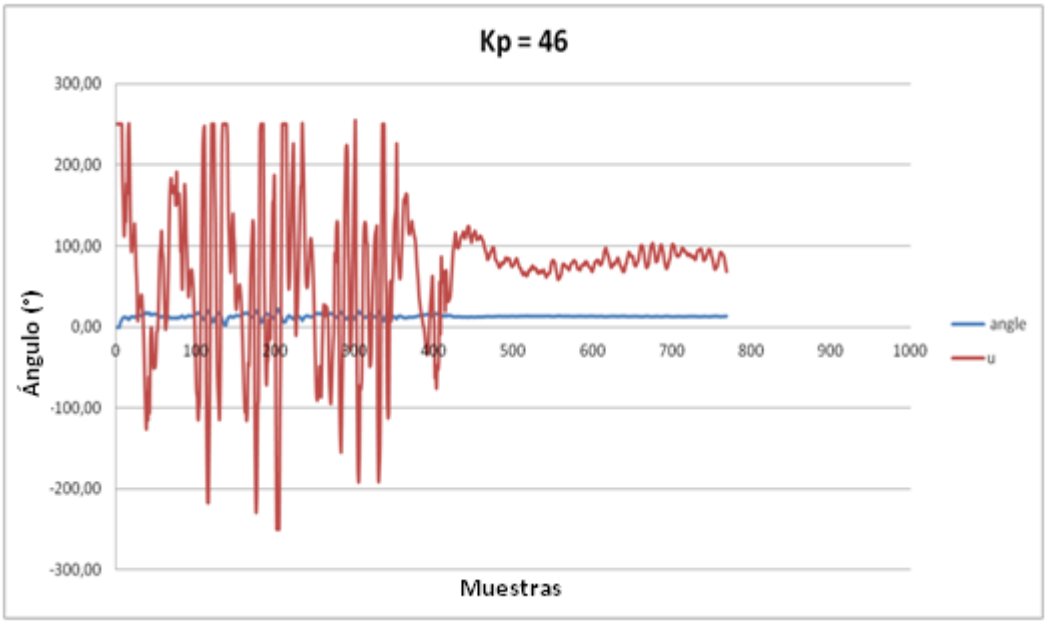



Figura 71. Respuesta para K = 46.

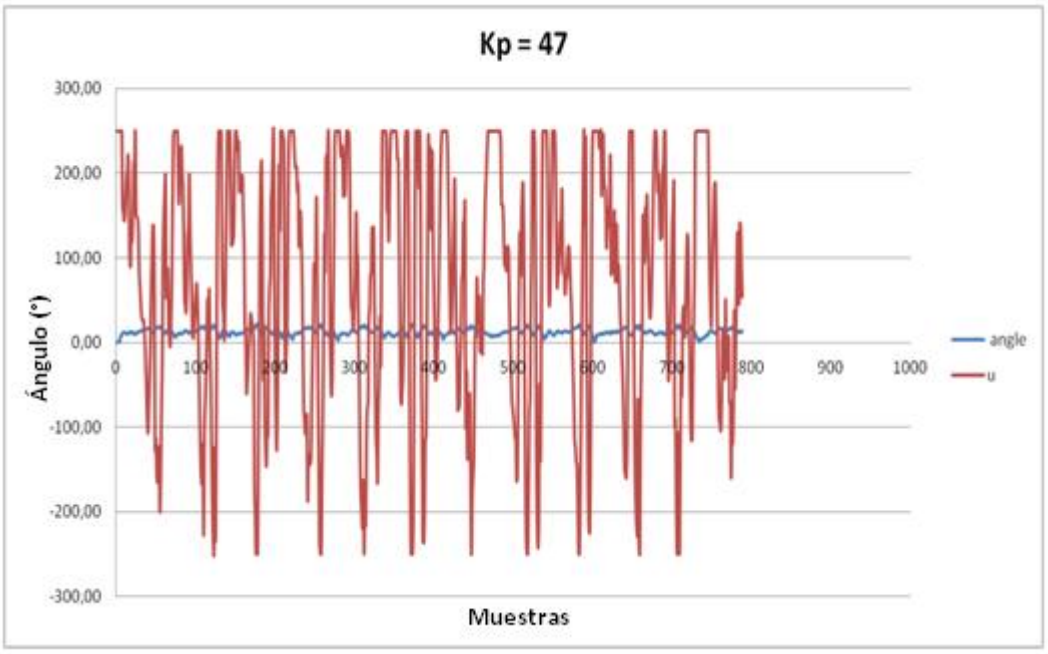



Figura 72. Respuesta para K = 47.

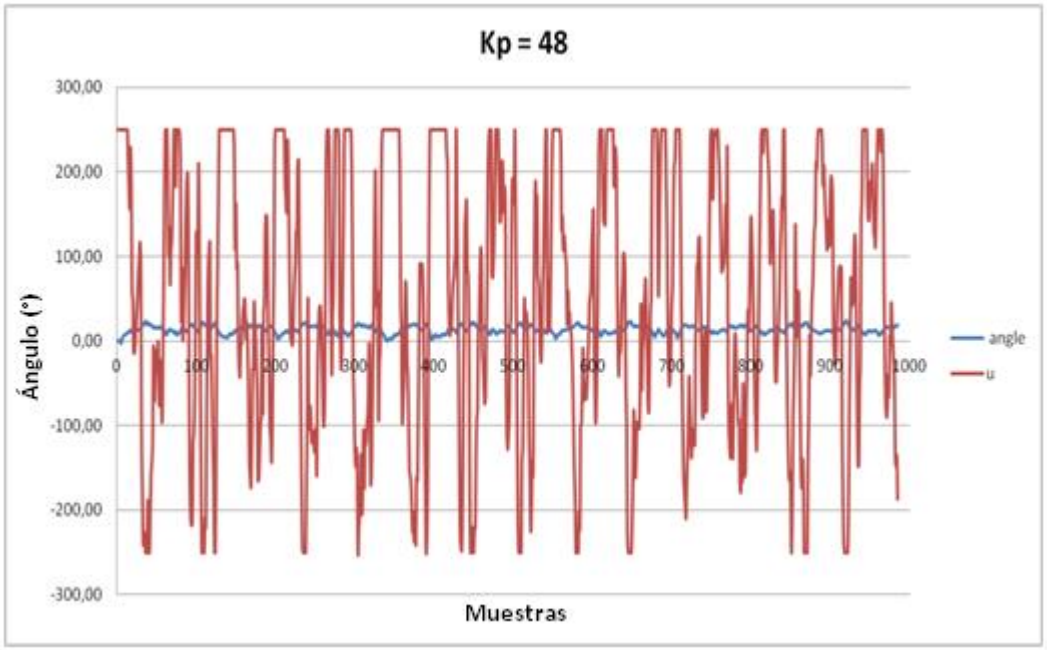



Figura 73. Respuesta para K = 48.

Aunque la respuesta no fue todo lo deseable que cabría por las no linealidades que se aprecian, hay que destacar que son notablemente menores al caso del ensayo con el péndulo invertido. Corresponden como en el apartado anterior, a la saturación de los motores (valor de saturación: 250).

Tras obtener los valores necesarios correspondientes a las constantes proporcional (Kp = 23.5), integral (Ki = 7.2) y derivativa (Kp = 1.8), se obtuvo la respuesta que se muestra debajo, que corresponde a una que sería bastante aproximada a la deseable, y que confirma que el algoritmo es correcto.

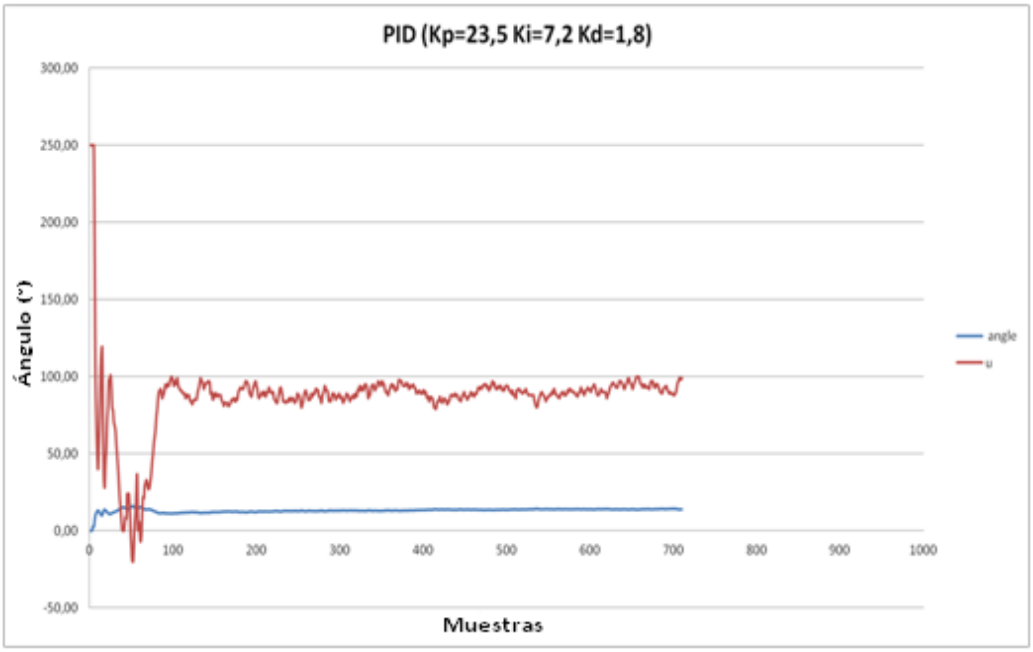



Figura 74. Resultado PID con el péndulo normal.

Pese a los prometedores datos obtenidos, a la hora de trasladarlos al problema del péndulo invertido, los resultados volvieron a ser nada satisfactorios. También se intentó ajustar las constantes de una forma más experimental, pero sin lograr cambios significativos.

Con el PID bien programado y dando buenas respuestas, pero con resultados todavía no deseables, se optó por buscar algún tipo de solución más extrema. Esto se consiguió a base de introducir no linealidades y ajustar en función de ellas alguna de las constantes.

Viendo por ejemplo que, para una constante proporcional adecuada para desviaciones grandes, ante pequeñas oscilaciones en el error, la respuesta era excesiva, se optó por no actuar entre valores de 1 y 1.5º desviación. Con esto se perdía la opción de que el péndulo permaneciera sin oscilar y sin desplazarse sobre la superficie, pero

mejoró notablemente en el equilibrio, que presentaba un pequeño balanceo o cabeceo, aunque seguía terminando por caer.

Para evitar esto, también se ajustó el término integral, que hiciera que la actuación fuera mayor ante errores sostenidos, y se complementó con otra condición no lineal. Esta imponía que a partir de 10º de desviación la actuación de los motores fuera máxima. Con lo que el robot ya permanecía por largos periodos en posición vertical.

Por último se terminó de ajustar la parte derivativa de forma que el cabeceo se redujera lo máximo que fuera posible, aunque sin terminar de eliminarlo por completo.

Finalmente fue esta la configuración que se adoptó, una mezcla entre el algoritmo PID clásico, con ciertas no linealidades impuestas para casos puntuales de error.

# 5. Presupuesto.

Coste del prototipo final y de 100 unidades, incluye únicamente el coste de los materiales empleados.

| PRESUPUESTO PROTOTIPO FINAL Y COSTE DE 100 UDS | | | | | | |
|---|---|---|---|---|---|---|
| Apartado | Componente | Precio Ud. (€) | Uds. | Precio prototipo (€) | Pecio 100 Uds. (€) | Precio 100 robots (€) |
| Sensor | | | | | | |
| | Acelerómetro Giróscopo MPU6050 | 7,52 | 1 | 7,52 | 6,59 | 659 |
| Microcontrolador | | | | | | |
| | Arduino Nano | 7,95 | 1 | 7,95 | 6,99 | 699 |
| Potencia | | | | | | |
| | Driver L293D | 2,69 | 1 | 2,69 | 1,75 | 175 |
| | Motor | 5,13 | 2 | 10,26 | 4,56 | 912 |
| Alimentación | | | | | | |
| | Pila recargable 9V | 9,08 | 2 | 18,16 | 9,08 | 1816 |
| PCB | | | | | | |
| | Placa de circuito impreso | 30,1 | 1 | 30,1 | 6,89 | 689 |
| Ruedas | | | | | | |
| | 2x Pololu 80x10mm | 9,56 | 1 | 9,56 | 9,56 | 956 |
| Chasis | | | | | | |
| | Chapa de madera | 0,75 | 1 | 0,75 | 0,5 | 50 |
| Otros | | | | | | |
| | Condensador cerámico | 0,05 | 2 | 0,1 | 0,02 | 4 |
| | Conector pila 9V | 0,4 | 2 | 0,8 | 0,32 | 64 |
| | Cable | 0,15 | 4 | 0,6 | 0,1 | 40 |
| TOTAL (€) | | | | 88,49 | | 6064 |

Figura 75. Presupuesto prototipo final y coste 100 uds.

El presupuesto final del prototipo asciende a OCHENTA Y OCHO EUROS CON CUARENTA Y NUEVE CÉNTIMOS, y el correspondiente a las 100 unidades son SEIS MIL SESENTA Y CUATRO EUROS.

El presupuesto total, correspondiente al coste del proyecto incluye, el coste de los materiales empleados para su construcción y ensayos, las herramientas que fueron necesarias para la fabricación y la mano de obra.

| **PRESUPUESTO TOTAL** | | | | |
|---|---|---|---|---|
| **Apartado** | Componente | Precio Ud. | Uds. | Precio |
| **MATERIAL** | | | | |
| **Sensor** | | | | |
| | Acelerómetro + Giróscopo MPU6050 | 7,52 | 1 | 7,52 |
| **Microcontrolador** | | | | |
| | Arduino Nano | 7,95 | 1 | 7,95 |
| | Arduino Uno | 18,9 | 1 | 18,9 |
| **Potencia** | | | | |
| | Driver L293D | 2,69 | 1 | 2,69 |
| | Motor | 5,13 | 2 | 10,26 |
| **Alimentación** | | | | |
| | Pila recargable 9V | 9,08 | 2 | 18,16 |
| **Protoboard** | | | | |
| | Protoboard | 2,1 | 1 | 2,1 |
| **Ruedas** | | | | |
| | 2x Pololu 80x10mm | 9,56 | 1 | 9,56 |
| **Chasis** | | | | |
| | Chapa de madera | 0,75 | 1 | 0,75 |
| | Metacrilato | 1,95 | 1 | 1,95 |
| **Otros** | | | | |
| | Condensador cerámico | 0,05 | 2 | 0,1 |
| | Conector pila 9V | 0,4 | 2 | 0,8 |
| | Cable | 0,15 | 4 | 0,6 |
| | Cable M/H Dupont | 0,05 | 20 | 1 |
| | Cable M/M | 0,05 | 12 | 0,6 |
| | Led rojo | 0,07 | 2 | 0,14 |
| | Bridas | 2,35 | 0,8 | 1,88 |
| | Cinta aislante | 1,1 | 0,1 | 0,11 |
| **HERRAMIENTAS** | | | | |
| **Mini-herramienta** | | | | |
| | Mini-herramienta | 22,4 | 1 | 22,4 |
| **Soldador** | | | | |
| | Soldador | 10,1 | 1 | 10,1 |
| **Estaño** | | | | |
| | Bobina estaño | 12 | 0,001 | 0,012 |

| | | | | |
|---|---|---|---|---|
| **Soporte** | | | | |
| | Tercera Mano | 7 | 1 | 7 |
| **Cargador** | | | | |
| | Cargador de baterías | 18,85 | 1 | 18,85 |
| **Polímetro** | | | | |
| | Polímetro | 18,2 | 1 | 18,2 |
| **MANO DE OBRA** | | | | |
| | Sueldo Ingeniero/mes | 1350 | 2 | 2700 |
| **SUBTOTAL** | | | | 2861,63 |
| | IVA(21%) | | | 600,94 |
| **TOTAL** | | | | 3462,09 |

Figura 76. Presupuesto total.

El presupuesto final del proyecto asciende a TRES MIL CUATROCIENTOS SESENTA Y DOS EUROS CON NUEVE CÉNTIMOS.

Datos a destacar del coste total serían:

- El 21% corresponde a impuestos, siendo el importe correspondiente al IVA de 600,94€.
- El 78% del total es directamente mano de obra, 2700 €.
- El coste del prototipo es de 88,5€, en caso de fabricarse 100 Uds. el precio de cada robot bajaría hasta los 60,64€.

# 6. Conclusiones y línea de futuros.

## 6.1. Conclusiones.

La elaboración de este proyecto ha supuesto una manera muy interesante y entretenida de conjugar en un único elemento, multitud de disciplinas vistas durante el grado, principalmente aquellas relacionadas con la electrónica. Además, la creación de este tipo de pequeños robots auto-suficientes supone una grata recompensa a la hora de afrontar un proyecto como los de final de grado, por lo que serían muy recomendables.

Este trabajo ha servido además como una primera toma de contacto para la realización de aplicaciones en las que personalmente estoy muy interesado, que son aquellas que requieren de la creación de alguna clase de dispositivo electrónico para ser realizadas. Dentro de estas aplicaciones destacarían los siguientes puntos tratados en el proyecto y que han resultado de especial interés:

- Selección de componentes: abarcaría el proceso que va desde el primer análisis de necesidades, el análisis de múltiples opciones, la consulta de hojas de características y la selección final. Se asemejaría este proceso al seguido para el segundo apartado de este proyecto.
- Diseño de sistemas: comprende el conexionado y la creación de los enlaces necesarios para conformar el dispositivo en su forma conjunta. Incluye campos como la creación de circuitos, adecuación de las conexiones y el estudio de las señales y sus modos de transmisión. Destacar de este punto la creación mediante Design Spark del prototipo de la PCB.
- Creación de prototipos: el llevar el sistema diseñado a su implementación física supone en primer lugar un reto importante, ya que muchas de las cosas perfectamente posibles en la teoría, pueden obtenerse resultados nada positivos e incluso frustrantes. Este apartado también supone un interesantísimo banco de pruebas a la hora de añadir mejoras en cuanto a prestaciones, diseño o robustez del producto entre otras.
- Comunicación entre dispositivos: merece un apartado puesto que era el tema menos conocido dentro de las necesidades del proyecto, pero que presentaba una importancia enorme. Se abordo el análisis de los tipos de comunicación

existentes con especial determinación en el que correspondía a la comunicación de tipo I2C, necesaria para obtener datos del MPU6050.

- Programación: Una vez diseñado y creado el prototipo es necesario la configuración del mismo para que actúe según lo esperado. Aunque en ocasiones puede ser una tarea algo pesada, la realización de programas en Arduino ha sido muy ágil pese a que el conocimiento sobre este sistema era escaso.
- Control de sistemas: La elaboración de un algoritmo y ver los resultados obtenidos, sobre todo a la hora de realizar el ensayo con el PID sobre el péndulo en posición normal, supuso también un refuerzo muy positivo en esta materia ya que el control siempre había resultado ser algo mayormente teórico y abstracto.
- Intangibles: Se incluyen en este apartado todos esos factores derivados del proyecto pero que no forman parte de él, aquellos como el enfrentarse a problemas, la solución de imprevistos, o el sobreponerse a resultados inesperadamente negativos.

## 6.2. Línea de futuros.

Algunas mejoras posibles para este robot serían:

- Estimación del ángulo mediante filtro Kalman. Con este sistema se ganaría algo de rapidez a la hora de obtener el ángulo ya que el filtro complementario puede resultar ser algo lento para aplicaciones como esta.
- Implementación de otros algoritmos de control, como podrían ser control de estados o lógica difusa.
- Elaboración de otro tipo de chasis, con alternativas como la estructura paralela o fabricarlo en plástico mediante impresora 3D.
- Añadir una conexión inalámbrica para poder enviar la telemetría a otro dispositivo. Podría utilizarse radio frecuencia de 433MHz o algún modulo bluetooth para ello.
- Podrían también utilizarse la conexión inalámbrica para dirigir el robot y que se mueva siguiendo el recorrido deseado por el usuario.
- Otros tipos de baterías, podrían utilizarse las de polímero de litio (LiPo).

# 7. Bibliografía.

Robots autobalanceados:

[1] http://madebyfrutos.wordpress.com/2013/05/02/vertibot/

[2] http://letsmakerobots.com/node/38610

[3] http://letsmakerobots.com/node/1505

Péndulo invertido:

[4] http://en.wikipedia.org/wiki/Inverted_pendulum

[5] http://iimyo.forja.rediris.es/invpend/invpend.html

[6] http://www.ib.cnea.gov.ar/~instyctl/Tutorial_Matlab_esp/invpen.html

[7] http://eprints.ucm.es/16096/1/memoriaPFC.pdf

[8] Software PID Control of an Inverted Pendulum Using the PIC16F684

Sensor:

[9] http://www.sensing.es/

[10] http://es.wikipedia.org/wiki/Potenci%C3%B3metro

[11] http://en.wikipedia.org/wiki/Rotary_variable_differential_transformer

[12] http://es.wikipedia.org/wiki/Codificador_rotatorio

[13] http://es.wikipedia.org/wiki/Aceler%C3%B3metro

[14] http://es.wikipedia.org/wiki/Gir%C3%B3scopo

[15] https://www.sparkfun.com/categories/160

[16] http://es.wikipedia.org/wiki/Unidad_de_medici%C3%B3n_inercial

[17] http://5hertz.com/tutoriales/?p=228

[18] http://5hertz.com/tutoriales/?p=431

Microcontrolador:

[19] http://es.wikipedia.org/wiki/Microcontrolador

[20] Apuntes de la asignatura Electrónica Industrial cursada en UPNA, Carlos Ruiz y Patxi Arregui.

[21] http://es.rs-online.com/web/

[22] http://www.arduino.cc/en/

Control:

[23] http://es.wikipedia.org/wiki/PID

[24] Apuntes de la asignatura Ingeniería de Control cursada en UPNA, Iñaki Arocena.

Potencia:

[25] http://www.monografias.com/trabajos60/servo-motores/servo-motores.shtml

[26] http://es.wikipedia.org/wiki/Servomotor

[27] http://www.todorobot.com.ar/informacion/tutorial%20stepper/stepper-tutorial.htm

[28] http://es.wikipedia.org/wiki/Motor_paso_a_paso

[29] http://es.wikipedia.org/wiki/Motor_de_corriente_continua

[30] http://es.wikipedia.org/wiki/Motor_el%C3%A9ctrico

[31] http://es.wikipedia.org/wiki/Puente_H_(electr%C3%B3nica

[32] http://www.st.com/web/en/resource/technical/document/datasheet/CD00002346.pdf

[33] https://www.sparkfun.com/datasheets/Robotics/L298_H_Bridge.pdf

[34] http://users.ece.utexas.edu/~valvano/Datasheets/L293d.pdf

PCB:

[35] Apuntes de la asignatura Fabricación y Ensayo de Equipos Electrónicos cursada en UPNA, Jesús Corrés.

Primeros pasos con Arduino:

[36] http://www.arduino.cc/en/

Obtención de datos con MPU6050

[37] MPU 6050 Datasheet

[38] http://www.quadruino.com/guia-2/sensores/protocolo-i2c-twi

[39] http://es.wikipedia.org/wiki/I%C2%B2C

[40] Apuntes Sistemas en adquisición en interfaz impartida en la UPNA, Javier Goicoechea

[41] Shane Colton. A Simple solution for Balance Filter. MIT. June 2007

Algoritmo de control

[42] http://brettbeauregard.com/blog/2011/04/improving-the-beginners-pid-introduction/

[43] Apuntes de la asignatura Ingeniería de Control cursada en UPNA, Iñaki Arocena.

# 8. Índice de figuras.

# 9. Anexos.

## 9.1. Anexo 1. Programa para la obtención de datos MPU6050.

```
//--------------------------------------------------------
//
//          ROBOT AUTOBALANCEADO
//
//--------------------------------------------------------
// Para la comunicación con el sensor via bus I2C
// se incluye la libreria Wire.h
//
#include <Wire.h>

// Los siguientes registros que se declaran tienen el nombre de acuerdo a:
//                      "MPU-6000 and MPU-6050 Register Map
//                      and Descriptions Revision 3.2".
//

// REGISTROS:
//

// GYRO_CONFIG Register   |bit_7|bit_6|bit_5|bit_4 |bit_3 |bit_2|bit_1|bit_0|
//                      |XG_ST|YG_ST|ZG_ST| FS_SEL[1:0] | --- | --- | --- |
// XG_ST, YG_ST, ZG_ST son bits de selftest.
// FS_SEL configura la sensibilidad del giróscopo.(bits 4&3)
// FS_SEL | Full Scale Range
//   0    |    250º/s
//   1    |    500º/s
//   2    |   1000º/s
//   3    |   2000º/s
//
#define MPU6050_GYRO_CONFIG        0x1B

// ACCEL_CONFIG Register  |bit_7|bit_6|bit_5|bit_4 |bit_3 |bit_2|bit_1|bit_0|
//                      |XA_ST|YA_ST|ZA_ST|AFS_SEL[1:0] | --- | --- | --- |
// XA_ST, YA_ST, ZA_ST son bits de selftest.
// AFS_SEL configura la sensibilidad del acelerómetro. (bits 4&3)
// AFS_SEL | Full Scale Range
//   0     |    2g
//   1     |    4g
//   2     |    8g
//   3     |   16g
//
#define MPU6050_ACCEL_CONFIG       0x1C
```

```
// Registros de datos de acelerómetro y giróscopo
//
#define MPU6050_ACCEL_XOUT_H       0x3B
#define MPU6050_ACCEL_XOUT_L       0x3C
#define MPU6050_ACCEL_YOUT_H       0x3D
#define MPU6050_ACCEL_YOUT_L       0x3E
#define MPU6050_ACCEL_ZOUT_H       0x3F
#define MPU6050_ACCEL_ZOUT_L       0x40
#define MPU6050_GYRO_XOUT_H        0x43
#define MPU6050_GYRO_XOUT_L        0x44
#define MPU6050_GYRO_YOUT_H        0x45
#define MPU6050_GYRO_YOUT_L        0x46
#define MPU6050_GYRO_ZOUT_H        0x47
#define MPU6050_GYRO_ZOUT_L        0x48

// PWR_MGMT_1 Register  |   bit_7    |bit_6|bit_5|bit_4| bit_3  |bit_2|bit_1|bit_0|
//                      |DEVICE_RESET|SLEEP|CYCLE| --- |TEMP_DIS|   CLKSEL[2:0]   |
// El SLEEP biten 1 pone al sensor en modo sleep, para comenzar habrá que ponerlo a 0.
#define MPU6050_PWR_MGMT_1         0x6B

// WHO_AM_I Register  |bit_7|bit_6|bit_5|bit_4|bit_3|bit_2|bit_1|bit_0|
//                    | --- |           WHO_AM_I[6:1]          | --- |
// The WHO_AM_I contiene la dirección del dispositivo. Valor por defecto 0x68
//
#define MPU6050_WHO_AM_I  0x75
#define MPU6050_I2C_ADDRESS 0x68
//------------------------------------------------------------
// Declaración de una variable para almacenar los registros obtenidos del sensor.
typedef union acc_gyro_t_union{
        struct{
                uint8_t acc_x_h;
                uint8_t acc_x_l;
                uint8_t acc_y_h;
                uint8_t acc_y_l;
                uint8_t acc_z_h;
                uint8_t acc_z_l;
                uint8_t gyro_x_h;
                uint8_t gyro_x_l;
                uint8_t gyro_y_h;
                uint8_t gyro_y_l;
                uint8_t gyro_z_h;
                uint8_t gyro_z_l;
        } reg;
        struct{
                int16_t acc_x;
                int16_t acc_y;
                int16_t acc_z;
                int16_t gyro_x;
                int16_t gyro_y;
                int16_t gyro_z;
        } value;
};
```

```
// variables MPU6050
//
double acc_x=0, acc_z=0, gyro_z=0;
double gyro_last_z=0;
double O_acc_y=0, O_gyro_z=0;

double acc_x_offs=0, acc_z_offs=0, gyro_z_offs=0;

 void setup(){
        // Declaracion de las variables:
        //
        uint8_t c;
        uint8_t Zero = 0;

        // configura e inicia la comunicacion del sensor.
        //
        Serial.begin(38400);

        // Inicializa la comunicación por el bus I2C.
        //
        Wire.begin();

        // Configuración al comienzo
        //    - Giro a 250 º/s
        //    - Acelerómetro a 2g
        //    - Reloj interno a 8MHz
        //    - Dispositivo en sleep mode.
        //        La siguiente función leerá el sleep bit, al comienzo esta en 1.
        //        Para comenzar el bit debe ser puesto a 0.
        //
        MPU6050_read (MPU6050_PWR_MGMT_1, &c, 1);
        Serial.print(F("PWR_MGMT_1 : "));
        Serial.println(c,BIN);
        Serial.println(F("_____________________________________________________________"));

        / Pone el sleep bit a 0 iniciando así el sensor.
        /
        MPU6050_write(MPU6050_PWR_MGMT_1, &Zero, 1);
        MPU6050_read (MPU6050_PWR_MGMT_1, &c, 1);
        Serial.print(F("PWR_MGMT_1 : "));
        Serial.println(c,BIN);
        Serial.println(F("_____________________________________________________________"));

}
```

```
void loop(){

        // Declaracion de las variables:
        //
        acc_gyro_t_union acc_gyro;

        // Lectura de los registros y almacenamiento en la registro acc_gyro
        // previamente definido
        //
        MPU6050_read(MPU6050_ACCEL_XOUT_H, (uint8_t*) &acc_gyro, sizeof(acc_gyro));

        // El orden de los registros alto y bajo no coincide entre el microcontrolador y el sensor.
        // Para ello la función cambio los intercambia obteniéndolos así de la forma adecuada.
        //
        acc_gyro.value.acc_x = MPU6050_cambio(acc_gyro.reg.acc_x_h, acc_gyro.reg.acc_x_l);
        acc_gyro.value.acc_y = MPU6050_cambio(acc_gyro.reg.acc_y_h, acc_gyro.reg.acc_y_l);
        acc_gyro.value.acc_z = MPU6050_cambio(acc_gyro.reg.acc_z_h, acc_gyro.reg.acc_z_l);
        acc_gyro.value.gyro_x = MPU6050_cambio(acc_gyro.reg.gyro_x_h, acc_gyro.reg.gyro_x_l);
        acc_gyro.value.gyro_y = MPU6050_cambio(acc_gyro.reg.gyro_y_h, acc_gyro.reg.gyro_y_l);
        acc_gyro.value.gyro_z = MPU6050_cambio(acc_gyro.reg.gyro_z_h, acc_gyro.reg.gyro_z_l);

        // Se imprime la información por el puerto serie
        //
        Serial.print(acc_gyro.value.acc_x, DEC);
        Serial.print(F(", "));
        Serial.print(acc_gyro.value.acc_y, DEC);
        Serial.print(F(", "));
        Serial.print(acc_gyro.value.acc_z, DEC);
        Serial.print(F(", "));
        Serial.print(acc_gyro.value.gyro_x, DEC);
        Serial.print(F(", "));
        Serial.print(acc_gyro.value.gyro_y, DEC);
        Serial.print(F(", "));
        Serial.print(acc_gyro.value.gyro_z, DEC);
        Serial.print(F(", "));
}
```

Función MPU6050_read:

```
// ----------------------------------------------------
// MPU6050_read
//
// Función para leer los registros del dispositivo I2C
//
void MPU6050_read(int start, uint8_t *data, int size){

        int i;

        //Comienza la comunicación estableciendo la dirección del esclavo
        //
        Wire.beginTransmission(MPU6050_I2C_ADDRESS);
```

```
		// Wire.write(value)
		// value: byte que será mandado
		// start contiene la dirección del registro
		// n son el numero de bytes recogidos
		//
		Wire.write(start);

		// Wire.endTransmission(parameter) ; parameter --> variable booleana
		// false envia un restart, manteniendo la conexión activa.
		// true envia un stop, dejando el bus libre tras la transmisión.
		//
		Wire.endTransmission(false);

		// Wire.requestFrom(address, quantity, stop)
		// address: la dirección de 7-bit del dispositivo al que solicita información
		// quantity: numero de bytes pedidos
		// stop : variable booleana.
		//		true envia un stop, dejando el bus libre tras la transmisión.
		//		false envia un restart, manteniendo la conexión activa.
		//
		Wire.requestFrom(MPU6050_I2C_ADDRESS, size, true);

		i = 0;

		// Wire.available() devuelve el numero de bytes disponibles para lectura.
		// El bucle while construye la variable con los datos
		//
		while(Wire.available() && i<size){
			data[i++]=Wire.read
		}
}
```

Función MPU6050_write:

```
// ---------------------------------------------------------
// MPU6050_write
//
// Función para escribir en los registros del dispositivo I2C
//
void MPU6050_write(int start, uint8_t *data, int size){

		//Comienza la comunicación estableciendo la dirección del esclavo
		//
		Wire.beginTransmission(MPU6050_I2C_ADDRESS);

		// Wire.write(value)
		// value: byte que será mandado
		// start contiene la dirección del registro
		//
		Wire.write(start);
		// Wire.write(data, length)
		// data: array de datos enviados como bytes
		// length: numero de bytes a transmitir
		//
		Wire.write(data, size);

		// Wire.endTransmission(parameter) ; parameter --> variable booleana
		// false envia un restart, manteniendo la conexión activa.
```

```
        // true envia un stop, dejando el bus libre tras la transmisión.
        //
        Wire.endTransmission(true);
}
```

Función MPU6050_cambio:

```
// ----------------------------------------------------------
// MPU6050_cambio
//
// Intercambio de los registros alto y bajo
// para que coincida con los del microcontrolador
//
uint16_t MPU6050_cambio(uint8_t x, uint8_t y){

         uint16_t aux;

        aux = ((x<<8) + y);
        return(aux);
}
```

Primeros resultados:

-440, 1580, 15152, -3632, -38, 75,

396, 392, 14524, -3632, -99, 22,

656, -272, 14624, -3648, -78, 61,

472, -696, 14652, -3632, -85, 53,

724, -416, 14508, -3648, -116, 61,

704, -64, 14584, -3648, -139, 61,

560, 336, 14844, -3664, -165, 51,

880, 660, 14492, -3616, -183, 65,

772, 476, 14720, -3632, -189, 68,

424, 608, 14564, -3664, -193, 75,

400, 672, 14500, -3648, -198, 117,

-4, 628, 15020, -3632, -193, 71,

68, 544, 14316, -3616, -206, 32,

328, 460, 14516, -3664, -201, 32,

648, 224, 14900, -3632, -197, 21,

372, 284, 15208, -3648, -159, 70,

472, 84, 14848, -3632, -139, 54,

-48, 304, 15212, -3648, -111, 45,

-700, 1228, 14980, -3664, -59, 15,

-888, 1976, 14368, -3632, -65, 12,

456, -28, 14788, -3648, -73, 26,

## 9.2. Anexo 2. Código para el uso de Parallax-DAQ

Parte incluida en el bucle de configuración:

```
void setup(){
        …
        // Configura captura de datos con Parallax_DAQ
        // Mostrará en Excel: Tiempo|ACC_X|ACC_Y|ACC_Z|GYRO_X|GYRO_Y|GYRO_Z|
        //                    … | … | … | … |  … |  … |  … |
        //                    … | … | … | … |  … |  … |  … |
        //
        Serial.println("CLEARDATA");
        Serial.println("LABEL, TIME, ACC_X,ACC_Y,ACC_Z,GYRO_X,GYRO_Y,GYRO_Z");
        …
}
```

Parte incluida en el bucle continuo:

```
void loop(){
        …
        // Print the raw values.
        //
        Serial.print(F("DATA, TIME, "));
        Serial.print(acc_gyro.value.acc_x, DEC);
        Serial.print(F(", "));
        Serial.print(acc_gyro.value.acc_y, DEC);
        Serial.print(F(", "));
        Serial.print(acc_gyro.value.acc_z, DEC);
        Serial.print(F(", "));
        Serial.print(acc_gyro.value.gyro_x, DEC);
        Serial.print(F(", "));
        Serial.print(acc_gyro.value.gyro_y, DEC);
        Serial.print(F(", "));
        Serial.print(acc_gyro.value.gyro_z, DEC);
        Serial.println(F(", "));
}
```

Resultados:

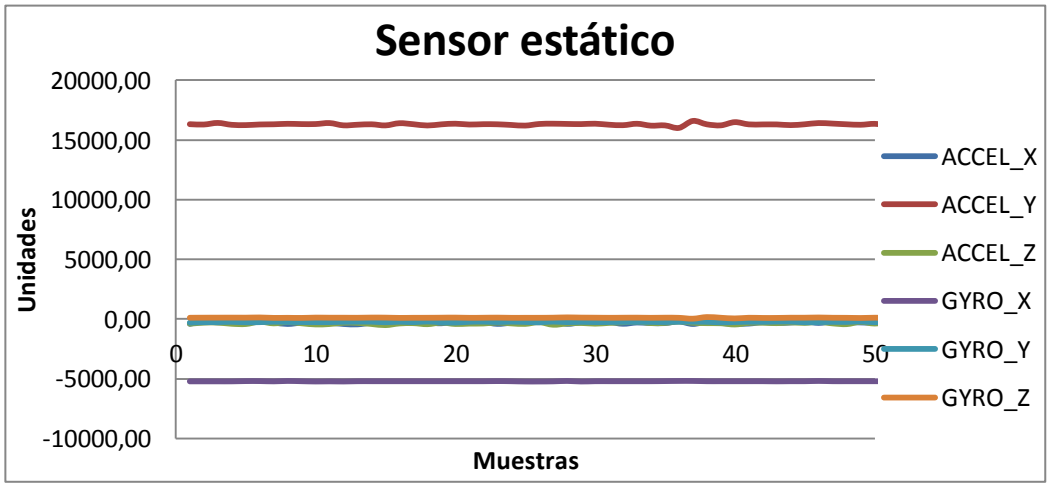



Figura 77. Toma de datos con el sensor estático.

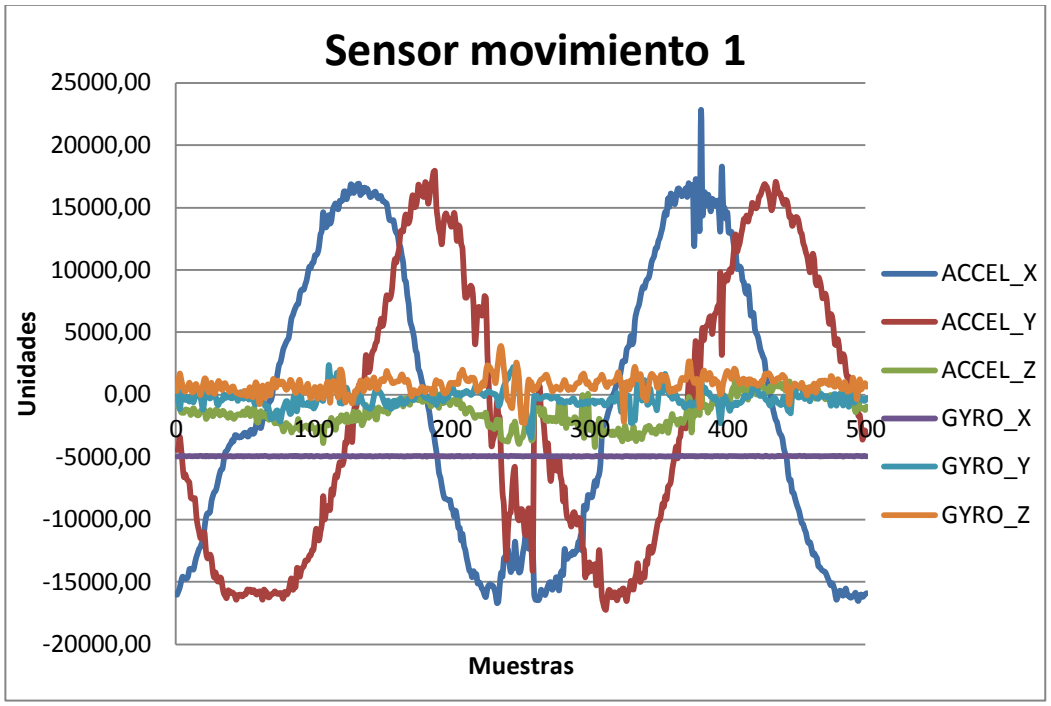



Figura 78. Toma de datos con el sensor en movimiento 1.

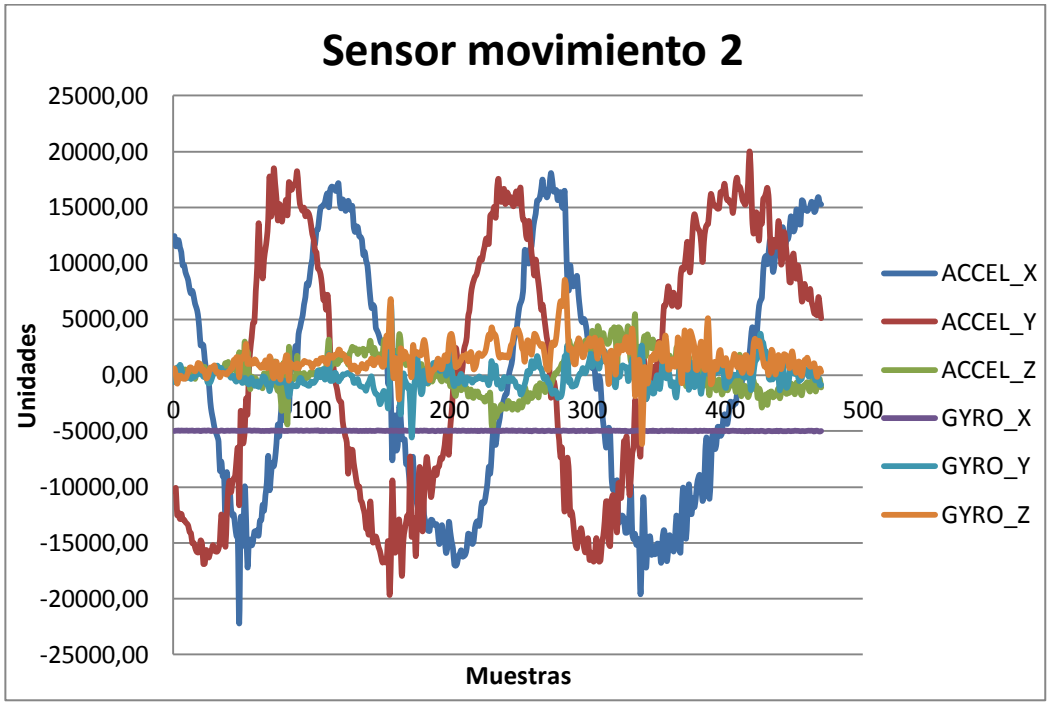



Figura 79. Toma de datos con el sensor en movimiento 2.

## 9.3. Anexo 3. Obtención del ángulo con los acelerómetros.

Función MPU6050_ang:

```
// ------------------------------------------------------
// MPU6050_ang
//
// Función que genera el ángulo con los valores obtenidos
// por los acelerómetros.
// Devuelve: ángulo (-180º , 180º)
//

double MPU6050_ang(int16_t x, int16_t y){

        double ang;

        if (x>0 && y<0){
                ang= - 180 + (atan2(x,y)*180/3.1416);
        }

        if (x<0 && y<0){
                ang= 180 + (atan2(x,y)*180/3.1416);
        }

        if (x>0 && y>0){
                ang= -180 + (atan2(x,y)*180/3.1416);
        }

        if (x<0 && y>0){
                ang= 180 + (atan2(x,y)*180/3.1416);
        }

        return(ang);
}
```

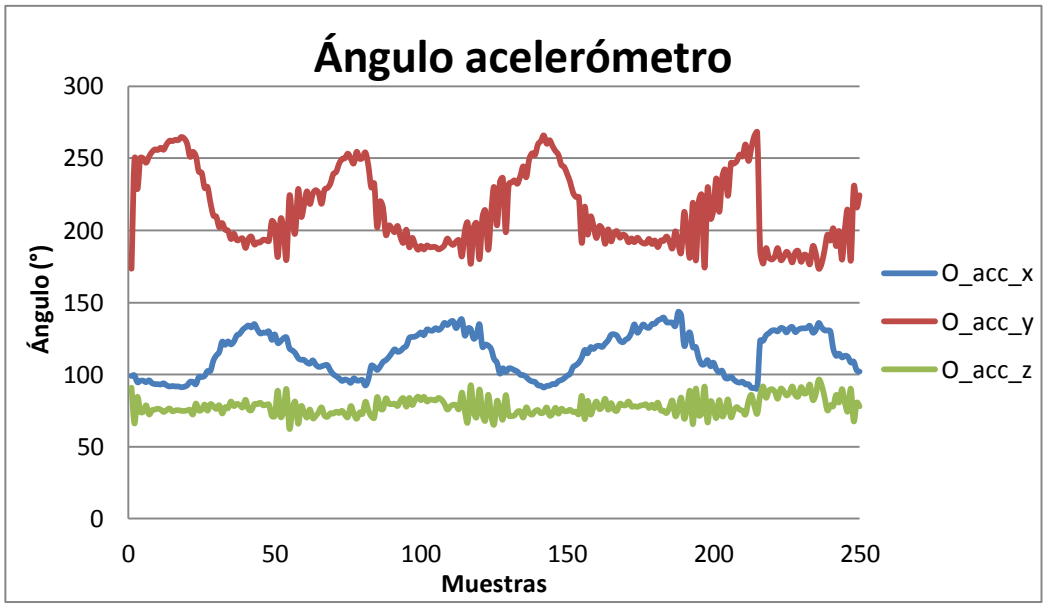



Figura 80. Ángulos obtenidos con los acelerómetros (O_acc).

## 9.4. Anexo 4. Obtención del ángulo con los giróscopos.

Función MPU6050_ang2:

```
// ------------------------------------------------------
// MPU6050_ang2
//
// Función que genera el ángulo con los valores obtenidos
// por los giróscopos.
// Devuelve: ángulo (-180º , 180º)
//

double MPU6050_ang2( double last, double curr, unsigned long time){

        double ang2;

        // curr = curr * 2000 / 32768 ==> con sensibilidad del giroscopo de 2000 º/s
        // el factor para las muestras es 0.061035
        // time = time/1000 ==> para pasar de time(ms) a time(s) hay que dividir entre 1000
        // curr * 2000 / 32768 * time / 1000
        //
        ang2 = last + curr * time * 0.061035 / 1000;

        return(ang2);

}
```

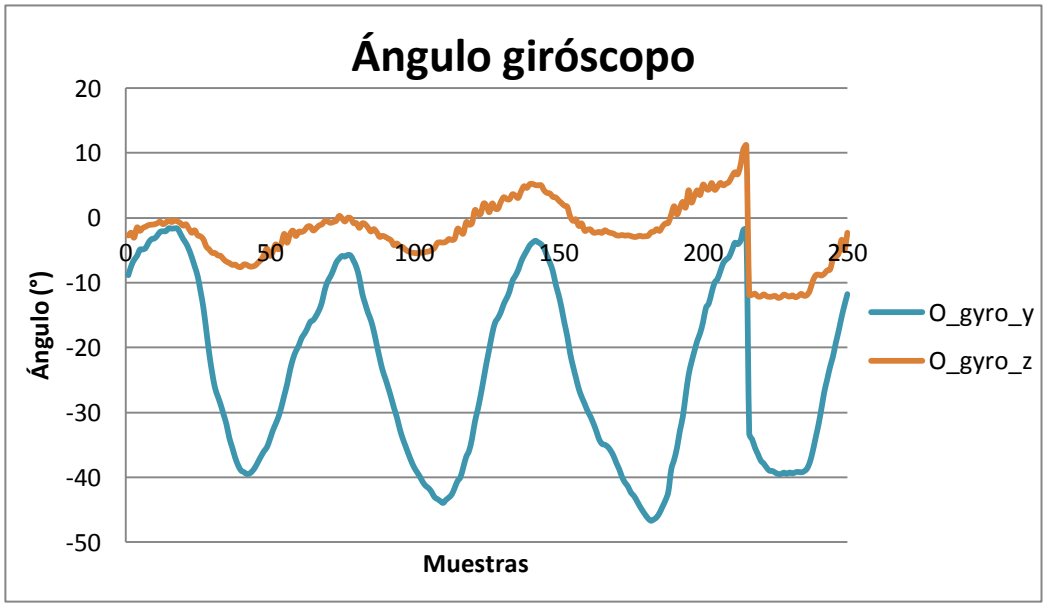



Figura 81. Ángulos obtenidos con los giróscopos (O_gyro).

## 9.5. Anexo 5. Eliminación de offsets.

Parte incluida en el bucle continuo:

```
void loop(){
….
// Bucle de 1000 repeticiones para promediar los valores
//
for (int i=0; i <= 1000; i++){
        // Lectura de los registros y almacenamiento en los registros acc_gyro
        //
        MPU6050_read(MPU6050_ACCEL_XOUT_H, (uint8_t*) &acc_gyro, sizeof(acc_gyro));

        // El orden de los registros alto y bajo no coincide entre el microcontrolador y el sensor.
        // Para ello la función cambio los intercambia teniendolos así de la forma adecuada.
        //
        acc_gyro.value.acc_x = MPU6050_cambio(acc_gyro.reg.acc_x_h, acc_gyro.reg.acc_x_l);
        acc_gyro.value.acc_z = MPU6050_cambio(acc_gyro.reg.acc_z_h, acc_gyro.reg.acc_z_l);
        acc_gyro.value.gyro_z = MPU6050_cambio(acc_gyro.reg.gyro_z_h, acc_gyro.reg.gyro_z_l);

        / Obtención de los offsets de los sensores.
        //
        acc_x_offs = MPU6050_offs(acc_gyro.value.acc_x, acc_x_offs, i);
        acc_z_offs = MPU6050_offs(acc_gyro.value.acc_z, acc_z_offs, i);
        gyro_z_offs = MPU6050_offs(acc_gyro.value.gyro_z, gyro_z_offs, i);
}

// offset del eje en el que actua G, según la disposición
// de nuestro sensor
//
acc_x_offs = acc_x_offs + 16384;
…
}
```

Función MPU6050_offs:

```
// ----------------------------------------------------------
// MPU6050_offs
//
// Función para calcular la media de una secuencia de números,
// el valor obtenido corresponderá con los offset de los sensores
//
double MPU6050_offs(double valor, double media, int i){

        double offs;

        offs = (media/(i+1)*i) + (valor / (i+1));

        return (offs);
}
```

## 9.6. Anexo 6. Uso del filtro complementario.

Parte incluida en el bucle continuo:

```
void loop(){
….
        // Valor del ángulo con el filtro complementario
        //
        angle = 0.88 * (angle + (gyro_z * 0.061035) * (time / 1000)) + O_acc_y * 0.12;
…
}
```

Resultado del empleo del filtro:

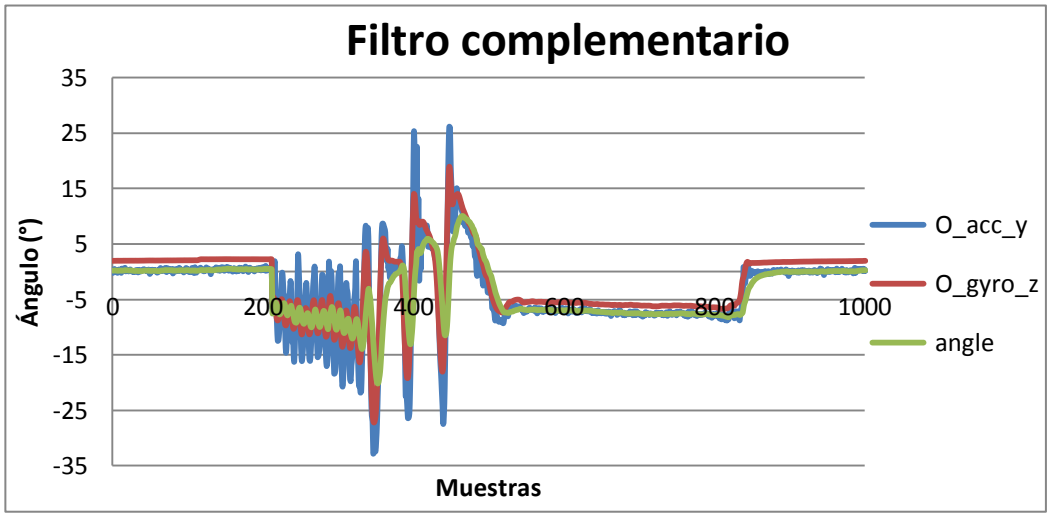



Figura 82. Efecto del filtro complementario (tiempo de muestreo de 20 ms).

Recordar como en el apartado 3.2.5, que se observan los valores del acelerómetro en azul, del giróscopo en rojo, y en verde la estimación del ángulo realizada.

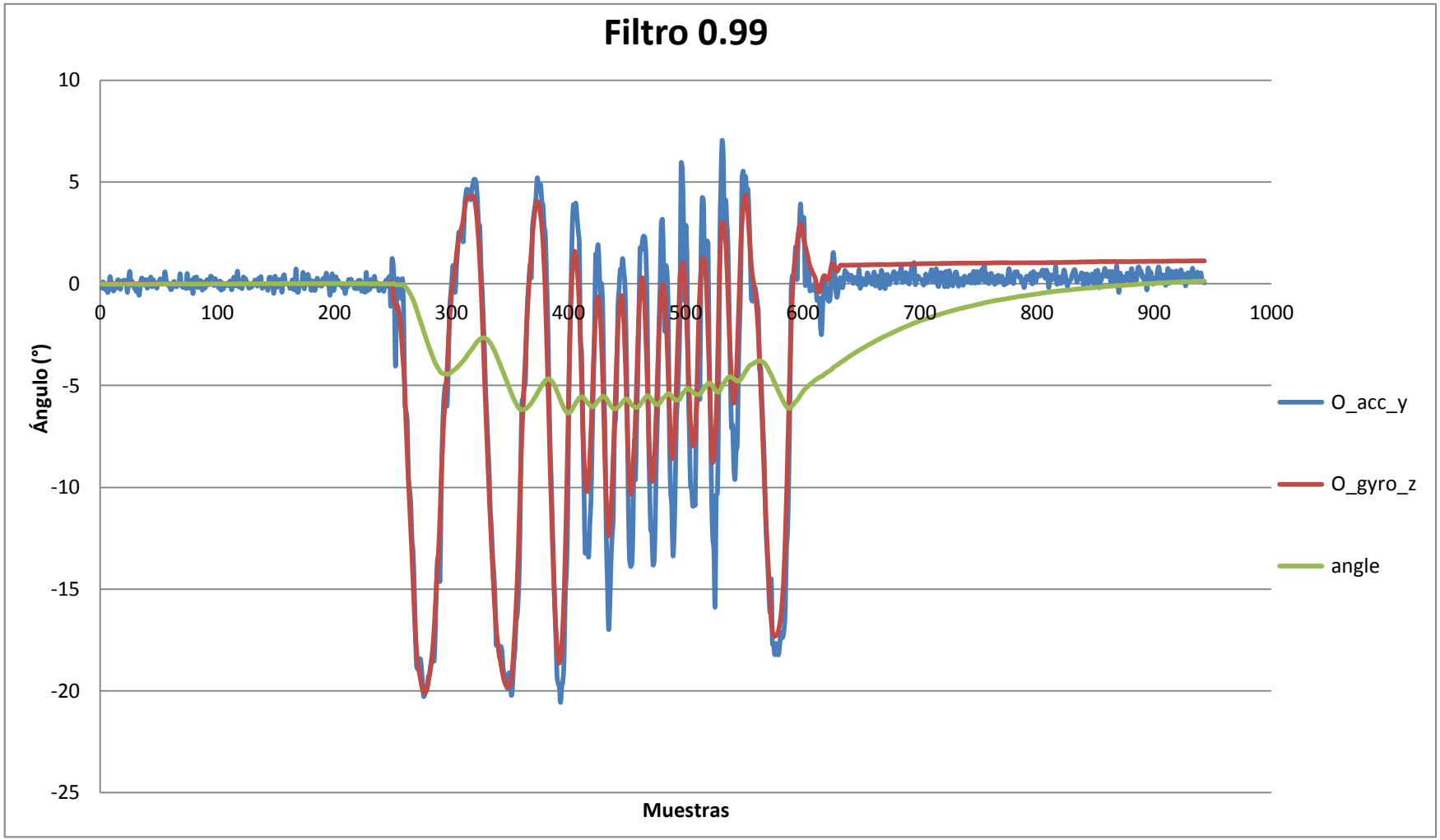



Figura 83. Filtro complementario 0.99 (tiempo de muestreo de 20 ms).

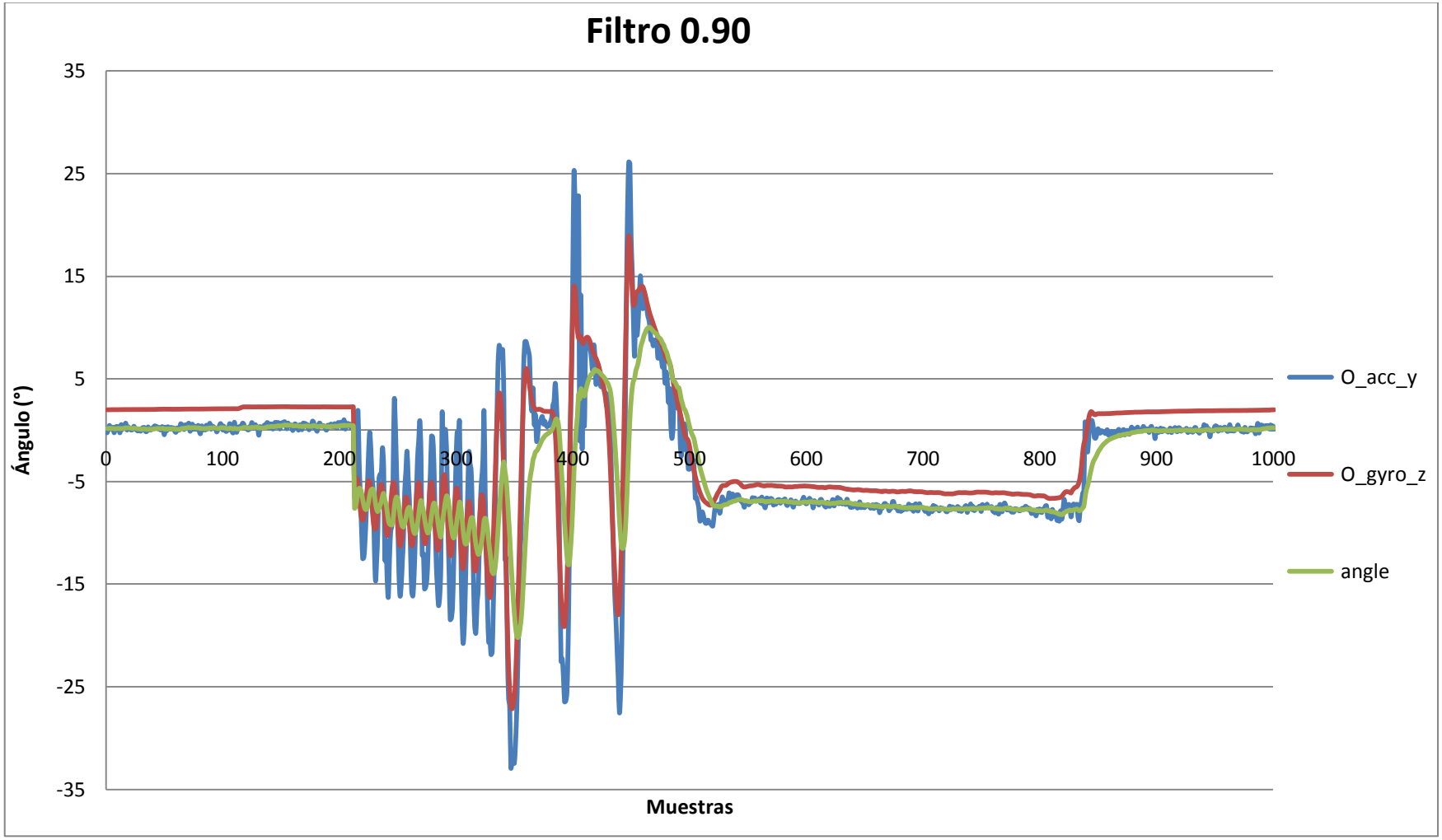



Figura 84. Filtro complementario 0.90 (tiempo de muestreo de 20 ms).

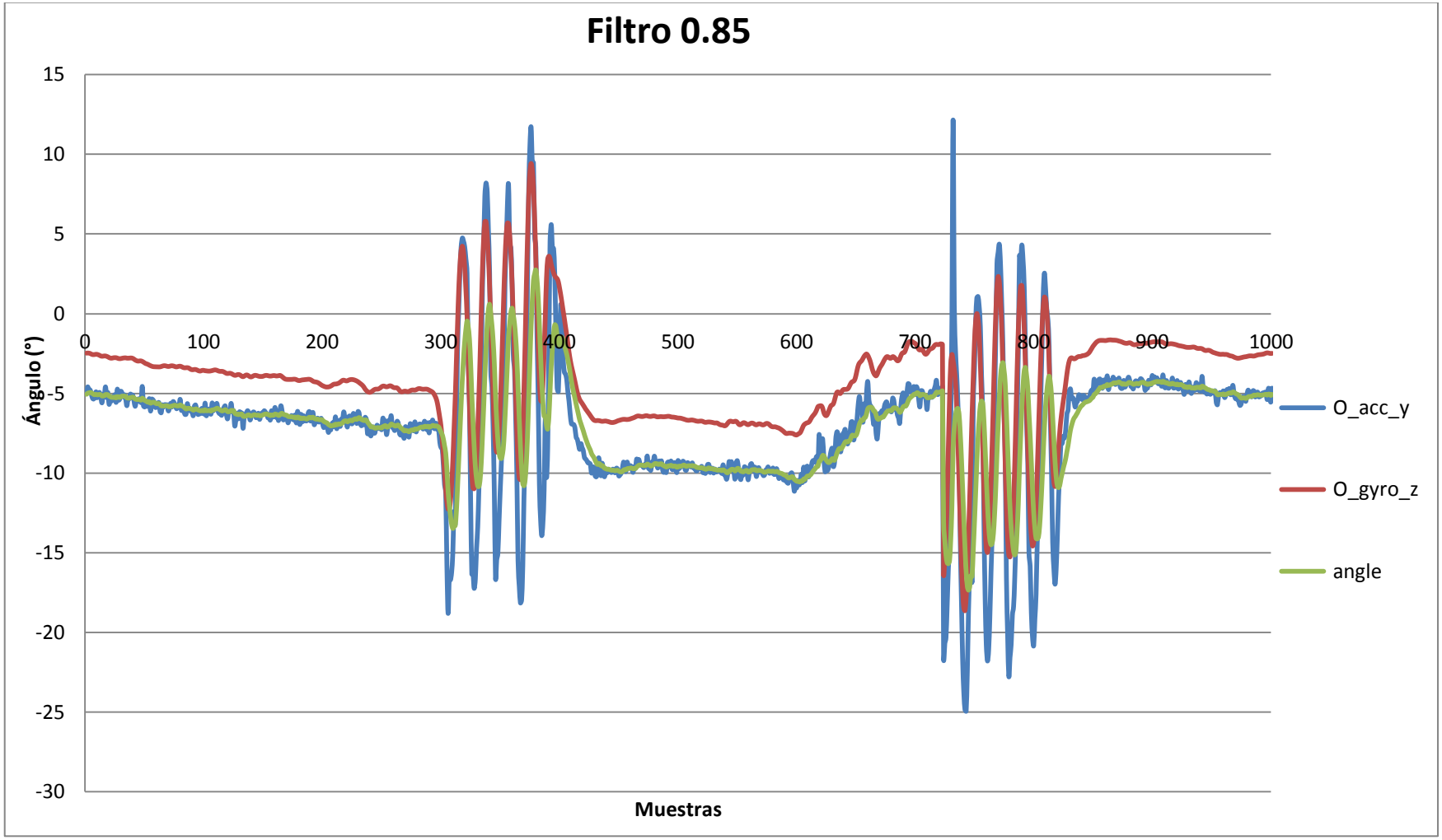



Figura 85. Filtro complementario 0.85 (tiempo de muestreo de 20 ms).

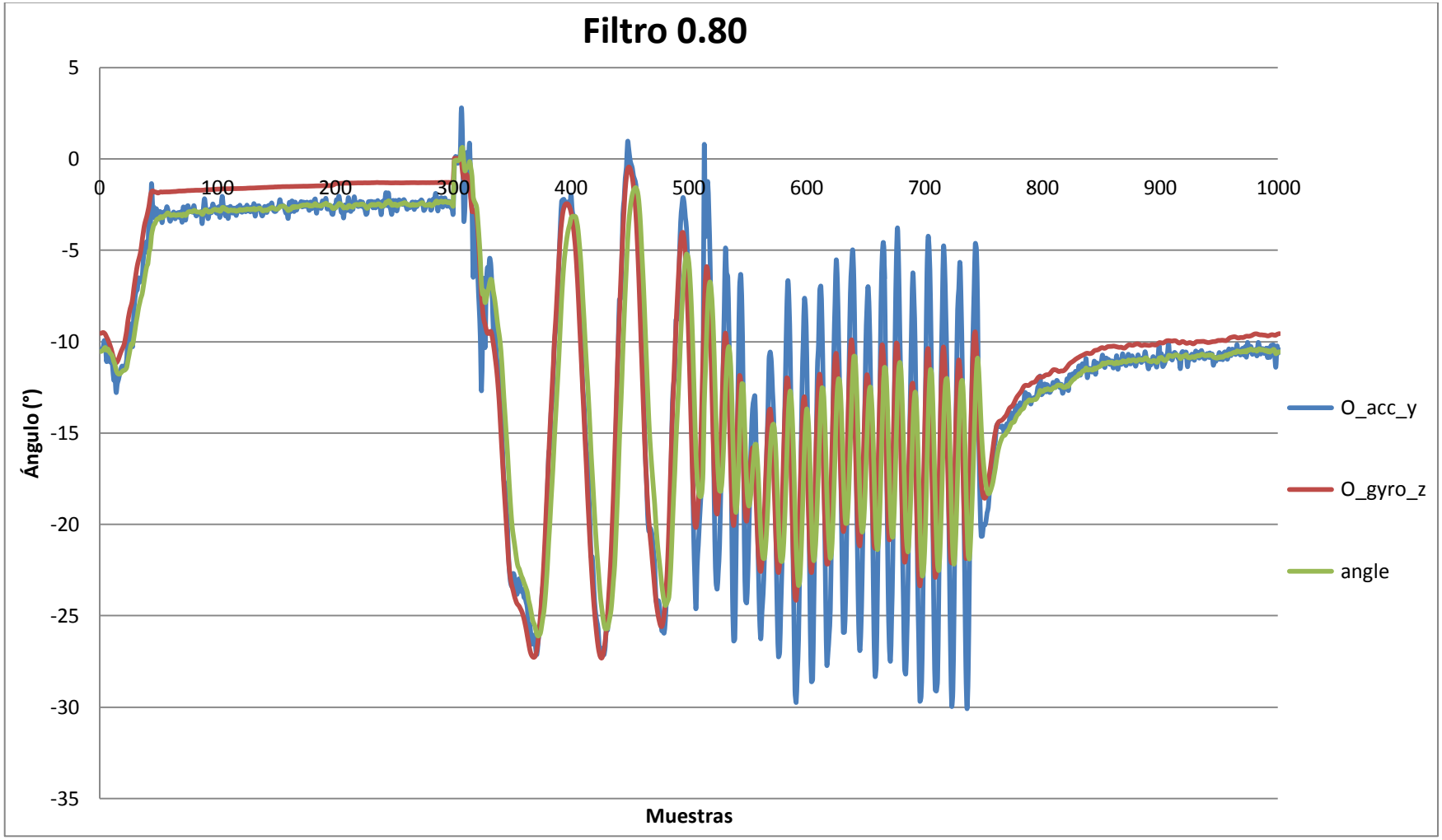



Figura 86. Filtro complementario 0.80 (tiempo de muestreo de 20 ms).

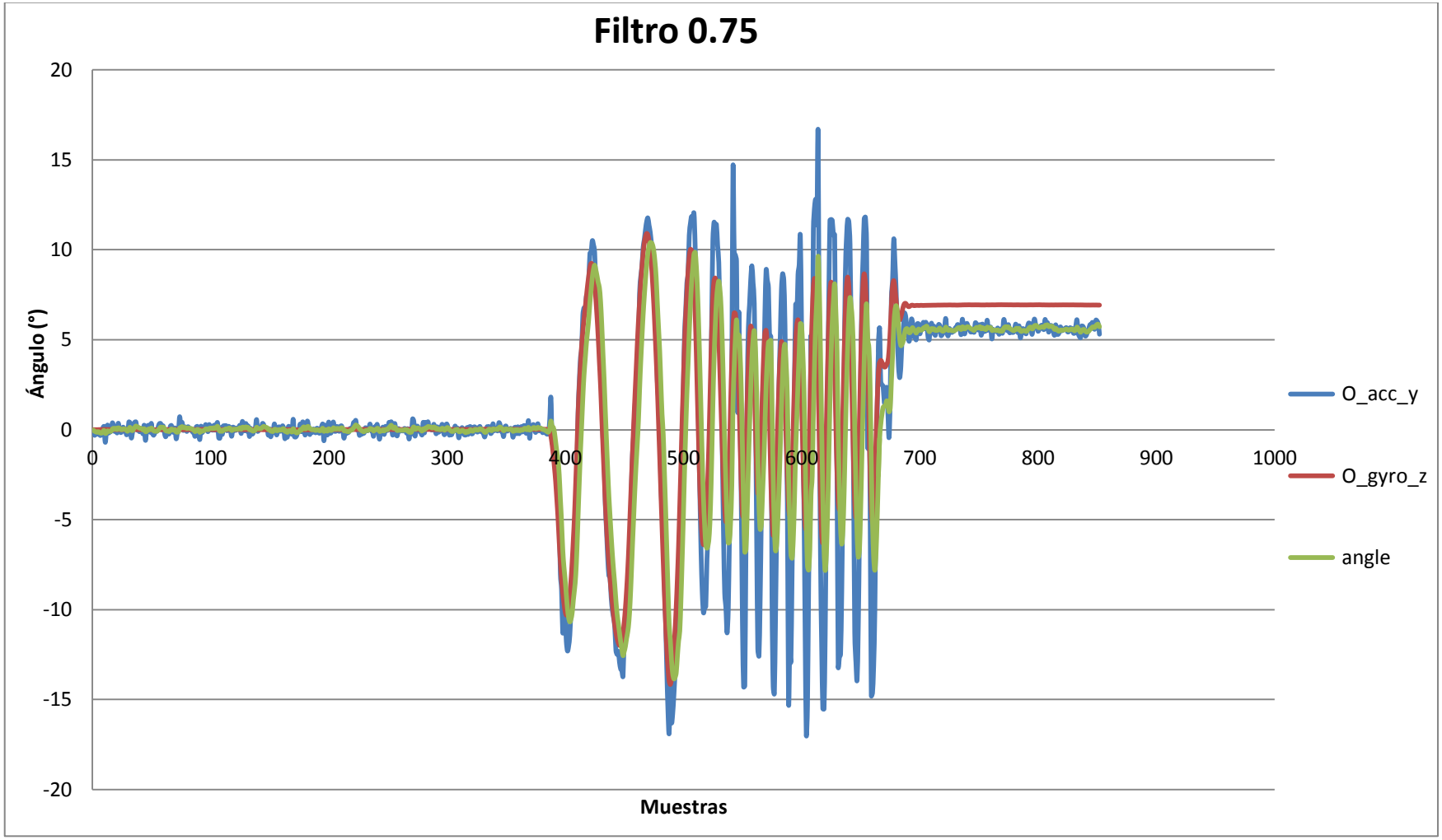



Figura 87. Filtro complementario 0.75 (tiempo de muestreo de 20 ms).

## 9.7. Anexo 7. Implementación del control PID.

Declaración de variables:

```
// variables PID
//
int u= 0;
float kp= 0, ki= 0, kd= 0;
float PID_prop = 0, PID_int= 0, PID_der= 0;
float error_int = 0, error_int_last= 0, error_der= 0;
float error= 0, error_last= 0;
```

Algoritmo PID incluido en el bucle continuo:

```
void loop(){
….
//-----------------------------------------------------------------------------------
//   ALGORITMO DE CONTROL (PID)
//-----------------------------------------------------------------------------------

// Constantes del PID
//
kp =  1;
ki =  1;
kd =  1;

// Ángulo de consigna y obtención del error
//
angle_consigna = 0;
error = angle_consigna - angle;

// Cálculo termino proporcional
//
PID_prop = kp * error;

// Cálculo termino integral
// con implementación anti-WindUp
//
error_int = error_int_last + error * time / 1000;

// condición anti-WindUp 1
//
if (error_int > 250/ki){
     error_int = 250/ki;
}
if (error_int < -250/ki){
     error_int = -250/ki;
}
PID_int = ki * error_int;
```

```
// Cálculo termino derivativo
// con implementación de derivative kick
//
error_der = angle - angle_last ;
PID_der = kd * error_der;

// Generación de la acción de control
//
u = PID_prop + PID_int - PID_der;

// condición anti-WindUp 2
//
if (u > 250){
      u = 250;}
if (u < -250){
      u = -250;}

error_int_last = error_int;
…
}
```

## 9.8. Anexo 8. Programa de testeo de motores.

Programa en el que se configura la disposición de las señales de control para el sentido de giro y la velocidad (con PWM). Corresponde con la disposición estudiada y el esquema del apartado 3.4. Sistema de potencia.

```
//-------------------------------------------------------
//          PRUEBA MOTOR
//-------------------------------------------------------
// Variables para el control de los motores (PWM + L293D)
//
// Variables motor 1
// in_1 e in_2 corresponde a salidas de la señal de control para
//      configurar el sentido del giro.
// en_1 corresponde a enable_1, será el pin por el que salga la señal PWM
//      señal PWM
//
int in_1 = 4;
int in_2 = 5;
int en_1 = 3;
// Variables motor 2
// in_3 e in_4 corresponde a salidas de la señal de control para
//      configurar el sentido del giro.
// en_4 corresponde a enable_1, será el pin por el que salga la señal PWM
//      señal PWM
//
int in_3 = 8;
int in_4 = 9;
int en_2 = 10;
// Señal de control
//
int u;

// Rutina de configuración
//
void setup() {

          // Declara los pines para el control de los motores como salidas
          //
          pinMode(en_1, OUTPUT);
          pinMode(en_2, OUTPUT);
          pinMode(in_1, OUTPUT);
          pinMode(in_2, OUTPUT);
          pinMode(in_3, OUTPUT);
          pinMode(in_4, OUTPUT);
}
```

```
// Rutina en bucle (ejecución continua)
//
void loop() {
        …
        // Valor para el ciclo de trabajo del PWM (entre 0 y 255)
        //
        u = 250;

        // Configuración de los pines y de la señal PWM
        // para giro de los motores en sentido 1
        //
        analogWrite(en_1, u);
        analogWrite(en_2, u);
        digitalWrite(in_1, LOW);
        digitalWrite(in_2, HIGH);
        digitalWrite(in_3, LOW);
        digitalWrite(in_4, HIGH);

        delay(500);

        // Configuración de los pines y de la señal PWM
        // para giro de los motores en sentido 2
        //
        analogWrite(en_1, u);
        analogWrite(en_2, u);
        digitalWrite(in_1, HIGH);
        digitalWrite(in_2, LOW);
        digitalWrite(in_3, HIGH);
        digitalWrite(in_4, LOW);

        delay(500);

        …
}
```

## 9.9. Anexo 9. Dimensiones y peso de los componentes.

Tabla con las dimensiones y pesos de los componentes:

| Componente | Peso(gr) | A(cm) | B(cm) | C(cm) |
|---|---|---|---|---|
| **Chasis** | 29 (chasis + soporte motor) | 20 | 10 | 0.5 |
| **Soporte motores** | | 10 | 3 | 0.5 |
| **Arduino Uno** | 27 | 7 | 5.2 | 0.1 (1 en los pines) |
| **Arduino Nano** | 4 | 4.4 | 1.8 | 0.1 (1 en los pines) |
| **Protoboard** | 48 (Protoboard + sensor + driver + cables)) | 8.2 | 5.4 | 1 |
| **Pila + conector** | 37 | 5 | 2.5 | 1.5 |
| **Motor** | 10 | 2.5 (+ 1 cm de eje) | 1 | 1 |
| **Driver** | | 2 | 0.75 | 1.25 (incluyendo patillas) |
| **MPU6050** | | 2 | 1.5 | (0.2) 1 cm en los pines |
| **Ruedas** | 11 | | | |

Figura 88. Dimensiones y peso de los componentes.

## 9.10. Anexo 10. Cálculo del centro de masas.

Cálculo aproximado del Centro de Masas mediante sistema de masas discreto.

$$\mathbf{r}_{\mathrm{cm}} = \frac{\sum_i m_i \mathbf{r}_i}{\sum_i m_i} = \frac{1}{M}\sum_i m_i \mathbf{r}_i$$

Imagen 89. Centro de Masas con sistema de masas discreto.

- En la disposición 1:

| **Disposición_1: Pilas sobre motores** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Elemento** | | **área (cm2)** | **volumen (cm3)** | **masa (gr)** | **densidad (gr/cm3)** | **x (cm)** | **y (cm)** | **z (cm)** |
| **Chasis** | | 124,75 | 62,375 | 29 | 0,465 | | | |
| | chasis_1 | 30 | 15 | 6,97 | 0,47 | 0 | 0 | 0,65 |
| | chasis_2 | 7 | 3,5 | 1,63 | 0,47 | 4,25 | 9,25 | 0,65 |
| | chasis_3 | 7 | 3,5 | 1,63 | 0,47 | -4,25 | 9,25 | 0,65 |
| | chasis_4 | 15 | 7,5 | 3,49 | 0,47 | 0 | 10 | 0,65 |
| | chasis_5 | 7,88 | 3,94 | 1,83 | 0,47 | 4,63 | 13 | 0,65 |
| | chasis_6 | 7,88 | 3,94 | 1,83 | 0,47 | -4,63 | 13 | 0,65 |
| | chasis_7 | 20 | 10 | 4,65 | 0,47 | 0 | 16,25 | 0,65 |
| | tapa_motor | 30 | 15 | 6,97 | 0,47 | 0 | 0 | -0,65 |
| | Chasis | 124,75 | 62,38 | 29 | 0,47 | 0 | 6,49 | 0,34 |
| **Motores** | | | | | | | | |
| | motor_1 | | 2,5 | 10 | | 3,75 | 0 | 0 |
| | motor_2 | | 2,5 | 10 | | -3,75 | 0 | 0 |
| **Pilas** | | | | | | | | |
| | Pila_1 | | 16,88 | 37 | | 0 | 2,25 | 0 |
| | Pila_2 | | 16,88 | 37 | | 0 | 3,75 | 0 |
| **Protoboard** | | | | | | | | |
| | protoboard + sensor + L293D | | 44,28 | 48 | | 0 | 7,2 | -0,1 |
| **Arduino Nano** | | | | | | | | |
| | Arduino Nano | | 1,58 | 4 | | 0 | 7,2 | 0 |
| **Robot Autobalanceado** | | | | | | | | |
| | | | 146,99 | 175 | | 0 | 4,48 | 0,03 |

Figura 90. Centro de masas en la disposición 1.

Conclusiones: Obtenemos un centro de masas centrado en el eje X, ligeramente desplazado en el eje z, y dispuesto a 4,5 cm aproximadamente del punto de balanceo en la vertical.

- Cálculo análogo al anterior para el Centro de Masas en la disposición 2:

| Disposición_2: Pilas a igual altura que motores | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Elemento** | | **área (cm2)** | **volumen (cm3)** | **masa (gr)** | **densidad (gr/cm3)** | **x (cm)** | **y (cm)** | **z (cm)** |
| **Chasis** | | 124,75 | 62,38 | 29 | 0,47 | | | |
| | chasis_1 | 30 | 15 | 6,97 | 0,47 | 0 | 0 | 0,65 |
| | chasis_2 | 7 | 3,5 | 1,63 | 0,47 | 4,25 | 9,25 | 0,65 |
| | chasis_3 | 7 | 3,5 | 1,63 | 0,47 | -4,25 | 9,25 | 0,65 |
| | chasis_4 | 15 | 7,5 | 3,49 | 0,47 | 0 | 10 | 0,65 |
| | chasis_5 | 7,88 | 3,94 | 1,83 | 0,47 | 4,63 | 13 | 0,65 |
| | chasis_6 | 7,88 | 3,94 | 1,83 | 0,47 | -4,63 | 13 | 0,65 |
| | chasis_7 | 20 | 10 | 4,65 | 0,47 | 0 | 16,25 | 0,65 |
| | tapa_motor | 30 | 15 | 6,97 | 0,47 | 0 | 0 | -0,65 |
| | Chasis | 124,75 | 62,38 | 29 | 0,47 | 0 | 6,49 | 0,34 |
| **Motores** | | | | | | | | |
| | motor_1 | | 2,5 | 10 | | 3,75 | 0 | 0 |
| | motor_2 | | 2,5 | 10 | | -3,75 | 0 | 0 |
| **Pilas** | | | | | | | | |
| | Pila_1 | | 16,88 | 37 | | 0 | 0 | 1,65 |
| | Pila_2 | | 16,88 | 37 | | 0 | 0 | -1,65 |
| **Protoboard** | | | | | | | | |
| | protoboard + sensor + L293D | | 44,28 | 48 | | 0 | 4,2 | -0,1 |
| **Arduino Nano** | | | | | | | | |
| | Arduino Nano | | 1,58 | 4 | | 0 | 4,2 | 0 |
| **Robot Autobalanceado** | | | | | | | | |
| | | | 146,99 | 175 | | 0 | 2,32 | 0,03 |

Figura 91. Centro de masas en la disposición 2.

Conclusiones: Como era de esperar, en este caso el centro de masas continua centrado en el eje X, ligeramente desplazado en el eje z, pero ha descendido a 2,3 cm aproximadamente del punto de balanceo en la vertical.