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

Diseño y construcción de un robot auto-balanceado mediante Arduino

Grado en Ingeniería Eléctrica y Electrónica

Trabajo Fin de Grado

Ander Gracia Moisés

Santiago Tainta Ausejo

Pamplona, 19 de junio de 2017

upna Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Agradecimientos

Me gustaría dar las gracias a Santiago Tainta Ausejo por la ayuda prestada, así como a todos los profesores con los que me encontraba mientras deambulaba por el laboratorio. También agradecerles a mi familia y amigos, que se preocupan y siempre están ahí.

"Nunca pienso en el futuro. Llega demasiado pronto".

Albert Einstein

“Ash Nazg durbatulûk, ash Nazg gimbatul, ash Nazg thrakatulûk agh burzum-ishi krimpatul.”

“Un Anillo para gobernarlos a todos, un Anillo para encontrarlos, un Anillo para atraerlos a todos y atarlos en las tinieblas.”

El señor de los anillos, la comunidad del anillo.

Resumen

El objetivo de este proyecto consiste en un pequeño robot capaz de mantener el equilibrio sobre sus dos ruedas mediante un control digital PID basado en Arduino. Además se plantea añadir un módulo de conexión inalámbrica para su control mediante un dispositivo Smartphone en tiempo real.

Tras comprobar el correcto funcionamiento de los diferentes sensores que se van a emplear, se diseña una PCB en la que se recogen los elementos necesarios para el correcto funcionamiento del robot. Una vez finalizado el diseño eléctrico, se debe realizar el diseño mecánico de las diferentes piezas que formaran la estructura. Estas piezas se diseñaran mediante la herramienta de dibujo SolidWorks para después ser impresas en 3D. Por último, se implementa el software necesario que permite tanto el auto-equilibrado del robot como su control mediante un Smartphone.

Lista de palabras clave

Robot auto-balanceado, Arduino, control PID, sensor de medida inercial (IMU)

Índice

Índice de figuras

Índice de Tablas

1 Introducción

1.1 Fundamentos

En este Trabajo de Fin de Grado (TFG) se realizará el diseño y construcción de un robot auto-balanceado cuyo control estará basado en Arduino y con la posibilidad de ser controlado remotamente desde un Smartphone. Este tipo de robots, buscan solucionar el problema del péndulo invertido, problema usado comúnmente en el campo de control, ya que se consigue que un sistema inestable pase a convertirse en uno estable gracias a la acción generada tras la lectura de sus diversos sensores de entrada.

La acción de control del robot generada para que este se mantenga en posición vertical se consigue mediante un control PID digital programado en Arduino, el cual funciona como controlador del sistema. La información que se necesita conocer para poder realizar el control es proporcionada por dos encoders situados en el eje de los motores. De esta forma es posible conocer la velocidad a la que se desplaza el robot. Además, empleando una unidad de medida inercial (IMU) que incorpora un acelerómetro y un giroscopio, se calculará el ángulo de inclinación del robot.

Uno de los aspectos más complicados a la hora de realizar este proyecto es la correcta sintonización del PID. Para realizar esta tarea se emplea el método de Ziegler-Nichols, con el cual se obtienen uno valores aproximados del PID, pero que no están completamente ajustados. Para conseguir un buen control, se parte de los resultados obtenidos y se modificarán, siguiendo un orden, mediante el método de prueba y error. Por este motivo este apartado es uno de los más complicado y costoso del proyecto.

Otra de las ideas que se quieren implementar en el robot, es la capacidad de poder controlarlo como si se tratase de un juguete radiocontrol. Para ello se añade un módulo de comunicación inalámbrica, con el que se podrá conectar el robot a un Smartphone mediante comunicación por vía bluetooth empleando una pequeña aplicación. Además se empleará esta conectividad para la sintonía de los parámetros del PID.

Finalmente, el último aspecto importante del proyecto, es la elaboración de los diferentes bloques de alimentación y mecánico. Se deberá diseñar una PCB que contenga todos los elementos necesarios para el correcto funcionamiento tanto de los sensores como del controlador que necesitamos. También se tiene que diseñar todos los elementos que componen la estructura del robot, la cual será impresa en 3D, para lo cual se empleará el programa de SolidWorks.

Dentro del ámbito del aprendizaje, este robot puede emplearse en asignaturas de control como referencia para implementar controladores y ver cómo afectan los diferentes términos de un control PID a un sistema en tiempo real. También se puede emplear para dar charlas en colegios, ya que es muy llamativo ver como un robot, que solo tiene dos ruedas, puede mantener el equilibrio por sí solo.

1.2 Objetivos

Como ya se ha comentado, este Trabajo Fin de Grado intenta resolver el problema del péndulo invertido mediante un control digital, específicamente, diseñar un algoritmo de control digital robusto e implementarlo en Arduino. Esto nos permite ahondar en el campo del control, en el ámbito de la electrónica digital y en el campo de la programación, de manera que ampliamos nuestros conocimientos. Para conseguir este fin, es necesario definir los objetivos que se pretenden alcanzar en los diferentes campos. Así, en el campo del control se plantean los siguientes objetivos:

- Comprender el funcionamiento de un PID digital y los diferentes parámetros que lo definen
- Entender cómo afectan estos parámetros a un sistema real
- Estudiar y buscar información para la mejora del control

Dentro del ámbito de la electrónica, los objetivos que se buscan son:

- Seleccionar los componentes y dispositivos que forman parte del diseño
- Diseñar y montar los diferentes circuitos eléctricos necesarios para la obtención de datos, para así poder analizar los datos que proporcionan
- Diseñar y montar una PCB, la cual contenga todos los componentes y dispositivos del diseño previsto

Por último, mediante la programación, se recogen los objetivos anteriores y se consigue realizar un programa que cumpla nuestro propósito. Así, los objetivos serian:

- Estudio y aprendizaje del entorno de programación de Arduino, basado en el lenguaje
- Comprender los diferentes protocolos de comunicación que se emplean, que en nuestro caso serán I2C y bluetooth
- Desarrollo de un programa capaz de leer la información de los diferentes dispositivos, y generar una acción de control mediante el control digital implementado en él

1.3 Metodología

Tras indicar cuál es la motivación y el propósito del presente trabajo, se expondrá a continuación la metodología empleada para llevar a cabo los objetivos mencionados:

1. Investigar acerca del problema del péndulo invertido y ver cuáles son las diferentes formas de solucionarlo
2. Investigar acerca de la estructura y sensores que se necesitan para la realización de este robot
3. Comprender el funcionamiento de los sensores de medida inercial (IMU) y elaborar un algoritmo que permita la obtención del ángulo de inclinación mediante el uso de un giroscopio y un acelerómetro
4. Diseño de la estructura del robot
   - Empleando el software de dibujo SolidWorks se obtienen los ficheros para poder imprimir los componentes del robot
   - Ensamblaje del robot
   - Colocación de los motores, batería e interruptor de ON/OFF
5. Diseño de una PCB que contiene los elementos necesarios para el correcto funcionamiento del robot, así como una correcta ordenación de los componentes
   - Diseño de la PCB mediante el software de diseño eléctrico de PCB DesignSpark
   - Soldado de los componentes necesarios
   - Ensamblado de la PCB en el robot en el espacio reservado
   - Conexión de los sensores, motores y fuente de alimentación
6. Programación del controlador digital PID
   - Programación de una versión inicial del control con un único lazo de realimentación
   - Estudio del funcionamiento de un control digital y métodos de mejora del mismo
   - Implementación de las mejoras en el controlador PID para conseguir mayor robustez
7. Sintonización del controlador para mantener la posición vertical
8. Inclusión de un segundo lazo de control que permita modificar la velocidad del robot
9. Programación de las acciones que se deben realizar tras mandar una orden desde un dispositivo smartphone

2 Estado del arte

2.1 Historia

Los “self balncing robots” son robots capaces de mantener el equilibrio sobre dos ruedas sin verse afectados por acciones externas. Para ello, se basan en el principio del péndulo invertido, el cual consiste en un péndulo o varilla que gira libremente sobre uno de sus extremos. Este extremo está unida a una articulación situada en un carro que se puede mover de manera horizontal, tal y como puede verse en la Figura 1.

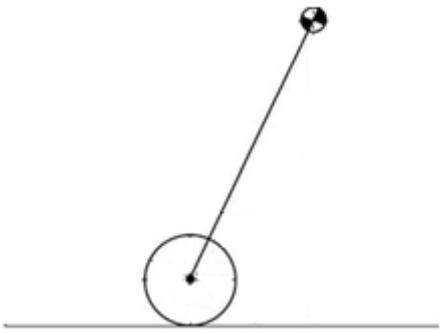


Figura 1. Péndulo invertido.

El origen de estos robots se remonta al año 1986 en la universidad Electro-Comunicaciones de Chōfu, Japón, donde el profesor Kazuo Yamafuji creó un robot capaz de simular el comportamiento de un péndulo invertido [1][2]. El diseño del robot incluye un eje provisto de ruedas y un carro que contenía el dispositivo de estabilización y control.

Figura 2. Robot de Kazuo Yamafuji. Fuente

Un ejemplo más reciente de robot que imita el péndulo invertido puede encontrarse en el año 2000 [3]. Este fue creado en el laboratorio de Electrónica Industrial del Swiss Federal Institute of Technology y le añadieron pesos a la varilla para simular el peso de un ser humano, tal y como puede observarse en la Figura 3. Este prototipo llamado “Joe” es una primera versión, y la base de los “SEGWAY HT”

Figura 3. Joe. **Fuente**:

En la actualidad los robots autobalanceados han sufrido un gran incremento en su popularidad. Esto ha sido debido principalmente al gran desarrollo que ha habido de plataformas de bajo coste para el desarrollo de sistemas eléctricos, así como a la fácil disponibilidad y bajo coste de los componentes electrónicos necesarios para poder ensamblarlos. De esta forma se pueden encontrar diseños basados en microcontroladores PIC o AVR, e incluso se puede llegar a construir un péndulo invertido mediante legos y la plataforma "Lego Mindstorm", dando como resultado un robot capaz de mantener el equilibrio midiendo la aceleración y calculando su inclinación.

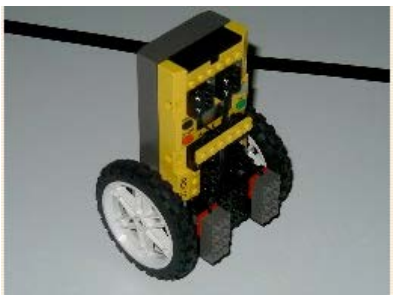


Figura 4. Legway.

La principal aplicación comercial de este tipo de robots es el SEGWAY, un vehículo de transporte giroscópico eléctrico de dos ruedas [4]. Creado en 2001 por Dean Kamen, es el primer dispositivo de transporte autobalanceado pensado para el transporte, el trabajo, la seguridad y el ocio. No es raro encontrar tanto patrullas de agentes de seguridad montados sobre Segways en las rondas como personas montadas de excursión por la ciudad.

Figura 5. Segway. **Fuente**

El futuro de este tipo de robots autobalanceados está destinado a ser robots que se mantienen en equilibrio sobre una bola. Para ello se tiene que diseñar un algoritmo de control preciso, así como un sistema de control de tres motores diferentes, ya que el movimiento es omnidireccional. Además este tipo de desplazamiento necesita un tipo de rueda especial que permita moverse en más de una dirección [5]. Un ejemplo de este tipo de robots es el famoso BB-8 de la película *Star Wars: The Force Awakens.*

Figura 6. Robot BB-8. Fuente

2.2 Principio de funcionamiento

Un péndulo invertido es un péndulo, cuyo centro de masa está situado por encima de su punto de pivotaje. A menudo se implementa sobre un carro y se limita el péndulo a un grado de libertad, consiguiendo un banco de pruebas muy completo e interesante para la ingeniería de control no lineal. La idea principal es mantener el sistema en equilibrio sin que se vea afectado por acciones externas.

Todos los robots equilibristas resuelven el problema del péndulo invertido, por lo que prácticamente todos tienen una forma similar y los mismos elementos que lo componen. Por este motivo se puede observar elementos comunes a todos ellos:

- Ruedas: tienen un diámetro bastante grande (generalmente 1/3 de la altura del robot), y tienen que proporcionan buen agarre con el suelo

- Diseño: Se crean estructuras altas con un centro de gravedad muy bajo

- Unidad de medida inercial: sensor capaz de obtener la medida de la inclinación del robot. Muy comúnmente se emplea la combinación de un acelerómetro y un giroscopio

- Control: se encarga de proporcionar una acción correctora del error producido entre una variable a medir y la referencia que debería seguir, se puede emplear cualquier tipo de control, ya sea analógico, digital o borroso. Habitualmente se emplea el control PID (acrónimo de Proporcional, Integrador y Derivativo) que actúa de tres maneras diferentes sobre el error y la acción que genera resulta igual a:

$$u(t) = K_P \cdot e(t) + K_I \int e(t) \cdot dt + K_D \cdot \frac{d}{dt} e(t)$$

2.3 Aplicaciones

Las aplicaciones de este tipo de robots que se basan en el péndulo invertido son muy variadas. Se puede encontrar estos tipos de aplicaciones en diferentes lugares, ya sea:

- *Laboratorios y aulas:* Estos dispositivos son un banco de pruebas muy completo para la ingeniería de control. Se puede encontrar diferentes prototipos de péndulo invertido, ya sean del tipo péndulo invertido sobre carro, o péndulo de Furuta [6].

Figura 7. Péndulo invertido sobre carro (izda.) y Péndulo de Furuta (dcha.). **Fuente**

- *Dispositivos comerciales:* Hay empresas que crean dispositivos que se basan en el péndulo invertido para el transporte de personas, mejorando la movilidad y la eficiencia a la hora de desplazarse rápidamente al trabajo o a cualquier lugar.

Figura 8. Segway. **Fuente**

- *Aeroespacial:* Se requiere el control activo de un cohete para mantenerlo en la posición vertical invertida durante el despegue. El ángulo de inclinación del cohete es controlado por medio de la variación del ángulo de aplicación de la fuerza de empuje, situado en la base de este. Uno de los ejemplos más claros en este sector es el *Falcon9* de la empresa *SpaceX* [7]. La idea es conseguir un cohete de bajo coste, de manera que el cohete debe ser reutilizable, así que se busca recuperar la lanzadera intacta sin que reciba daños. Esto se consigue haciendo que la lanzadera aterrice verticalmente, por lo que hay que buscar solucionar el problema del péndulo invertido.

Figura 9. Despegue de un cohete Falcon9. **Fuente**

- *Biomecánica:* Es frecuentemente utilizado para modelar bípedos caminantes, como el humanoide Asimo de Honda [8]. En los robots bípedos la pierna de apoyo en contacto con el suelo se modela como un péndulo invertido, mientras que la pierna en movimiento se comporta como un péndulo que oscila libremente suspendido de la cadera del humanoide.

Figura 10. Asimo. **Fuente**

3 Descripción del sistema.

3.1 Introducción.

Este apartado trata de dar explicación a los diferentes elementos que forman parte del robot. Se buscara investigar sobre los componentes necesarios que se van a emplear, de manera que se entienda el funcionamiento de estos. Los principales elementos que forman el sistema son:

- Controlador: Se emplea un control PID digital con el que se intentara solucionar la inclinación del ángulo
- IMU: La unidad de medida inercial se emplea para el cálculo del ángulo de inclinación, y está formada por un acelerómetro y un giroscopio, por lo que se buscara una forma de entender su funcionamiento y descubrir como extraer el ángulo de los datos proporcionados.
- Motores DC: Son los encargados del movimiento del robot, por lo que deberán actuar en función de la información que se obtenga del control

3.2 Modelo matemático del sistema.

Como ya se ha comentado en los apartados anteriores, este sistema se comporta como un péndulo invertido. Si se quiere realizar un control para corregirlo, será necesario empezar por comprender cuál es la dinámica que rige el movimiento del péndulo invertido. Una vez conocida la dinámica, se puede determinar qué tipo de sensores y actuadores serán necesarios para implementar el control.

En nuestro caso, para conseguir que el robot mantenga el equilibrio será necesario modelar el conocido problema del péndulo invertido. Este problema consiste en un péndulo cuyo centro de masas se encuentra por encima del eje de balanceo. Así, que el sistema tenga esta característica, le proporciona al péndulo una inestabilidad estática, la cual se puede compensar si se aplica un momento o par en el eje de giro.

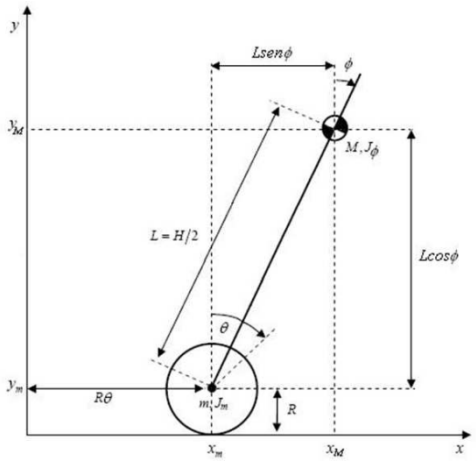


Figura 11. Esquema del péndulo invertido. Fuente

El principal objetivo de este proyecto consiste en controlar el equilibrio de un péndulo invertido, basándose en el ángulo de inclinación de este respecto a la vertical. Analizar uno de estos sistemas es un problema clásico se la teoría de control y en la dinámica de sistemas, así

que hay mucha información [9][10]. También están presentes en laboratorios de control, en bancos de prueba en los que experimentar con diferentes tipos de controles, como puede ser un PID, un control borroso (fuzzy)…

Las ecuaciones de movimiento de los péndulos invertidos, dependen de las condiciones iniciales del péndulo. Suponiendo una configuración inicial en la cual se fija el punto de pivote en el espacio, la dinámica del sistema es similar a la de un péndulo no invertido. Si no se asume ninguna fricción debida al rozamiento, y el sistema está restringido a un movimiento en dos dimensiones, la ecuación que rige el movimiento es [11]:

$$\alpha - \frac{g}{L}\sin(\theta) = 0$$

donde:

- α: Aceleración angular del sistema.
- g: Aceleración de la gravedad (9.8m/s^2).
- L: Longitud del péndulo.
- Θ: Angulo de inclinación respecto a la vertical.

Así, el péndulo se acelerara más conforme mayor sea la distancia entre el péndulo y la posición vertical de equilibrio. Esta ecuación de movimiento del péndulo invertido se puede obtener mediante el par y el momento de inercia. El péndulo se supone como una masa ***m*** puntual fijada a una varilla, la cual está fijada a un punto de pivote en el otro extremo. El par neto del sistema es igual al momento de inercia por la aceleración angular:

$$\tau_{neto} = I \cdot \alpha$$

El par que proporciona la gravedad corresponde al par neto del sistema:

$$\tau_{neto} = m \cdot g \cdot L \cdot \sin(\theta)$$

con θ siendo la inclinación del péndulo respecto a la vertical. Juntando las ecuaciones anteriores, el resultado de la ecuación:

$$I \cdot \alpha = m \cdot g \cdot L \cdot \sin(\theta)$$

donde el momento de inercia del sistema, I, viene dado por:

$$I = m \cdot L^2$$

Sustituyendo la ecuación anterior:

$$m \cdot L^2 \cdot \alpha = m \cdot g \cdot L \cdot \sin(\theta)$$

Y simplificando la expresión anterior:

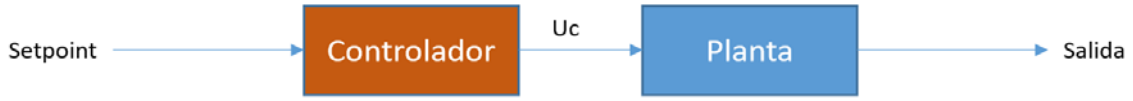
$$\boldsymbol{\alpha = \frac{g}{L} \cdot \sin(\theta)}$$

Así es posible a la ecuación matemática que rige el sistema físico del péndulo invertido, de manera que ya se conoce que variable es necesaria controlar para mantener el equilibrio del robot.

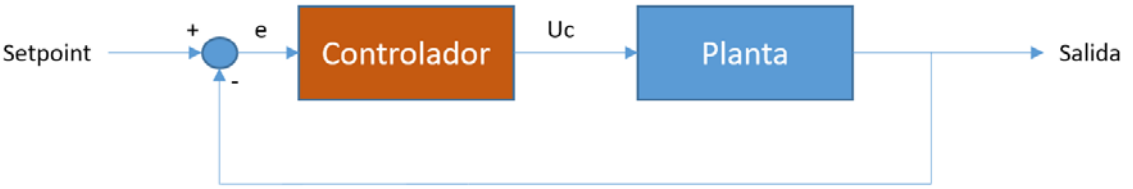
3.3 Control PID

A la hora de realizar el control de una variable de proceso se puede realizar de dos maneras diferentes. Mediante un sistema de control en lazo abierto o con un sistema de control en lazo cerrado:

- **Métodos en lazo abierto:** La información de las características estáticas y dinámicas de la planta se obtienen en lazo abierto en respuesta a un escalón.



- **Métodos en lazo cerrado:** La información de las características del lazo se obtienen a partir de un test en lazo cerrado, generalmente un controlador con acción proporcional pura.



En un sistema típico de control, la variable de proceso es el parámetro que se desea controlar. Para ello, mediante el uso de un sensor se obtiene la variable de proceso y se envía a un sistema de control retroalimentado. Esto es lo que se conoce como un sistema de control en lazo cerrado.

El control PID fue desarrollado por el matemático, ingeniero y científico ruso *Nikolai Fyodorovich Minorsky* [12]. La idea fundamental detrás de un controlador PID es leer la variable del proceso a la salida y obtener a la salida del actuador un término de compensación para la entrada sumando la parte proporcional, integral y derivativa.

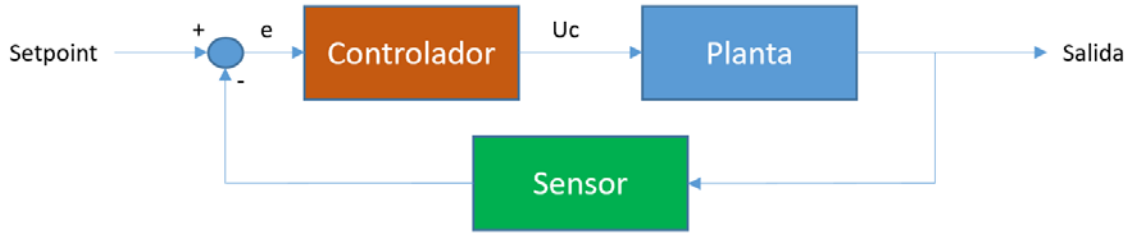


Figura 12. Sistema de control y planta realimentado con un control PID.

La diferencia entre la variable de proceso y la referencia que se desea conseguir es utilizada por el algoritmo de control para determinar la salida ($U_C(t)$) que se envía a la planta del sistema. Esta salida se obtiene a partir del algoritmo de control PID, el cual está formado por tres parámetros distintos: el proporcional, el integral y el derivativo. Estos tres parámetros se suman para así obtener la salida necesaria para conseguir el control.

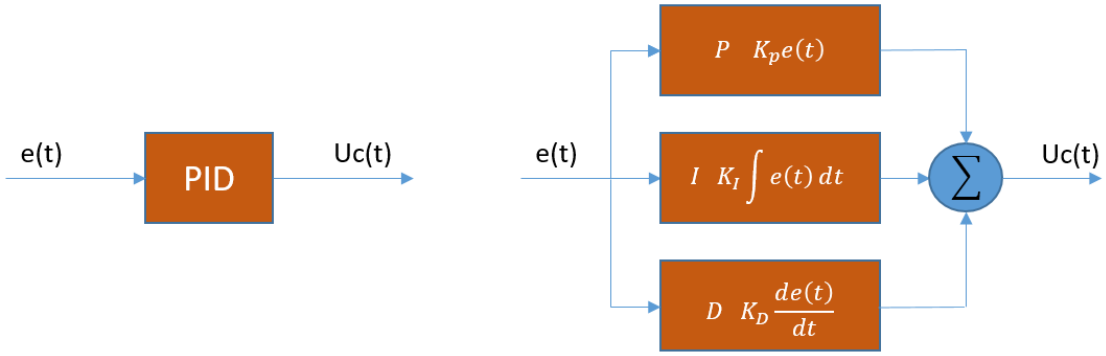


Figura 13. Acción generada por un control PID.

Siguiendo lo explicado en la imagen anterior se llega a obtener la ecuación general de un controlador PID. Esta es:

$$U_c(t) = K_P \cdot e(t) + K_I \cdot \int e(t)dt + K_D \cdot \frac{d}{dt}e(t)$$

Si desarrollamos lo anterior, se llega a la siguiente solucion:

$$U_C(t) = K_P \left[e(t) + \frac{1}{T_i}\sum e(t) + T_d \Delta e(t)\right]$$

dónde:

- e(t) = error(t) obtenido a partir de la diferencia entre la referencia(t) y la variable de proceso(t)
- Uc(t) Salida del controlador.
- K_P Ganancia proporcional. Ganacia del controlador.
- T_i Constante de tiempo integral.
- T_d Constante de tiempo derivativa.

Como se puede observar, esta solución consta de tres apartados diferentes: proporcional, integral y derivativo. A continuación se estudiará cada uno de los mismos por separado.

Apartado proporcional:

La parte proporcional es el producto entre la señal de error y una constante proporcional para lograr que el error en estado estacionario se aproxime a cero. La parte proporcional no considera el tiempo para generar una corrección de las perturbaciones, buscando mantener la variable controlada en el punto de consigna, denominado Setpoint.

$$P_{sal} = K_P \cdot e(t)$$

Apartado integral:

El control integral disminuye y elimina el error en estado estacionario provocado por el modo proporcional. Actúa cuando hay desviación entre la variable y el punto de consigna, integrando la desviación en el tiempo y sumándosela a la acción proporcional.

$$I_{sal} = K_I \int e(t)dt = \frac{K_P}{T_i}\int e(t)dt$$

Apartado derivativo:

La acción derivativa actúa si hay un cambio en el valor absoluto del error. La función es mantener el error mínimo corrigiéndolo proporcionalmente a la misma velocidad que se produce.

$$D_{sal} = K_D \frac{d \cdot e(t)}{dt}$$

A la hora de definir los valores de las diferentes variables del controlador PID, se debe tener en cuenta las siguientes condiciones:

	K_p aumenta	T_i disminuye	T_d aumenta
Estabilidad	se reduce	Disminuye	Aumenta
Velocidad	Aumenta	Aumenta	Aumenta
Error estacionario	No eliminada	Eliminado	No eliminado

Tabla 1. Relaciones entre el sistema y las variables del PID.

A pesar de que los controladores PID son muy utilizados para cualquier tipo de dispositivo, estos presentan ciertas desventajas cuando se implementan para el control de sistemas en los que sea necesario tener una ganancia del lazo PID (K_P) reducida. Si la gancia es elevada, el controlador se saturara, por lo que empezara a generar la accion de control maxima posible con valores de error muy pequeños, lo que hara perder precision a la hora de controlar el sistema.

Por otro lado el controlador PID puede ser muy sensible a la presencia de ruido. El ruido esta formado por pequeñas variaciones de tension de muy alta frecuencia que pueden afectar especialmente a la parte derivativa del control. Esto se debe a que estas pequeñas variaciones pueden aumentar significativamente si se derivan, por lo que en muchas ocasiones, en lugar de emplear un controlador PID, se emplea un control PI, que elimina la parte derivativa y soluciona este probleama a costa de no poder controlar el estado transitorio del sistema. Otra posibilidad para resovler este problema sería la implantacion de un filtro paso bajo que minimice el ruido.

Hay una version diferente del controlador PID, en la cual se propone modficar la estructura clasica del PID (la explicada anteriormente), que consiste en “pesar” de diferente manera la señal de referentcia para los distintos tipos de accion. Esto se emplea en aplicaciones en las cuales la señal de referencia varia muy rapido en el tiempo, o cunado existe mucho ruido en la medida de la variable controlada.

3.4 Sintonización de un PID.

A la hora de implementar un controlador PID será necesario establecer los valores que deben tener los parámetros de ganancia, tiempo integral y tiempo derivativo para que el sistema responda de manera adecuada. Este proceso se conoce como sintonización del PID y, para poder realizarlo, primero hay que obtener la información estática y dinámica del lazo. Para obtener estos valores existen diferentes métodos de sintonización, como pueden ser:

3.4.1 Método Ziegler-Nichols

También conocido como método de las oscilaciones sostenidas o método de oscilación, fue propuesto por Ziegler y Nichols en el año 1942. Para la realización de este método se somete al sistema, en lazo cerrado, a una entrada escalón y utilizando solo un control proporcional.

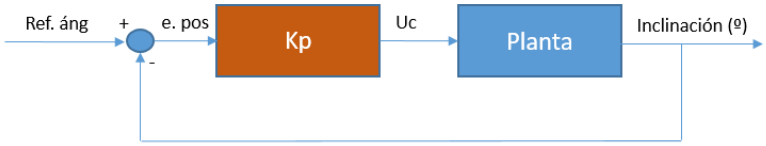


Figura 14. Sistema con una entrada escalón.

Una vez realizada la primera prueba, se varía el valor del controlador proporcional (K_P) hasta conseguir que el sistema tenga una respuesta oscilante sostenida (oscilación con amplitud constante). En este momento se registra el valor de la ganancia y el periodo de la oscilación. Los valores recomendados de sintonización son:

	K_P	T_i	T_d
P	0,5·Kc	∞	0
PI	0,45·Kc	τ/1,2	0
PID	0,6·Kc	τ/2	τ/8

Tabla 2. Sintonización PID mediante Ziegler-Nichols.

3.4.2 Método Basado en la curva de reacción

Este método se realiza en lazo abierto y se obtiene realizando el siguiente procedimiento:

- Se lleva a la planta a un punto de operación normal para una entrada constante.
- Se aplica una variación en la entrada de tipo escalón hasta un rango del 10 al 20% del rango completo.
- Registrar la salida hasta que se estabilice en el nuevo punto de operación.

Los parámetros del modelo se calculan con las siguientes ecuaciones matemáticas y con la gráfica de respuesta a escalón del sistema:

$$K_0 = \frac{y_\infty - y_0}{y_\infty - u_0} \quad ; \quad \tau_0 = t_1 - t_0 \quad ; \quad v_0 = t_2 - t_1$$

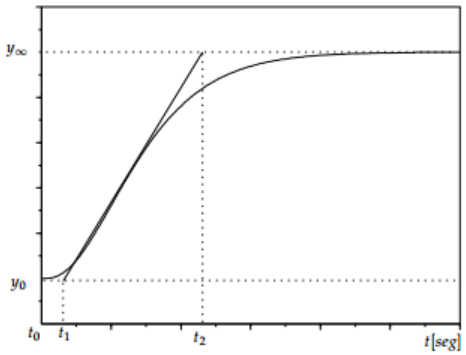


Figura 15. Respuesta ante entrada escalón.

Los valores recomendados de sintonización son:

	K_P	T_i	T_d
P	$v_0/(K_0\tau_0)$		
PI	$0.9v_0/(K_0\tau_0)$	$3\tau_0$	
PID	$1.2v_0/(K_0\tau_0)$	$2\tau_0$	$0.5\tau_0$

Tabla 3. Sintonización PID mediante curva de reacción.

3.4.3 Método de Tyreus y Luyben en lazo cerrado

Este método, similar al método Ziegler-Nichols, evalúa los parámetros del controlador a partir de la ganancia y del periodo de oscilación de la respuesta ante escalón del sistema. La principal diferencia con el otro método es que este propone un ajuste más suave que el Z-N y se aplica principalmente a plantas que contengan un integrador.

	K_P	T_i	T_d
PI	Kc/3,2	τ/0,45	0
PID	Kc/2,2	τ/0,46	τ/6,3

Tabla 4. Sintonización PID mediante Tyreus y Luyben.

Otros métodos que se emplean para la sintonización de controladores PID son los siguientes:

- Controlador PID “serie”
- Controlador PID “paralelo”
- Método en lazo abierto de Cohen y Coon

3.5 Unidad de medida inercial (IMU).

3.5.1 Introducción

Las unidades de medida inercial (IMU, Inertial Measurement Unit) son dispositivos electrónicos que miden e informan acerca de la velocidad y orientación de un dispositivo. Para ello usan la información combinada de diversos sensores, principalmente magnetómetros, acelerómetros y giroscopios. Estos dispositivos son el componente principal de los sistemas de navegación usados en aviones, naves espaciales, buques o misiles guiados.

La mayoría de los IMU´s parten de la combinación de un acelerómetro en tres ejes y un giroscopio en tres ejes. Ambos dispositivos se complementan muy bien para combinar la información, ya que entre ambos se compensan las limitaciones. Estas limitaciones son:

- Los acelerómetros no tienen deriva (drift) a medio o largo plazo. Esto es debido a que realizan la medición absoluta del ángulo que forma el sensor con la dirección vertical marcada por la gravedad. Sin embargo, son muy sensibles a los movimientos del sensor, lo cual puede introducir una gran cantidad de ruido de alta frecuencia en la medida, haciendo que las medidas no sean fiables a corto plazo.
- Los giroscopios funcionan bien para movimientos cortos o bruscos. El problema es que estos miden la velocidad angular, por lo que es necesario obtener el ángulo por integración respecto al tiempo, resultando en la acumulación de los errores y el ruido en la medición. Este es el motivo por el que aparece el error (drift) a medio o largo plazo.

Como vemos, las características de medición entre los acelerómetros y los giroscopios son opuestos, pero se complementa muy bien entre sí por lo que al combinarlos es posible obtener mejores funcionalidades que usándolos por separado. Por tanto, para poder utilizar una IMU correctamente, será necesario combinan las mediciones de ambos dispositivos. Para combinar la información de ambos sensores se suelen emplean principalmente dos métodos diferentes:

- Filtro complementario: basado en la combinación de un filtro paso alto y paso bajo, este filtro presenta como principales ventajas su fácil implementación y bajo coste computacional [13].
- Filtro Kalman: algoritmo desarrollado en 1960 por Rudolf E. Kalman que permite combinar la información de diferentes fuentes ruidosas minimizando el error a través de un bucle predictivo. Aunque se obtienen medidas mejores, su coste computacional es mucho más elevado [14][15].

El objetivo final de este proceso será obtener una medida lo más realista posible de los ángulos de navegación [16] del objeto. Estos ángulos vienen definidos al resolver el problema de determinar la posición y orientación de un objeto respecto a una base en un espacio tridimensional. Para ello es necesario definir tres parámetros para su posición relativa, así como otros tres más para determinar su orientación. Los ángulos de navegación, o ángulos de Tait-Bryan, permiten expresar la orientación relativa de ambos sistemas, describiendo la orientación de un objeto mediante tres rotaciones ortogonales en torno al eje X (roll), Y (pitch) y Z (yaw).

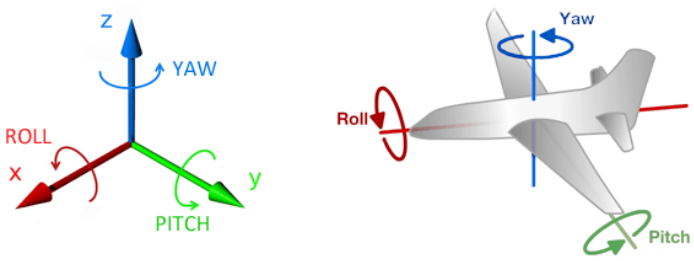


Figura 16. Ángulos de navegación. Fuente

3.5.2 Acelerómetro

Uno de los elementos que utiliza los IMU´s para el cálculo de la posición y la orientación de un objeto en un espacio tridimensional es el acelerómetro. Este instrumento está destinado a obtener medidas absolutas de las aceleraciones en tres ejes: X, Y y Z.

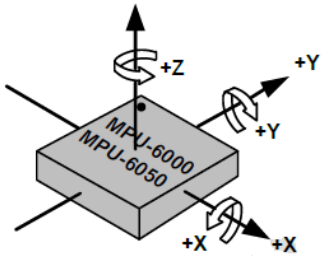


Figura 17. Ejes del sensor MPU6050. Fuente

Para recordar, sabemos que la aceleración es la variación de la velocidad respecto el tiempo:

$$\vec{a} = \frac{\partial \vec{\vartheta}}{\partial t} = \frac{\partial^2 \vec{\vartheta}}{\partial t^2}$$

Y la primera ley de Newton dice:

"Todo cuerpo persevera en su estado de reposo o movimiento uniforme y rectilíneo a no ser que sea obligado a cambiar su estado por fuerzas impresas sobre él."

$$\vec{F} = m \cdot \vec{a}$$

Esto es, cualquier cuerpo que tenga una masa m cualquiera, requiere una fuerza para poder variar su velocidad. De manera equivalente, un cuerpo que esté sometida a una aceleración experimentara cierta fuerza. Esto es lo que se emplea para la construcción de los dispositivos capaces de medir aceleración.

A la hora de la fabricación de un sensor de aceleración podríamos construirlo mediante un cuerpo sólido, en cuyo interior se suspende una masa sujeta por muelles al cuerpo exterior. Si se aplica una aceleración al conjunto, la masa suspendida se desplaza ejerciendo una fuerza sobre los muelles. Esta fuerza provoca que uno de estos muelles se estire, y por consiguiente el otro se contraiga.

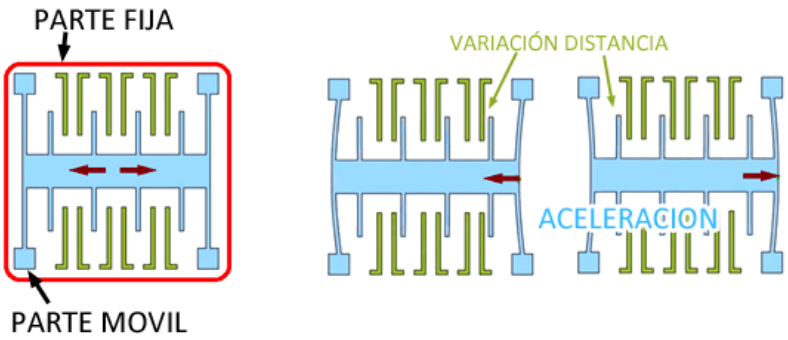


Figura 18. Descripción de un acelerómetro. Fuente

Para construir este acelerómetro en un MEMS (Microelectromachanical System) [17], se emplea una estructura micromecanizada de polisilicio construida sobre una oblea de silicio. Los muelles de polisilicio suspenden la estructura sobre la oblea y le proporciona resistencia para soportar las aceleraciones.

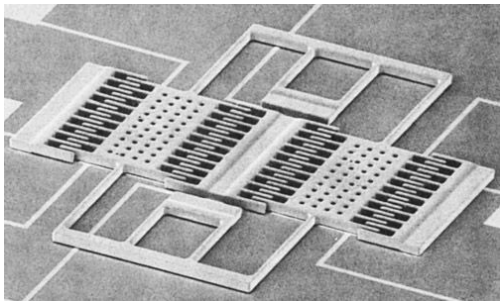
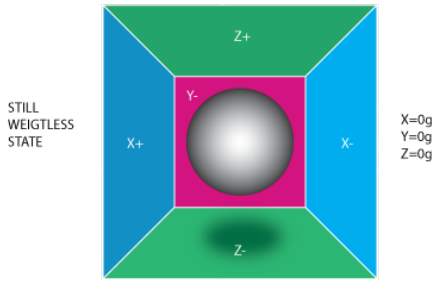
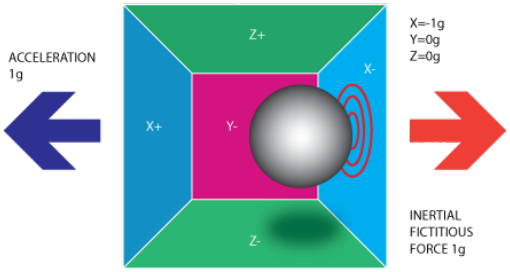


Figura 19. Sensor de aceleración (MEMS). Fuente

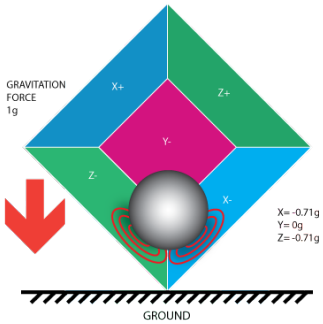
Cuando el dispositivo se somete a aceleraciones la parte interior (puntos) se deforma y se desplaza respecto a la parte fija, de forma similar al dispositivo imaginario basado en muelles. El desplazamiento es detectado a través de la variación de la capacidad del sistema. Por otra parte, los acelerómetros disponibles normalmente son de 3 ejes, por lo que son capaces de medir la aceleración a la que está sometido el sensor en X, Y y Z independientemente. De esta manera se puede conocer simultáneamente la magnitud y dirección de la aceleración medida de forma absoluta. Para entender esto, supondremos que el acelerómetro es un cubo con una bola en su interior. Cada cara del cubo será una dirección del espacio y la cara opuesta representara la dirección negativa del espacio. Si nos situamos en un lugar que no esté afectado por campos gravitatorios, la aceleración a la que está sometida la bola del interior será cero, y por consiguiente la aceleración en los tres ejes del espacio también será cero.



Si movemos el cubo a la izquierda con una aceleración de 1g (g=9,81m/s^2), la bola se apoyara sobre la cara X$^-$ y el valor de la aceleración en el eje X = -1g.



De la misma manera, al volver a la superficie terrestre, si colocamos el acelerómetro inclinado 45 grados a la derecha, la bola tocara dos caras a la vez.



Aunque este concepto es útil para entender como el acelerómetro interactúa con fuerzas externas, es más interesante para el cálculo del angulo considerar el eje de coordenadas fijado a los ejes del acelerómetro y que el vector fuerza rota sobre el origen del centro de coordenadas.

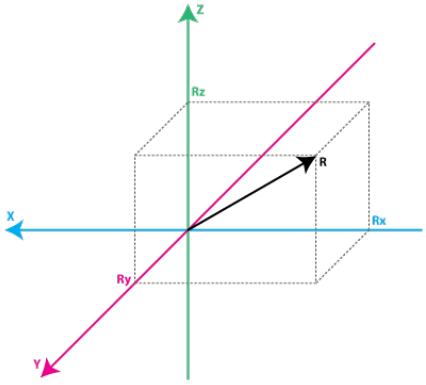


Figura 20. Descomposición de la aceleración en 3 ejes. Fuente

Con esto se puede pasar a obtener el ángulo que forma el vector con los ejes del acelerómetro y poder obtener el ángulo de nuestro sensor. Para ello basta con aplicar las siguientes formulas:

$$\theta_x = \tan^{-1}\frac{A_x}{\sqrt{A_y^2 + A_z^2}}$$

$$\theta_y = \tan^{-1}\frac{A_y}{\sqrt{A_x^2 + A_z^2}}$$

Sin embargo un acelerómetro no es un sensor adecuado para determinar la velocidad de un sistema, y mucho menos su posición, por lo que se debe emplear otra herramienta para obtener el ángulo de inclinación.

3.5.3 Giroscopio

Un giroscopio es un dispositivo que permite medir el ángulo de rotación girado por un determinado mecanismo. Estos dispositivos, a diferencia del acelerómetro, son dispositivos puramente diferenciales, por lo que siempre se miden ángulos relativos a una referencia arbitraria.

Los giroscopios que se emplean en MEMS son los denominados giroscopios de efecto coriolis (CVG). El efecto coriolis es una fuerza que aparece sobre un cuerpo en movimiento cuando se encuentra en un sistema en rotación. Esta fuerza es:

$$\vec{F}_c = -2m(\vec{\omega} \times \vec{\vartheta})$$

El funcionamiento de estos sensores CVG es que un objeto vibratorio tiende a vibrar en el mismo, aunque este plano rote. Por efecto coriolis, el objeto vibratorio ejerce una fuerza y midiéndola se puede determinar la rotación a la que está sometido el giroscopio.

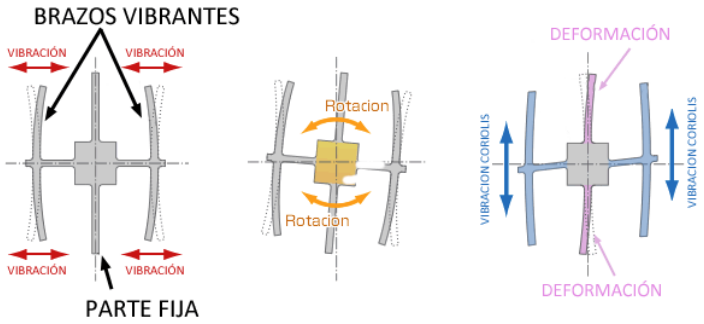


Figura 21. Funcionamiento del giroscopio. Fuente

Para conseguir registrar el efecto de la fuerza Coriolis, un MEMS dispone de estructuras parecidas a las vistas con el acelerómetro. Unas partes se someten a vibraciones debido a la rotación del giroscopio y el efecto que ejerce la fuerza de Coriolis deforma la estructura, o cual provoca una variación de la capacitancia del sistema que se puede medir.

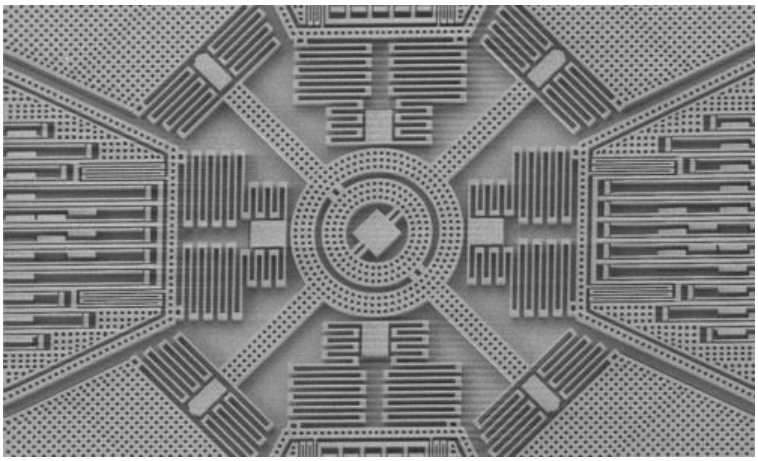


Figura 22. Giroscopio MEMS. Fuente

De la misma manera que con el acelerómetro, el giroscopio nos proporciona información de tres ejes, por lo que obtenemos de forma independiente la rotación en X, Y y Z, es decir, en los tres ángulos de navegación. Sin embargo, al emplear la fuerza de Coriolis para realizar la medida, los giroscopios vibratorios no registran el ángulo girado, sino la velocidad angular, que es la variación del ángulo respecto al tiempo:

$$\omega = \frac{\partial \theta}{\partial t}$$

Si se desea obtener la información del ángulo de posición del sensor, será necesario realizar la integración respecto al tiempo, con lo que obtenemos:

$$\theta_{gyro} = \omega_{gyro} \cdot \Delta t$$

El principal problema en este tipo de giroscopios de vibración proviene de este cálculo. Al tener que obtener las mediciones mediante una integración, se producirá una acumulación de errores en la medida, lo que causa una deriva en la medición (drift) a medio y largo plazo respecto a su valor real (incluso con el sensor estático). Sin embargo, la ventaja que tienen respecto a los acelerómetros es que son sensores con una respuesta rápida y elevada precisión en tiempos cortos, además de presentar una buena respuesta a cambios bruscos y tener una inmunidad al ruido decente. Sin embargo, esto es válido únicamente para intervalos pequeños de tiempo.

3.5.4 Filtro complementario

Como hemos visto, el giroscopio de una IMU permitirá realizar medidas rápidas y precisas pero presentará un drift en la medida. Por otro lado, el acelerómetro tiene una respuesta sin drift pero se ve muy influenciado por el ruido. Para compensar ambos errores, se puede usar la medición obtenida por el giroscopio para tiempos cortos y realizar la corrección de la deriva con la medición realizada por el acelerómetro en tiempos largos. Esto lo conseguimos mediante un filtro complementario.

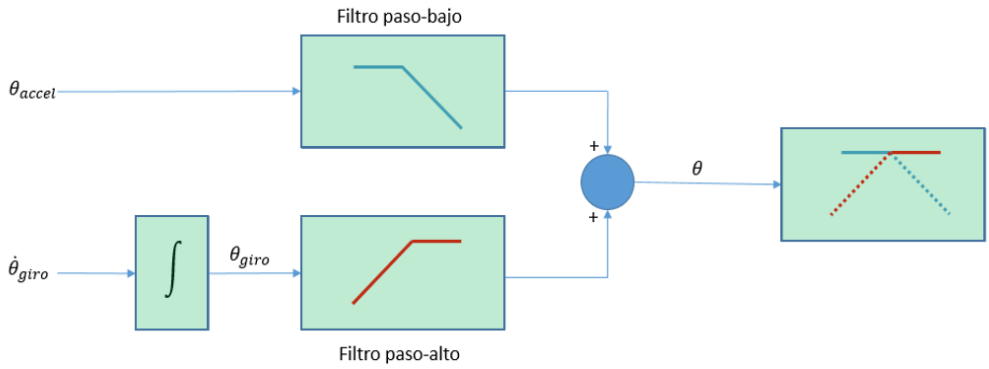


Figura 23. Esquema del filtro complementario.

El filtro complementario es un filtro de Kalman de estado estacionario para una cierta clase de problemas de filtrado. Como puede verse en la Figura 23, la idea básica del filtro complementario es combinar las dos salidas obtenidas con el acelerómetro y el giroscopio para obtener una buena estimación del ángulo de orientación. El filtro se comportará como un filtro paso-bajo para la medida del acelerómetro, y como un filtro paso-alto para la medida del giroscopio. De esta manera las componentes de alta frecuencia del acelerómetro están dominadas por el giroscopio. La forma matemática de implementar este filtro es:

$$angulo = FC * angulo\_gyro + (1 - FC) * angulo\_accel$$

dónde:

- angulo_gyro: es el ángulo obtenido mediante el giroscopio.
- angulo_accel: es el ángulo obtenido mediante el acelerómetro.
- FC = es una constante definida por el usuario.

En la Figura 24 puede observarse un ejemplo de aplicación del filtro complementario. Como se puede ver, la medida únicamente del giroscopio presenta una deriva lenta en el tiempo que hace que las medidas sean válidas únicamente a corto plazo. Por otra parte, la medida del acelerómetro presenta una gran cantidad de ruido, aunque sí que permanecen estables a largo plazo. Mediante la aplicación de un filtro complementario se consigue eliminar ambos efectos, recuperándose una señal sin ruido pero estable en el tiempo.

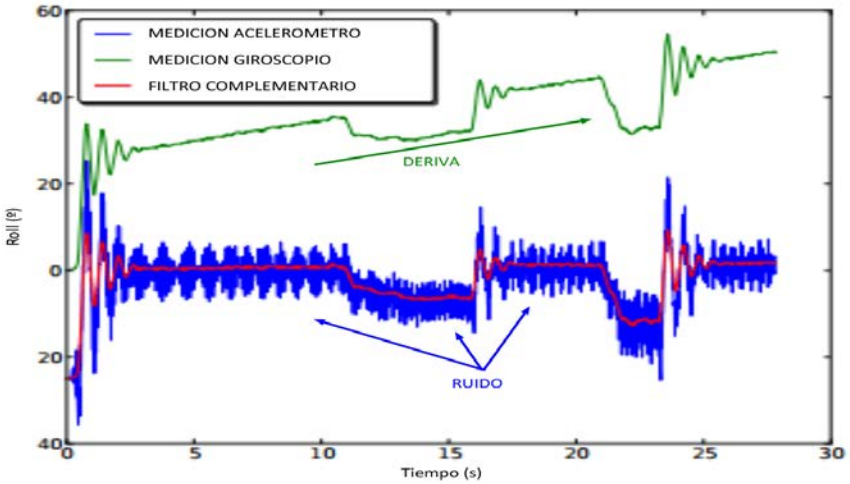


Figura 24. Ejemplo de deriva producida por el giroscopio y ruido del acelerómetro. Fuente

3.6 Motores de corriente continua

3.6.1 Introducción

El motor de corriente continua es una maquina eléctrica que convierte la energía eléctrica en energía mecánica, de manera que genera un movimiento rotatorio gracias a la acción producida por el campo magnético. Este tipo de motores también se denominan como motor de corriente directa, motor CC o motor DC.

Este tipo de máquinas fueron las primeras en utilizarse para la producción de energía eléctrica a gran escala, pero fueron sustituidas por las máquinas de corriente alterna debido a su mayor eficiencia. El modo de funcionamiento característico de las máquinas de corriente continua es como motor, ya que presentan un control de par y velocidad muy buenos. Todos los motores DC están formados por dos partes bien diferenciadas, el estator y el rotor:

- El estator (o inductor) es el encargado de dar tanto el soporte mecánico como de contener los polos de la máquina. Estos polos pueden ser creados mediante un devanado de hilo de cobre o mediante imanes permanentes.

Figura 25. Estator devanado bobinado. Fuente

- El rotor (o inducido) generalmente tiene forma cilíndrica y está unido al eje del motor, por lo que es el elemento que gira al transformarse la energía eléctrica en energía mecánica. En él se sitúa un devanado que se alimenta con corriente continua mediante el uso de escobillas, por lo que es la parte más frágil del motor, ya que están sometidas a desgaste.

Figura 26. Rotor con colector de delgas. Fuente

3.6.2 Principio de funcionamiento.

Supongamos que hay un rotor con una espira funcionando como generador. En este caso se obtiene que al girar el rotor, el área de la espira varia y se produce una variación del campo magnético que atraviesa dicha espira. Como el campo magnético varía en función del tiempo, también variara el flujo magnético produciendo una tensión en bornes de la espira.

$$e = (\vec{\vartheta} \wedge \vec{B})\,\vec{l}$$

- $\vec{\vartheta}$ = *Velocidad tangencial del rotor.*
- $\vec{B}$ = *Densidad de flujo.*
- $\vec{l}$ = *Longitud de conductor.*

Figura 27. Una espira funcionando como generador. Fuente

Si se calcula la fuerza electromotriz inducida (FEM) en cada una de las partes de la espira, se obtiene que la FEM se comporta como una señal cuadrad de valor máximo igual:

$$|e| = |2 \cdot \vartheta \cdot B \cdot l|$$

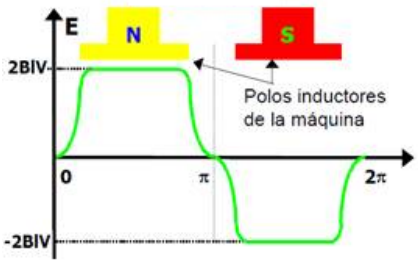


Figura 28. Tensión generada mediante una espira. Fuente

Mediante el colector de delgas y las escobillas, se consigue que la tensión tenga el mismo modulo y signo, de manera que se consigue rectificar la señal y obtener un valor de FEM continua.

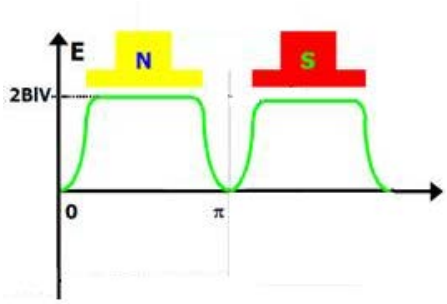


Figura 29. Tensión tras el colector de delgas. Fuente

Los motores de corriente continua están formados no únicamente por una sola espira, sino que están compuestos por varias espiras que forman el devanado del rotor. Por ello, la posición de las escobillas respecto al colector de delgas se deben colocar de forma precisa ya que sino, no se consigue rectificar la tensión. Para evitar este problema, las escobillas se colocan de forma que el paso por cero de la tensión, coincida con el cambio de una espira a otra. Si el motor tiene N_C espiras, entonces:

$$\boldsymbol{e = 2 \cdot N_C \cdot \vartheta \cdot B \cdot l}$$

Desarrollando la ecuación anterior se llega a la ecuación que relaciona la FEM inducida en un motor de corriente continua y la velocidad de giro:

$$\vartheta = r \cdot \omega$$

$$B = \frac{\emptyset}{A_p}$$

$$A_p = \frac{2\pi r l}{n} = \pi r l$$

$$e = 2\, N_C \omega r \frac{\emptyset}{\pi r l} l = \frac{2N_C}{\pi} \emptyset \omega$$

$$\boxed{e = K \cdot \emptyset \cdot \omega}$$

Figura 30. Ecuación de un motor de corriente continua.

dónde:

- r = radio.
- A_p = Área polar.
- n = Nº de polos.
- N_C = Nº de espiras.

3.7 Encoders

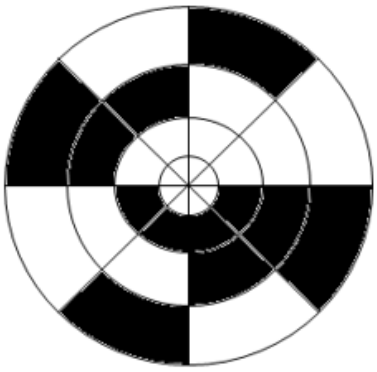
Los encoders, conocidos también como codificadores o decodificadores, son dispositivos electromecánicos que permiten convertir la posición angular de un eje a un código digital. Esta señal digital se puede procesar con un sistema de control de manera que se puede controlar la posición de un eje. Estos dispositivos se utilizan en diversas aplicaciones, ya sea en robótica, cámaras fotográficas, ratón del ordenador… Existen principalmente dos tipos de encoders según su diseño básico: encoders absolutos y encoders incrementales.

3.7.1 Encoders absolutos

Este tipo de encoder proporciona un código digital para cada ángulo del eje, lo que permite conocer la posición en cualquier momento, incluso en el encendido. Mediante un microprocesador es posible leer el código y determinar el ángulo de giro de forma absoluta. Para ello, se emplean principalmente dos códigos de codificación: codificación en código binario y codificación en código Gray.

- Codificación en código binario: Dependiendo del número de bits del encoder, se pueden obtener diferentes secciones de giro según:

$$n = Nº\ bits \quad \rightarrow \quad Sector = 2^n$$



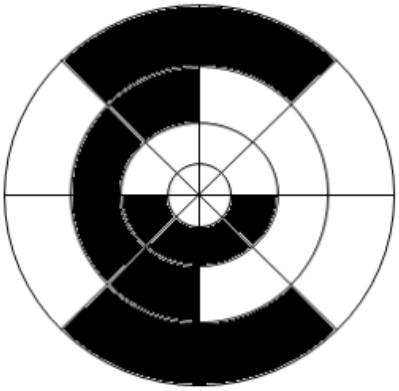
Sector	Código Binario			Ángulo
1	0	0	0	0º-45º
2	0	0	1	45º-90º
3	0	1	0	90º-135º
4	0	1	1	135º-180º
5	1	0	0	180º-225º
6	1	0	1	225º-270º
7	1	1	0	270º-315º
8	1	1	1	315º-360º

Tabla 5. Encoder y divisiones en código binario de 3 bits.

El problema que presenta este tipo codificación binaria es que si los datos digitales no se toman a la vez, al producirse un cambio de sector los bits que nos dan la información no se actualizan a la vez, puede dar a error a la hora de calcular la posición del eje.

- Codificación en código Gray: Dependiendo del número de bits del encoder, se pueden obtener diferentes secciones de giro según:

$$n = Nº\ bits \quad \rightarrow \quad Sector = 2^n$$



Sector	Código Gray			Ángulo
1	0	0	0	0º-45º
2	0	0	1	45º-90º
3	0	1	1	90º-135º
4	0	1	0	135º-180º
5	1	1	0	180º-225º
6	1	1	1	225º-270º
7	1	0	1	270º-315º
8	1	0	0	315º-360º

Tabla 6. Encoder y divisiones en código Gray de 3 bits.

Para evitar el problema existente con el código binario se diseñó el código Gray, que o es más que una forma de codificación binaria en la que solo cambia un bit por cambio de sector. Para el caso de un encoder de 3 bits el código Gray queda como se ve en la tabla anterior.

3.7.2 Encoder incremental

El encoder incremental, o también conocido como codificador rotativo incremental, es un dispositivo, generalmente mecánico, óptico o magnético, que se encuentran principalmente en equipos de sonido (tanto domésticos como automóviles) debido a su bajo coste. Este tipo de encoders proporcionan una señal muy simple de fácil interpretación que proporciona información tanto de velocidad, como del sentido de giro. Para ello se generan dos señales cuadradas, y una de ellas desfasada 90º respecto a la otra. Así se consigue:

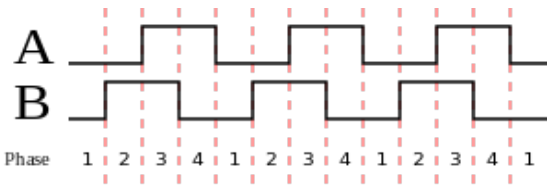


Figura 31. Salidas digitales de las dos fases de un encoder incremental.

Mirando los bits recibidos de las dos fases, se puede conocer el sentido de giro del encoder, con la figura vista arriba se obtiene la siguiente tabla:

Giro Horario		Giro Anti-horario	
Phase A	Phase B	Phase A	Phase B
0	0	0	1
0	1	0	0
1	1	1	0
1	0	1	0

Tabla 7. Sentido de giro de los encoder incrementales.

4 Diseño mecánico

4.1 Introducción

Para la fabricación del robot será necesario diseñar y construir una estructura donde poder colocar los distintos componentes del mismo. Además, este diseño ha de ser modular y fácilmente modificable ya que, al emplearse como banco de pruebas, será necesario poder realizar modificaciones al mismo de forma sencilla. Por ello se optó por emplear la impresión 3D, realizándose el diseño 3D del robot mediante el software de diseño mecánico en 3D SolidWorks.

Tras investigar sobre otro tipo de robots autobalanceados, como se ve en el estado del arte, se buscó diseñar una estructura resistente y ligera. Además, el diseño tiene que permite el fácil reemplazo de una pieza en caso de rotura, por lo que todo el robot está constituido por piezas independientes. La elección de la estructura estuvo basada en su sencillez y en la ubicación de su centro de gravedad, el cual se encuentra muy bajo para dotarle de una mayor estabilidad. Las ruedas fueron diseñadas con un diámetro superior a lo normal para mejorar también la estabilidad. Además, para impedir que patinen sobre el suelo, ambas ruedas están forradas con una goma, obteniéndose por tanto un rozamiento bastante alto. Por último la plataforma inferior y media están pensadas para colocar la pila y la PCB, mientras que la plataforma superior permite colocar diferentes objetos y ver como el robot mantiene el equilibrio.

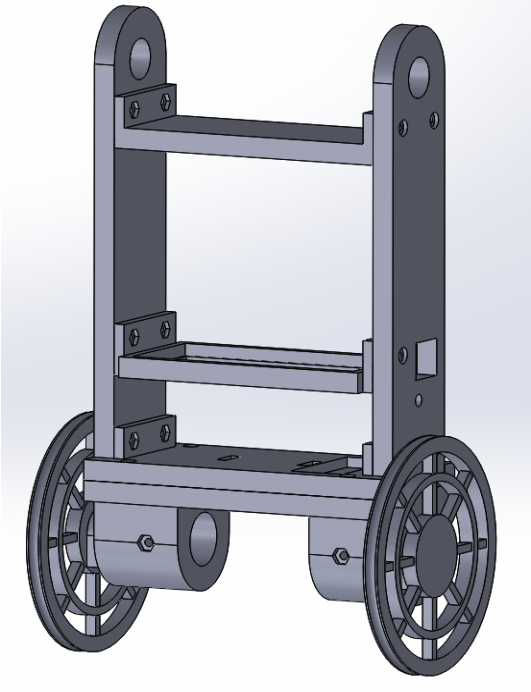


Figura 32. Diseño del robot.

4.2 Software SolidWorks

SolidWorks [18] es un software de diseño y modelado mecánico en 3D, desarrollado por SolidWorks Corp., una filial de Dassault Systémes, S.A. (Francia). Se trata de un software CAD (diseño asistido por computadora) para el sistema operativo Microsoft Windows.

Figura 33. Logo SolidWorks. Fuente

El programa permite modelar piezas y conjuntos además de extraer los planos técnicos como diferente tipo de información necesaria para la producción. Otra herramienta interesante que incorpora SolidWorks, y que es muy útil para nuestro proyecto, es la posibilidad de generar documentos STL, los cuales son los empleados habitualmente para la impresión 3D. Además de estas ventajas, hay otra muy importante, y es que existe una gran cantidad de documentos y tutoriales en internet, tanto del propio desarrollador como de personas ajenas a él, lo cual resulta de mucha ayuda a la hora de realizar el diseño.

A continuación, se explicarán los pasos que se han dado para realizar el diseño de la estructura del robot:

1. Una vez se ha ejecutado el programa de SolidWorks, aparece una ventana en la cual se elige que tipo de documento se desea realizar. Las opciones que nos presenta el programa son, la creación de una pieza desde cero, realizar un ensamblaje de diferentes piezas 3D y la realización de un dibujo en 2D.

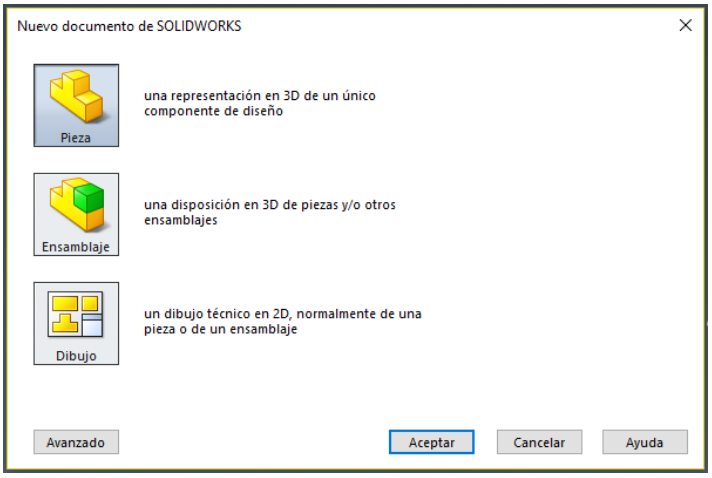


Figura 34. Ventana de creación de un nuevo documento de SolidWorks.

2. Una vez dentro del programa, se nos presentan diferentes opciones para la realización de una pieza. Lo primero es realizar un croquis de la pieza deseada con sus dimensiones con ayuda de las múltiples herramientas de dibujo que el software presenta.

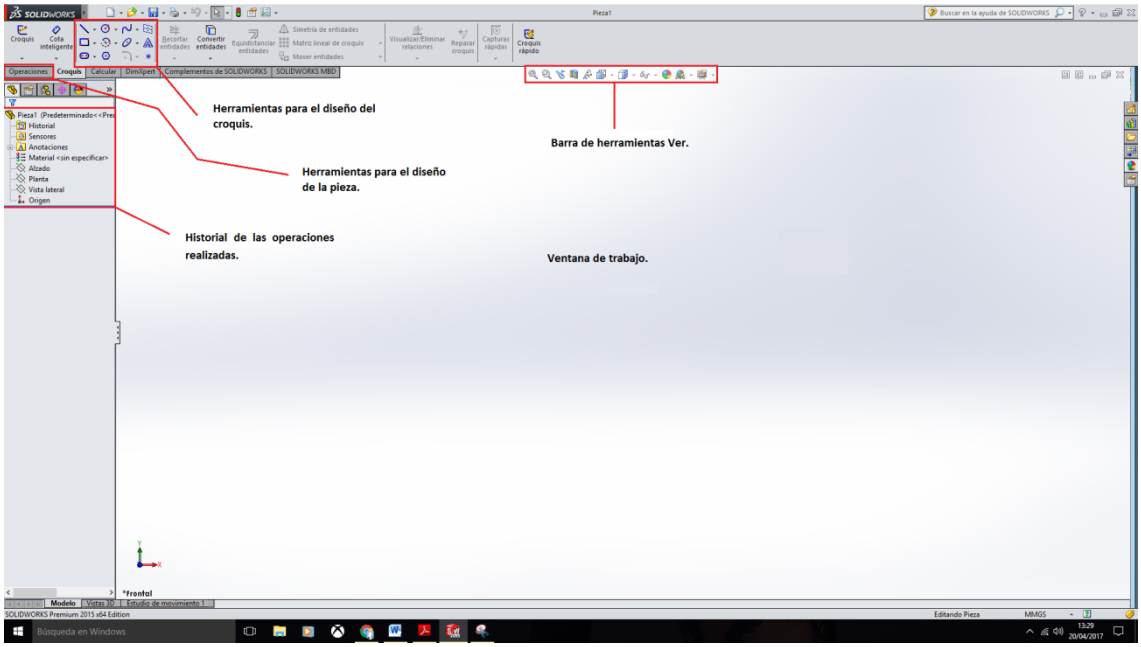


Figura 35. Ventana de trabajo de SolidWorks.

3. Tras definir el croquis que se desea, para poder crear la pieza en 3D hay que someter a la pieza a la operación de “Extruir saliente/base” con un determinado grosor. Una vez que se tiene la pieza extruida, se le puede someter a otras operaciones como puede ser “simetría” o “Extruir corte”.

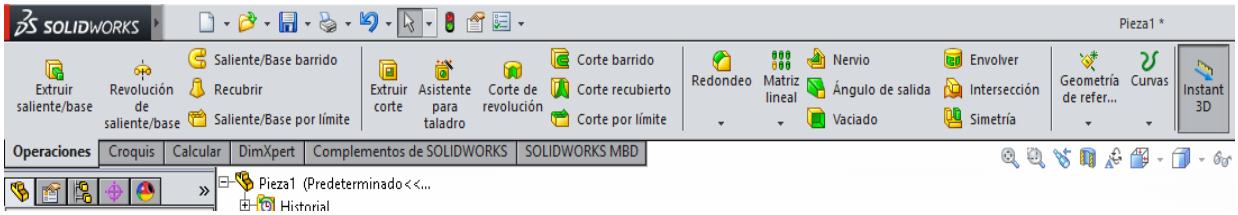


Figura 36. Ventana con las diferentes operaciones.

Realizando estas operaciones, se diseñaron las diferentes piezas que pueden verse en la Figura 37. Además, los diferentes planos con las vistas y las medidas de las diferentes piezas creadas, se pueden encontrar en el anexo “Planos”

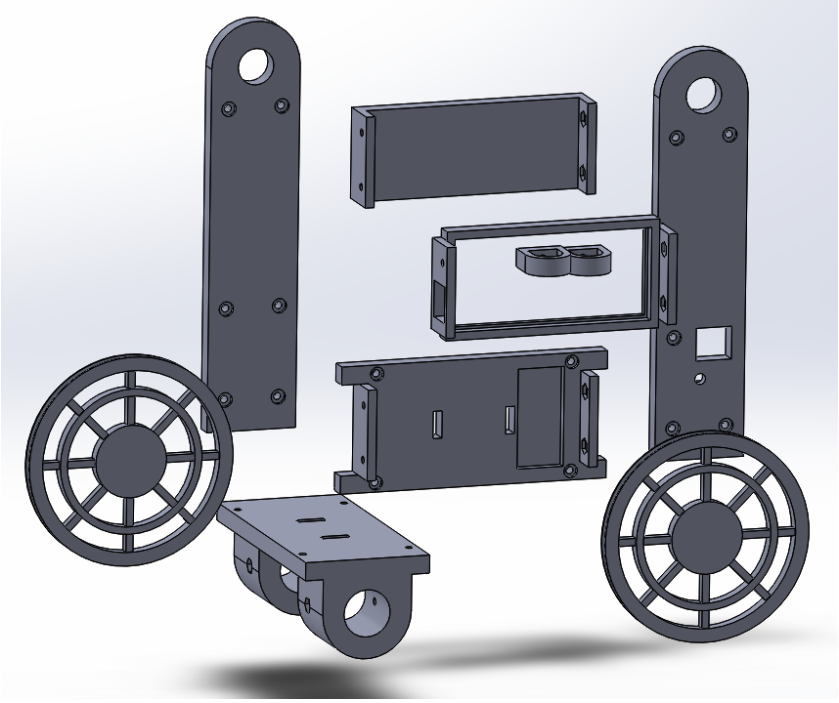


Figura 37. Diferentes piezas que componen el robot.

4. Una vez que se han diseñado todas las piezas, SolidWork nos ofrece la opción de crear un ensamblaje y comprobar como quedara la estructura final diseñada. Para ello hay que insertar todas las piezas, las cuales quedaran dispersas por la zona de trabajo.

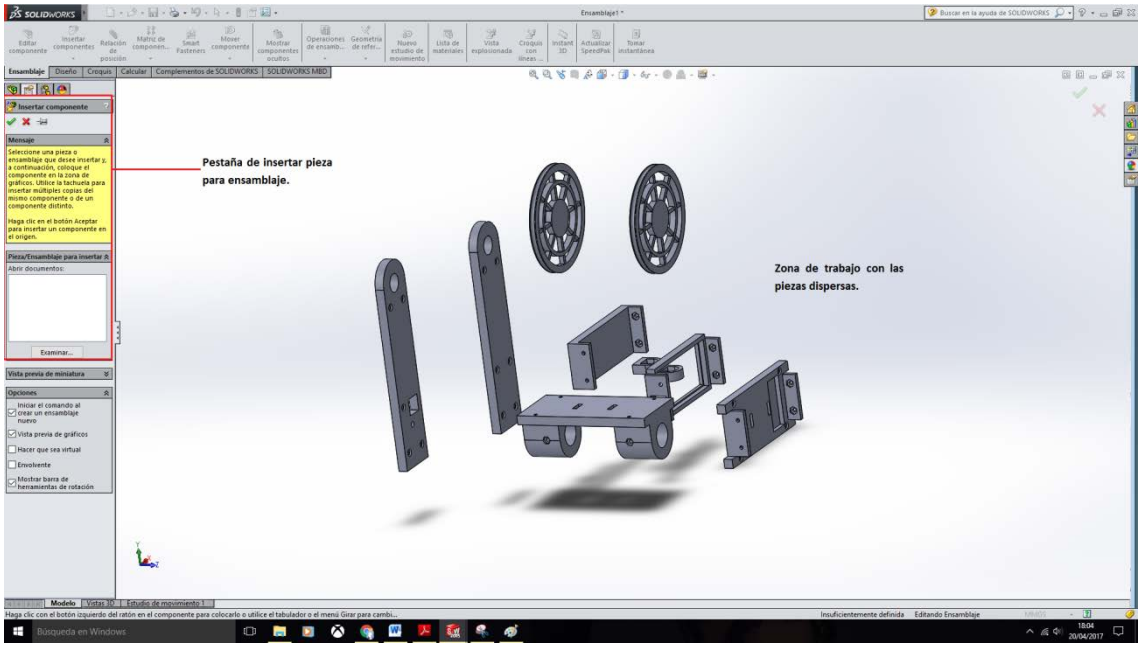


Figura 38. Disposición de los elementos para crear un ensamblaje.

5. Una vez que están todas las piezas en la zona de trabajo, se pueden ordenar mediante la opción de insertar “Relación de posición”.

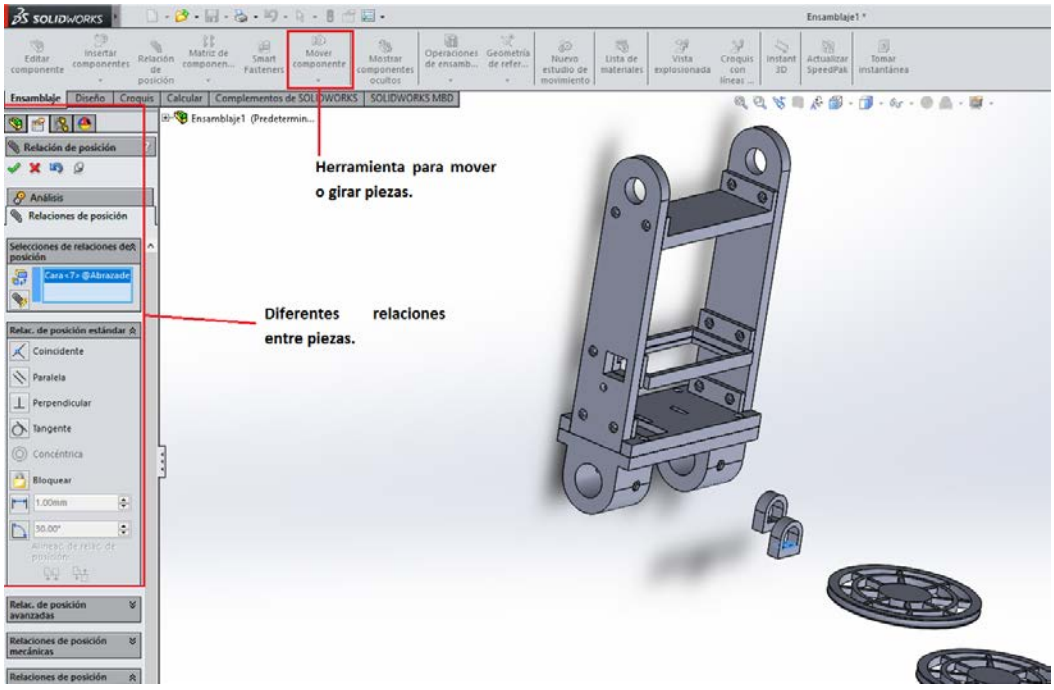


Figura 39. Operaciones para realizar el ensamblaje.

6. Por último, el resultado final es:

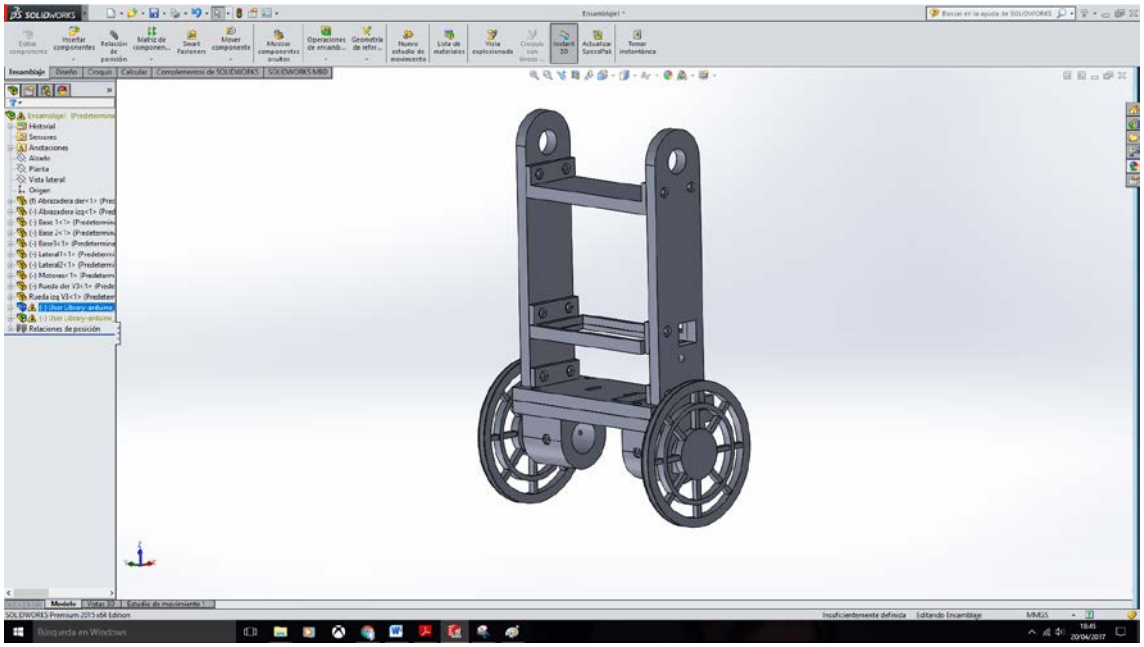


Figura 40. Diseño final del robot.

4.3 Montaje del robot

Tras realizar diversas iteraciones de diseño en las cuales se fueron depurando los defectos que se encontraron, se procedió a imprimir las piezas en 3D y al montaje, con lo que el resultado final obtenido es:

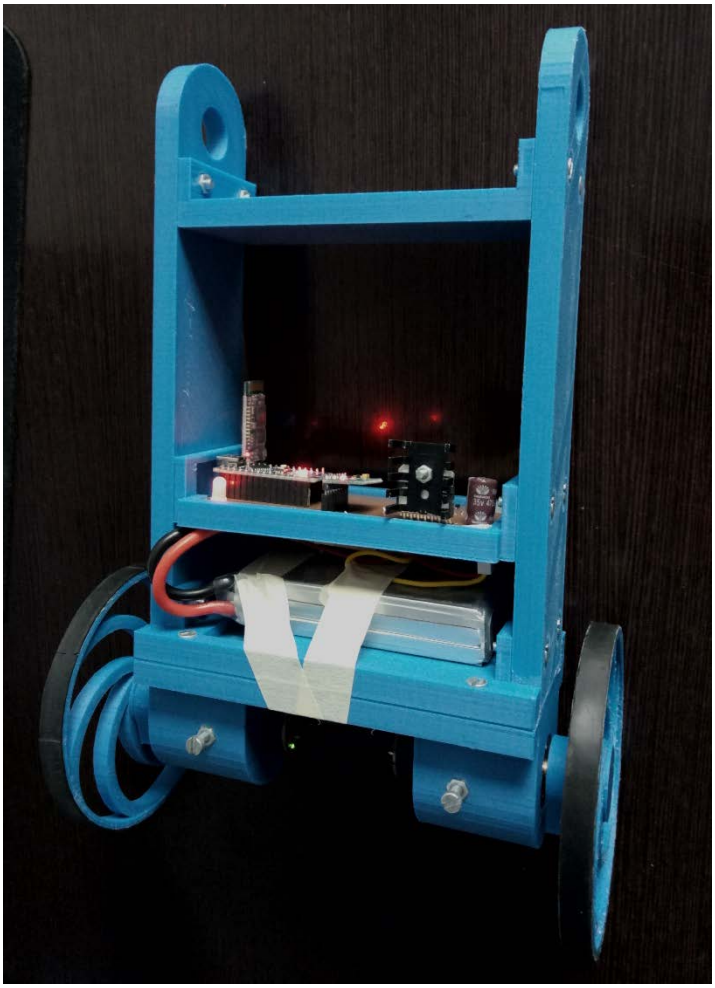


Figura 41. Robot final.

Como se ha mencionado anteriormente, los diferentes planos de las piezas acotadas, se pueden encontrar en el anexo “Planos”. En dicho anexo, se describe de forma precisa, las medidas que debe tener cada una de las piezas que forman el robot. Como se puede observar, este diseño cumple los criterios de flexibilidad y facilidad de modificaciones mencionadas, ya que la totalidad de piezas que lo componen se puedan separar en cualquier momento para realizar operaciones de mantenimiento, siendo necesario únicamente aflojar unos pocos tornillos y tuercas, Por ese motivo, el montaje resulta ser muy rápido y sencillo.

5 Diseño eléctrico

5.1 Introducción

En este apartado se tratará de explicar los pasos seguidos para la realización de una PCB, en la cual se distribuirán los diferentes elementos necesarios para que el robot pueda funcionar. Para ello nos centraremos en los tres bloques más importantes que conforman la misma:

- Bloque de alimentación: En este apartado, se busca realizar la correcta alimentación de todos los dispositivos que se van a alojar en la PCB. Para ello se deberá pasar de la tensión proporcionada por la batería a 5V constantes en todas las zonas de los diferentes circuitos. Aquí también se recoge lo necesario con el circuito de potencia.
- Bloque de señal: Se debe realizar todas las conexiones necesarias que permitan la correcta comunicación entre los sensores, actuadores y el microprocesador, con el consiguiente de que estas no se interfieran entre sí.
- Bloque de control: Se elegirá un microcontrolador, y se distribuirán todos elementos del circuito que lo necesiten entre sus diferentes pines de conexión.

5.2 Bloque de alimentación.

Esta etapa se encarga de la alimentación eléctrica de todo el robot, incluyendo la etapa de control, la etapa de sensores, además de la alimentación de los motores. Para este proyecto se necesita tener dos alimentaciones diferentes, 5V y 3,3V, para los diferentes dispositivos. Además hay que alimentar los motores mediante un driver para que el robot pueda cumplir su función de mantener el equilibrio.

5.2.1 Alimentación.

Inicialmente, la alimentación del robot se realizó mediante un módulo comercial MB102 [19] desarrollado por la compañía YwRobot Corporation, empresa dedicada a la fabricación de dispositivos electrónicos de bajo coste. Este módulo está diseñado para conectarlo en protoboards y elegir la alimentación que se desea generar, ya sea de 5V o de 3,3V. Estos dispositivos suelen ser empleados para la elaboración de pequeños proyectos, pudiendo encontrarse fácilmente en tiendas online como son eBay o Amazon a un coste muy bajo (inferior a los 2€

Figura 42. Módulo de alimentación MB102. Fuente

La alimentación de este módulo se puede realizar de dos formas: mediante el uso de una pila o batería, o con un cable USB. Para saber que está correctamente alimentado, este dispositivo incorpora un indicador led que avisara si se está usando de forma correcta. Otra característica importante de este módulo, es la posibilidad de elegir la tensión de salida que proporciona, pudiendo seleccionar una alimentación de 5V y otra de 3,3V a la vez. En la Figura 42 puede verse el esquema eléctrico del módulo.

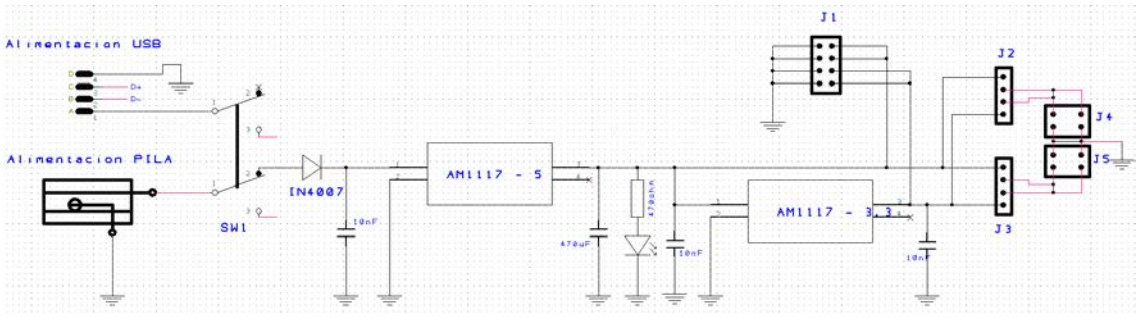


Figura 43. Esquema eléctrico del módulo de alimentación. Fuente

Como puede verse, los diferentes componentes que se encuentran en el módulo tienen la siguiente función:

- Alimentación USB: como se ve en el esquema, esta entrada solo se utiliza para la alimentación de 5 voltios mediante un conector USB hembra. Los pines centrales, D+ y D- no están conectados a ningún elemento, por lo que no puede recibir datos.

Figura 44. USB hembra. Fuente

- Alimentación PILA: se trata de un conector del tipo Jack hembra, al que se le puede colocar cualquier tipo de batería o pila que se encuentre entre los rangos de 6 y 12V permitidos por el modulo.

Figura 45. Conector Jack. Fuente:

- AM1117-5 y AM1117-3.3: son los dos reguladores de tensión que incorpora el módulo MB102 para regular la tensión de salida hasta dos valores diferentes (5V y 3,3V respectivamente). Las características y modos de conexión, además de los

condensadores necesarios para su correcto funcionamiento, pueden verse en el datasheet [20] y se implementan en la placa mediante elementos superficiales.

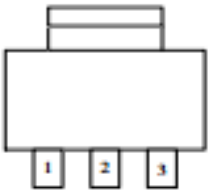


Figura 46. Empaquetado SOT-223 empleado por los reguladores de tensión. **Fuente**

- IN4007 [21]: diodo rectificador del tipo zener, para mantener la tensión que llega a los reguladores a un valor constante. Además, este diodo realiza también la función de protección ante una inversión de la polaridad a la entrada de la placa.

- Diodo LED: se conecta este led para comprobar el correcto funcionamiento del módulo, así como indicador de encendido.

- SW1: botón cuya finalidad es encender o apagar el módulo de alimentación.

- J1: serie de pines en los cuales se tiene una tensión de 5V y 3.3V para poder conectar diferentes elementos. De los ocho pines que tiene, cuatro pines son de GND, dos pines son de 5V y los otros dos pines restantes son de 3.3V.

Figura 47. Conector J1.

- J2 y J3: pines para la selección de la tensión de salida del módulo. Para seleccionar la tensión, hay que conectar un "jumper" entre los pines en función de la tensión deseada.

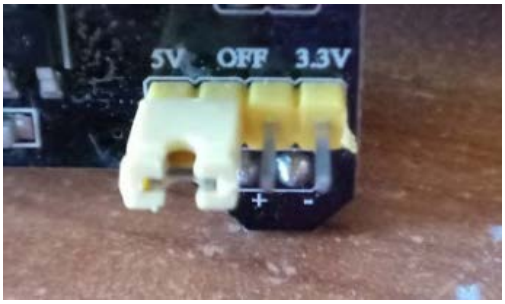


Figura 48. Conector J2 y J3.

- J4 y J5: pines que incorpora el modulo en su parte inferior para la conexión de este en una protoboard.

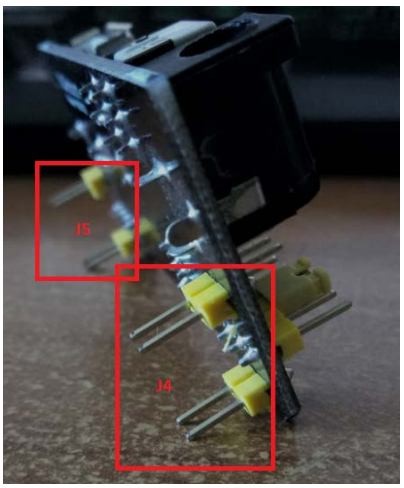


Figura 49. Conector J4 y J5.

5.2.2 Driver motores

Un driver motor [22] es un dispositivo, o grupo de dispositivos, que se utilizan para controlar de una manera predeterminada el funcionamiento de un motor eléctrico. El control se consigue mediante unas señales de entrada, ya sean analógicas o bien digitales, con las que conseguimos:

- Seleccionar el sentido de giro del motor: gracias a la utilización de un driver, podemos elegir el sentido horario o anti horario del motor de manera muy simple.
- Regular la velocidad: mediante el uso de una señal PWM, el driver permitirá de manera muy simple regular la velocidad de variación de un motor DC.
- Permite la alimentación necesaria para el correcto funcionamiento del motor.
- Protección del circuito: al incluir un driver en el circuito, nos ofrece protección contra posibles sobrecargas y fallas que puedan aparecer.

Figura 50. Driver L293DN. **Fuente**

PWM (Pulse-Width Modulation)

La modulación PWM [23], también conocida como modulación por ancho de pulsos, es una técnica basada en la modificación del ciclo de trabajo de una señal periódica (habitualmente una señal cuadrada), ya sea para transmitir información, o como es nuestro caso, para controlar la cantidad de energía que se envía a los motores. Por tanto, esta técnica permitirá regular tanto

la velocidad como el par de los motores de corriente continua. En la etapa de control se genera una señal PWM que se envía al driver, así de esta manera este regula la tensión que llega a los bornes del motor y controlamos la velocidad de estos.

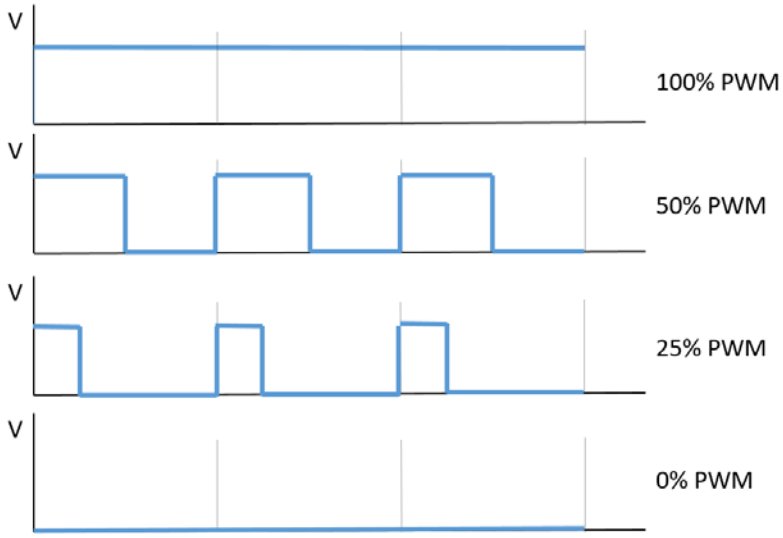


Figura 51. Regulación PWM.

Puente en H

El puente en H es el circuito electrónico que permite cambiar el sentido de giro de los motores de corriente continua. Este circuito lo componen cuatro interruptores, los cuales pueden ser mecánico o electrónicos (diodos, transistores…) de forma que, en función de cuales se abran o cierren, permitirán cambiar el sentido de alimentación de un motor. Los podemos encontrar como circuitos integrados, pero también se pueden crear con componentes discretos.

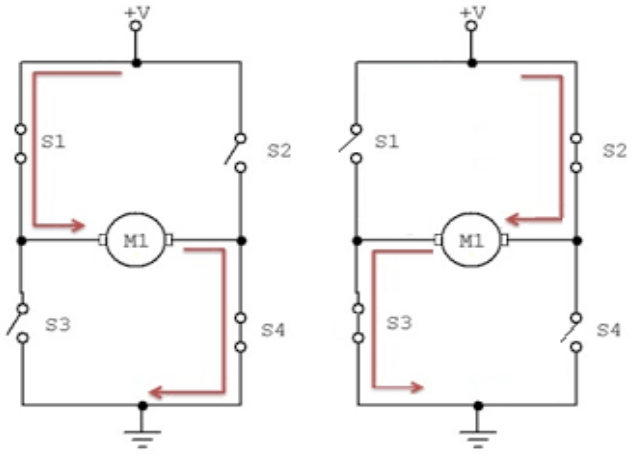


Figura 52. Funcionamiento del puente en H. Fuente

El funcionamiento de un puente en H es bastante simple. Cuando se cierran los interruptores S1 y S4, y se abren S2 y S3, se aplica una tensión positiva en bornes del motor, así que este gira en un sentido. Cuando se abren los interruptores S1 y S4 y se cierran S2 y S3, se invierte la tensión en bornes del motor, por lo que este gira en sentido contrario. Por último, en caso en que todos los interruptores están abiertos el motor no girará.

Para el proyecto que se está realizando se optó inicialmente por emplear el driver L293DN [24] de la empresa Texas Instruments. Dentro de la gran variedad de drivers que existen en el mercado, se eligió este por varios motivos:

1. Se trata de un driver y un puente en H incluidos en un encapsulado.
2. Se puede usar para controlar 2 motores a la vez, por lo que en nuestro caso es lo necesario.
3. Se trata de un dispositivo bastante pequeño y además barato (aproximadamente 2€)
4. La versión L293DN incorpora diodos de protección del motor simplificando el diseño.
5. El conexionado necesario para su uso es muy simple
6. Ya se había usado previamente, por lo que se conocía su funcionamiento.

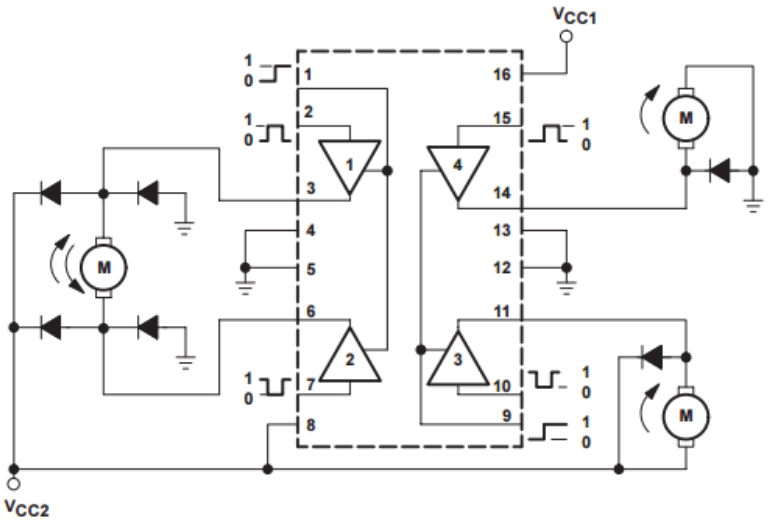


Figura 53. Driver L293D. **Fuente**

5.2.3 Alimentación versión 2.0

Tras varios días programando y probando el robot, aparecieron diversos problemas entre los que destacó el reinicio que se producía en el microcontrolador cuando el robot cambiaba de sentido de giro de las ruedas. Empleando un osciloscopio se pasó a la verificación de los diferentes elementos del robot, observándose que la alimentación del Arduino bajaba por debajo de los 3,3V al producirse estos cambios. Este transitorio era debido a la demanda de corriente que se producía en los motores para cambiar el sentido de giro, por lo que los reguladores no eran capaces de mantener la tensión a la salida y su valor bajaba de 5V a 2,5V, produciéndose el reseteo del programa.

La primera solución, y más fácil de implementar, fue colocar un condensador electrolítico de gran valor (470µF) que ayudara a la batería a proporcionar la corriente que esta no puede suministrar.

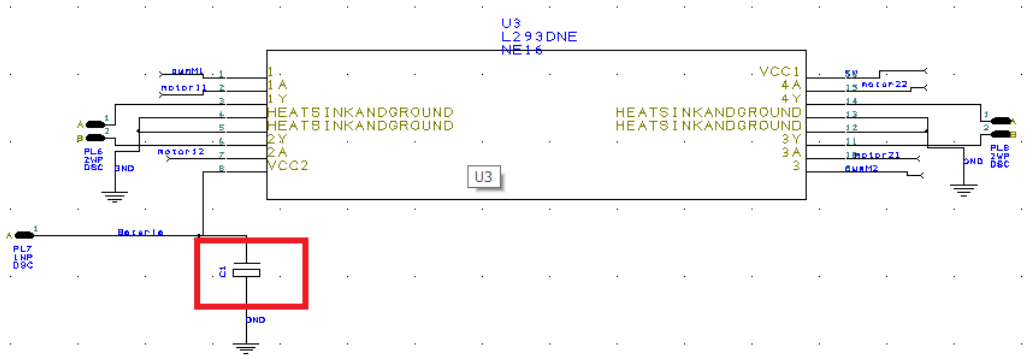


Figura 54. Diseño del Driver con el condensador.

A pesar de que la idea es buena, esta no solucionaba el problema ya que cuando la demanda de corriente era alta, se seguía produciendo una bajada tensión que desconectaba el Arduino, parándose el robot. En ese momento se decide otra solución que nos podría ayudar, se decide colocar dos baterías en paralelo, de manera que la tensión sea de 9V pero la corriente que nos proporcionan sea el doble.

Figura 55. Pilas en paralelo.

Cuando se implementaron estas soluciones, se consiguió que el robot ya no se detuviese, pero apareció otro problema ¿El driver es capaz de suministrar toda la corriente que demandan los motores o está funcionando al límite? Tras unas mediadas de corriente y consultar el datasheet del driver L293DN, se comprobó que la demanda de corriente de cada uno de los motores era muy superior a la especificada en su hoja de características, por lo que se estaba superando el rango de operación del driver. Para solucionar este problema se procedió al rediseño de la PCB, empleando un nuevo driver cuya intensidad por canal fuera superior. Además, se observó que el empleo de una placa comercial para la alimentación estaba sobredimensionado, por lo que se sustituyó por un regulador de tensión integrado en la propia placa.

Regulador LM7805

En esta segunda versión de la PCB la alimentación de todos los dispositivos se realizará empleado un regulador de tensión LM7805 de la empresa STMicroelectronics [25]. Se trata de un regulador de tensión lineal de 5V y 1A, con el que se consigue pasar de la tensión de 9V de la batería a una tensión continua de 5V estables. Dado que no existe ningún elemento que demande una corriente elevada, 1A es suficiente para alimentar todos los componentes empleados.

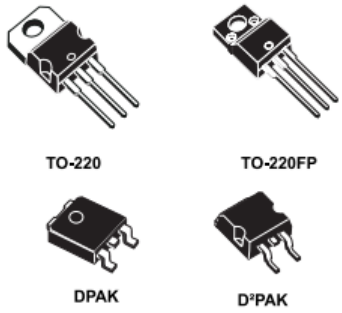


Figura 56. Regulador de tensión lineal LM7805. **Fuente**

El diseño que se realiza en DesignSpark para la correcta alimentación de la placa es muy sencillo, ya que únicamente es necesario añadir un par de condensadores a la entrada y salida del regulador. En la placa este circuito se implementó usando el empaquetado DPAK del regulador y dos condensadores superficiales.

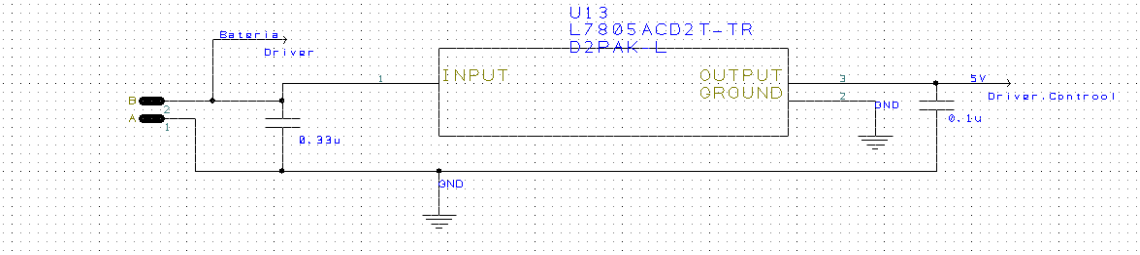


Figura 57. Esquemático de conexión del regulador LM7805.

Driver L298N

El nuevo driver que se emplea para el control y sentido de giro de los motores es el driver L298N de la empresa ST Microelectronics [26]. Su funcionamiento es similar al empleado anteriormente, pero este permite una corriente de 3A por cada canal, con lo que soporta perfectamente la corriente que demandada por los motores.

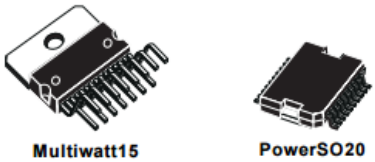


Figura 58. Driver L298N. **Fuente**

El problema que tiene este driver, es que no dispone de un puente de diodos, por lo que será necesario añadir uno a la placa que sea capaz de soportar la corriente demandada. Por otra

parte, este driver nos permite conocer la corriente que demandan los motores, ya que dispone de dos patillas de "Current Sensing", si a estas se le coloca una resistencia de potencia de 0.5Ω con tierra. Dado que no nos interesaba conocer este valor, se procedió a conectar estos pines directamente a tierra.

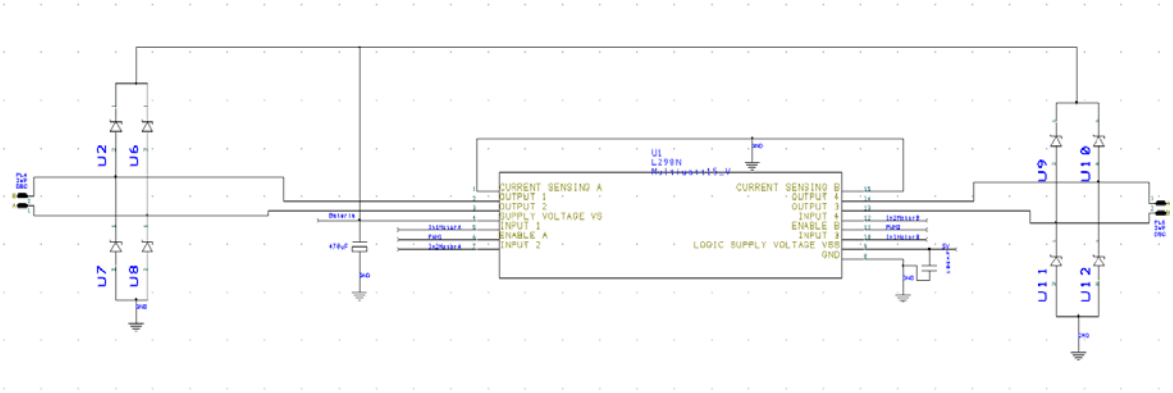


Figura 59. Esquemático de conexión del regulador L298N.

5.3 Bloque de señal

En esta etapa se le proporciona al robot la capacidad de interpretar el entorno que lo rodea mediante el empleo de los diferentes sensores que lleva incorporado. Estos le permiten conocer las variables que se pretenden controlar, y son parte de los elementos fundamentales que necesita un sistema de control en lazo cerrado. Las variables que presenta el proyecto son el ángulo de inclinación y la velocidad: el robot debe conocer cuál es su inclinación y su velocidad en todo momento para poder aplicar una acción de control que lo corrija, en función de las condiciones deseadas por el usuario.

Otra variable de entrada a controlar será la orden que debe seguir el robot. Estas instrucciones se enviarán mediante el protocolo de comunicación bluetooth desde un dispositivo Smartphone. De esta manera se podrá ordenar al robot que se desplace en una dirección determinada o se ponga a girar. Además, como se verá más adelante, esta conexión se empleó para la sintonía de los parámetros del controlador PID.

5.3.1 IMU

Como ya se ha visto en el Apartado 3.5, el dispositivo que permite conocer la inclinación del robot va a ser una IMU de 6 grados de libertad. El modelo elegido va a ser el MPU-6050 de la empresa *InvenSense*[27], ya que este dispositivo integra un acelerómetro y un giroscopio y la información resultante es fácilmente accesible mediante la interfaz de comunicación I2C.

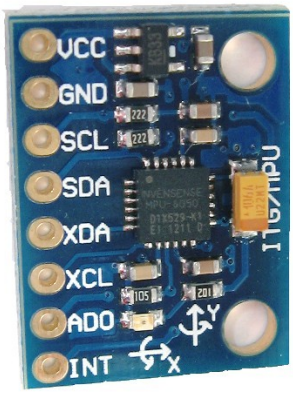


Figura 60. Sensor IMU MPU-6050. **Fuente**

El protocolo de comunicación I2C [28] es un bus serie de datos desarrollado en 1982 por *Philips Semiconductors*. Este sistema de comunicación se diseña como un bus maestro/esclavo que se emplea para la comunicación entre un microcontrolador y sus diferentes periféricos en sistema integrados.

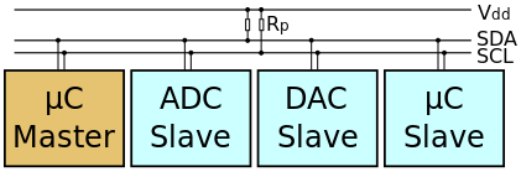


Figura 61. Comunicación I2C. **Fuente**

Como se puede apreciar, el protocolo de comunicación I2C solo precisa de dos líneas para la transmisión de la información. Con este sistema se consigue velocidades de transmisión de 0.1Mbit/s hasta los 4.3Mbit/s de las versiones más recientes, todas ellas bidireccionales. La conexión para permitir esta comunicación es la siguiente:

- SDA (Serial Data Line): Línea de datos.
- SCL (Serial Clock Line): Línea de reloj, marca los tiempos de RW (Lectura/Escritura).
- GND: Todos los dispositivos deben de tener masa común.

Como se verá más adelante, que el dispositivo disponga de una interfaz de comunicación I2C integrada permitirá el uso de la librería Wire [29] de Arduino. Esto simplificará la programación del microcontrolador, ya que en ella esta integradas todas las funciones requeridas para la comunicación en este bus.

Aparte de la comunicación I2C, una vez se reciban los datos estos deben de ser tratados para lograr obtener el ángulo de inclinación. El IMU MPU6-6050 permite dos posibilidades de obtener los ángulos diferentes:

- Obtener el ángulo mediante el procesador DMP que incorpora el sensor, el cual realiza la fusión de la información del acelerómetro y el giroscopio de manera independiente
- Obtener el ángulo mediante los valores obtenidos por el acelerómetro y el giroscopio y procesarlos en el microcontrolador

5.2.1.1 DMP (Digital Motion Processor).

EL DMP [30] es un procesador digital de movimiento que fusiona los datos del acelerómetro y giroscopio para proporcionar los ángulos de cabeceo (pitch), alabeo (roll) y guiñada (yaw) en forma de cuaterniones. El fabricante del sensor, *InvenSense,* no proporcionó hasta 2014 el algoritmo que se emplea para fusionar los datos, así que la solución que se puede emplear es usar la librería i2cdevlib[31] en la cual hay un código de ejemplo para extraer estos ángulos. A pesar de que las medidas obtenidas empleando el DMP son más precisas que las obtenidas al realizar la fusión en el microcontrolador, se optó por no empelarlo ya que requería el uso de una librería externa de una gran complejidad matemática y, por tanto, de difícil comprensión.

5.2.1.2 Fusión del acelerómetro y giroscopio.

Como ya se explicó anteriormente, empleando el acelerómetro es posible conseguir información de la aceleración a la que está sometido el sensor en sus tres ejes. Al estar en la tierra, el sensor siempre estará sometido a la aceleración de la gravedad, así que se puede emplear esto para conocer el ángulo de inclinación respecto al suelo. Como el valor de la gravedad es conocido (9.8m/s^2) y la aceleración a la que se somete la IMU también la conocemos, se puede emplear la trigonometría para el cálculo el ángulo de inclinación mediante estas fórmulas:

$$Angulo_{X\,accel} = atan\left(\frac{Y}{\sqrt{X^2 + Z^2}}\right)$$

$$Angulo_{Y\,accel} = atan\left(\frac{X}{\sqrt{Y^2 + Z^2}}\right)$$

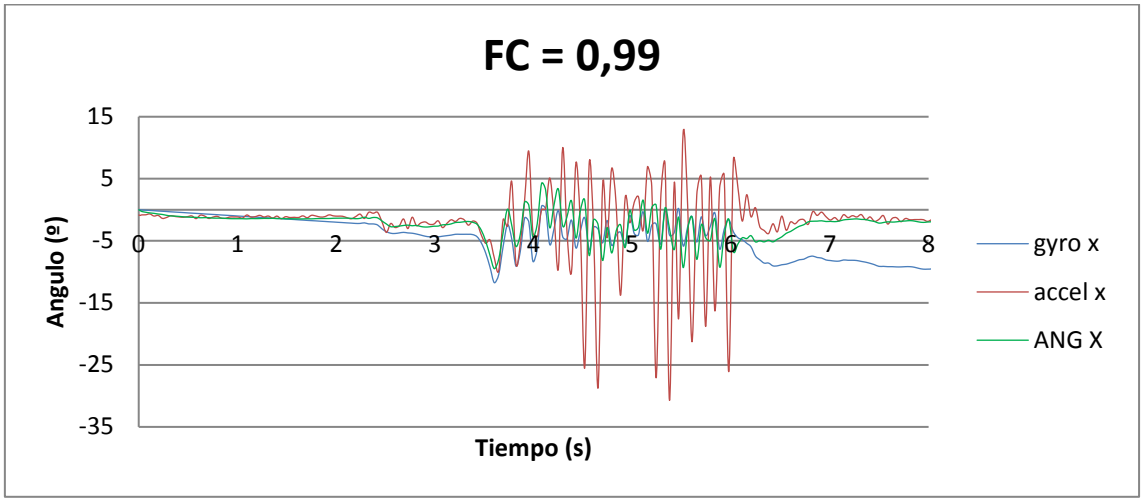
Por otro lado, al emplear el giroscopio ya se vio que este no indica ángulos, sino que mide velocidad angular. Por tanto, para obtener el ángulo será necesario realizar la integración en el tiempo para obtener de las señales medidas:

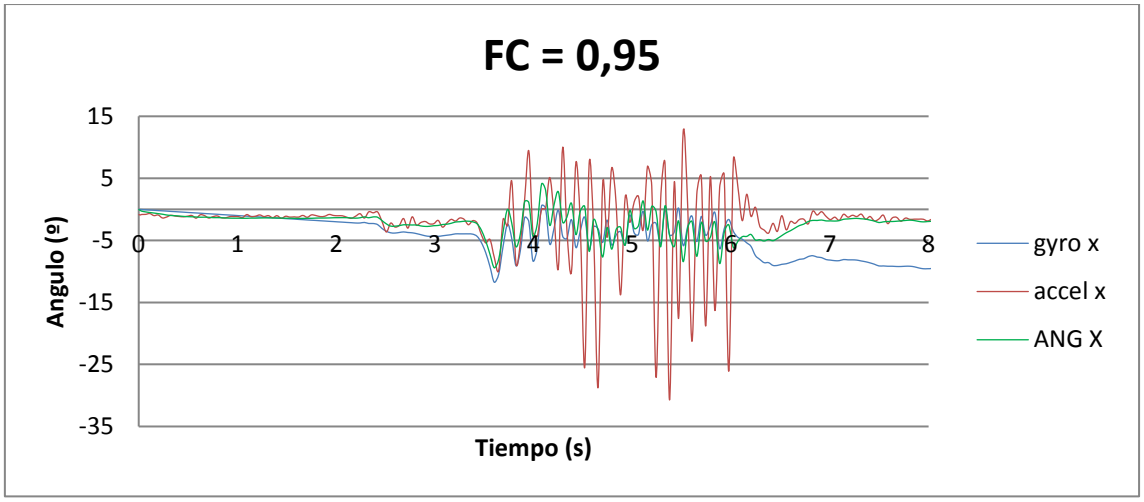
$$Angulo_{X\,giro} = Angulo_X Anterior + X\Delta t$$

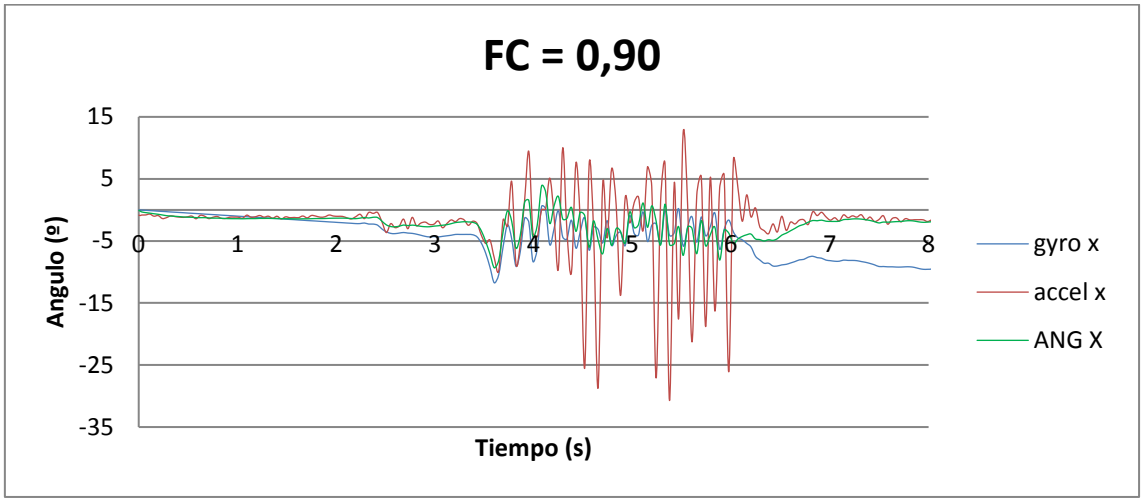
Por último es necesario filtrar la medida y combinarla empleando un filtro complementario de manera que queda:

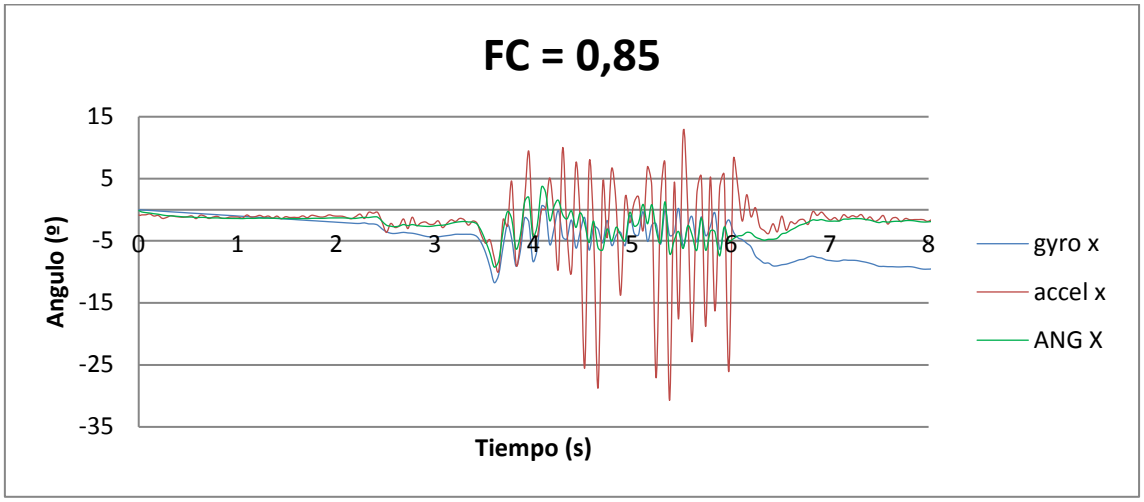
$$Angulo_X = (1 - FC) \cdot Angulo_{X\,accel} + FC \cdot Angulo_{X\,giro}$$

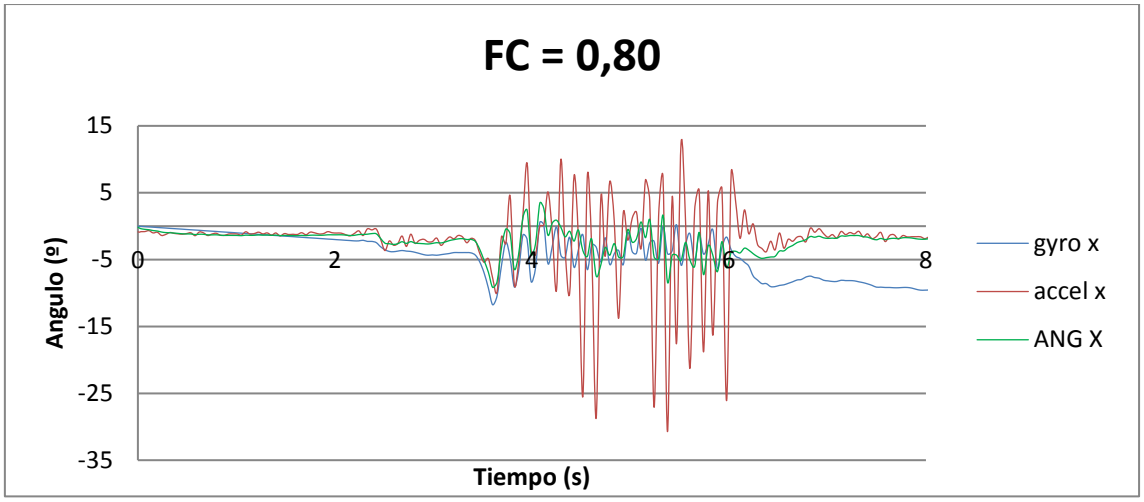
Donde FC es un factor variable entre 0 y 1 que permite obtener mejores o peores resultados en la medida y que depende del sensor empleado. Por tanto, será necesario buscar un óptimo valor para el mismo que elimine simultáneamente las componentes de alta frecuencia de la medida del acelerómetro y la deriva producida por el giroscopio. Para ello, se realizaron diferentes medidas con diferentes valores de FC en busca del valor óptimo. Los resultados pueden observarse en la siguiente figura.

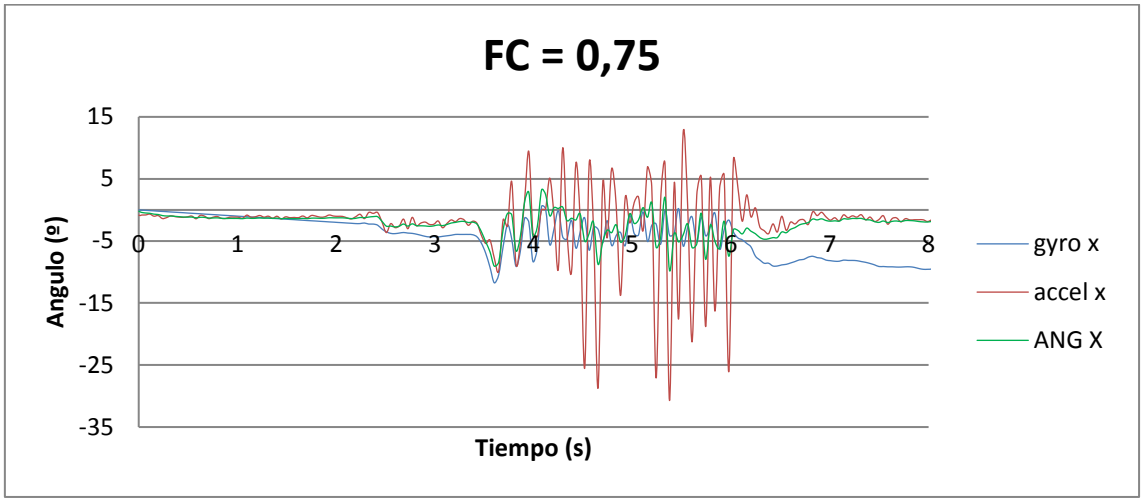

FC = 0,99
15
5
-5
-15
-25
-35
0
1
2
3
4
5
6
7
8
Angulo (º)
Tiempo (s)
gyro x
accel x
ANG X


FC = 0,95
15
5
-5
-15
-25
-35
0
1
2
3
4
5
6
7
8
Angulo (º)
Tiempo (s)
gyro x
accel x
ANG X


FC = 0,90
15
5
-5
-15
-25
-35
0
1
2
3
4
5
6
7
8
Angulo (º)
Tiempo (s)
gyro x
accel x
ANG X


FC = 0,85
15
5
-5
-15
-25
-35
0
1
2
3
4
5
6
7
8
Angulo (º)
Tiempo (s)
gyro x
accel x
ANG X


FC = 0,80
15
5
-5
-15
-25
-35
0
2
4
6
8
Angulo (º)
Tiempo (s)
gyro x
accel x
ANG X


FC = 0,75
15
5
-5
-15
-25
-35
0
1
2
3
4
5
6
7
8
Angulo (º)
Tiempo (s)
gyro x
accel x
ANG X

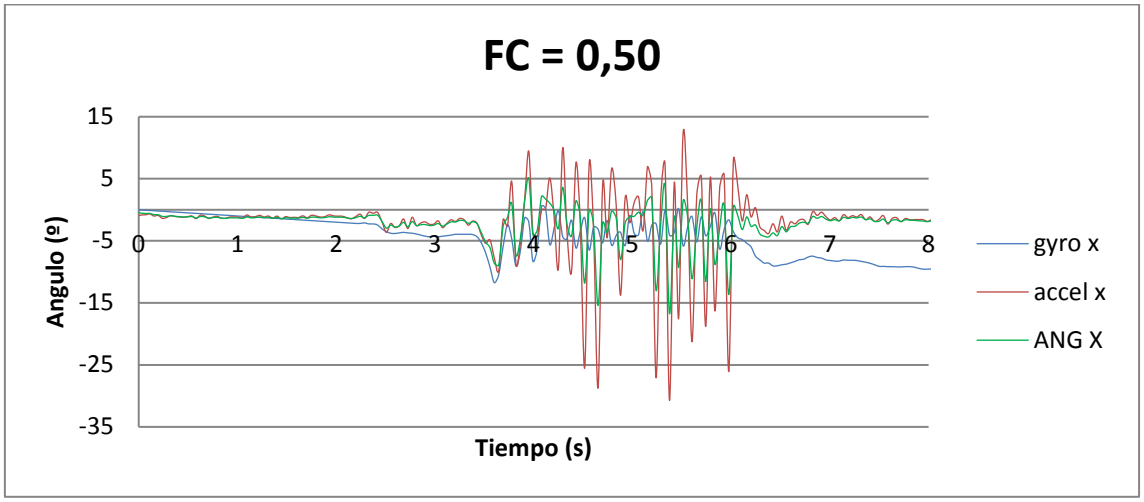


Figura 62. Colección de graficas con diferentes valores de constante del filtro.

De la colección de medidas obtenidas en el laboratorio se observó que el valor óptimo de la constante está comprendido entre 0.8 y 0.95, ya valores inferiores no terminaban de eliminar adecuadamente el ruido del acelerómetro mientras que valores superiores hacían que la medida fuera muy lenta ante variaciones bruscas. Por tanto se opta por usar una constante para el filtro complementario de valor igual a: $\boldsymbol{FC = 0.85}$, ya que con este valor se elimina la deriva producida por el giroscopio, además de eliminar el ruido medido por el acelerómetro de forma muy satisfactoria.

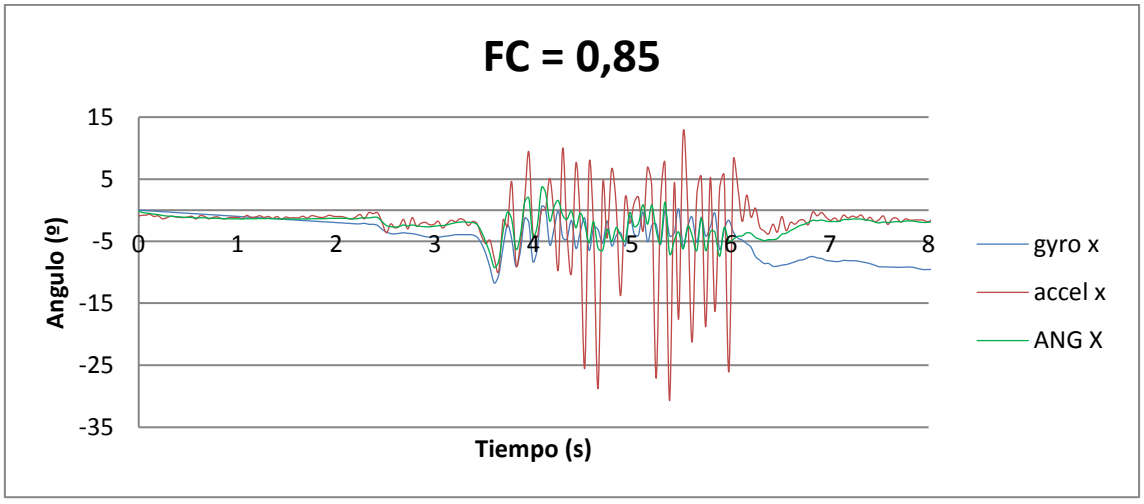


Figura 63. Obtención del ángulo con FC=0.85.

5.3.2 Encoders

Emplear encoders para la realización de este proyecto, es una opción optativa, ya que no son necesarios para mantener el equilibrio. Estos se usan principalmente para el cálculo de la velocidad del robot, de manera que permitirá realizar un doble lazo de control con el que regular la velocidad de desplazamiento del sistema. Para este proyecto, se buscaron unos motores de corriente continua que tuviesen ya unos encoders incrementales acoplados a los mismos.

Figura 64. Motor con encoder. Fuente

Empleando el encoder, se puede obtener el valor de la velocidad lineal del sistema, si se conoce el tiempo que emplea en dar una vuelta completa y las diferentes características mecánicas del motor y encoder:

Componente	Tipo	Uso
Encoder	Incremental	Sentido de giro
Reductora	1:34	Relación entre giro del eje motor, y eje de salida
PPR	10	(Pulse Per Revolution) pulsos por giro del eje motor
Rueda	62,5 cm	Radio de la rueda.

Tabla 8. Características del motor elegido. Fuente

Con esta información, la velocidad en rpm del eje del motor se podrá calcular como:

$$rpm_{eje} = \frac{1\ vuelta}{Ts_{encoder}\ ms} \cdot \frac{1000\ ms}{1\ s} \cdot \frac{60\ s}{1\ min}\ (rpm)$$

De esta manera se obtiene la velocidad en revoluciones por minuto del eje del motor, pero esto no es útil, ya que entre el eje del motor y el eje en el cual se acopla la rueda, hay instalada una reductora. Por tanto, las revoluciones por minuto de la rueda son:

$$rpm_{eje} = \frac{rpm_{eje}}{34} = \frac{1\ vuelta}{Ts_{encoder}\ ms} \cdot \frac{1000\ ms}{1\ s} \cdot \frac{60\ s}{1\ min} \cdot \frac{1}{34}\ (rpm)$$

$$\boldsymbol{rpm_{eje} = \frac{60000}{Ts \cdot 34}\ (rpm)}$$

Como el número de pulsos del encoder por cada revolución es diez, se puede contar el tiempo de dos maneras diferentes:

- Esperar a que se produzcan las diez pulsaciones y medir el tiempo en milisegundos.
- Contar solo 5 pulsaciones midiendo el tiempo en segundos, y dividir el resultado final por 2.

La fórmula resultante del cálculo de las revoluciones por minuto de la rueda quedará como:

$$rpm_{rueda} = \frac{60}{2 \cdot Ts \cdot 34}\ (rpm)$$

Finalmente, lo que se quiere obtener al emplear un segundo lazo de control para la velocidad es controlar la velocidad lineal del robot, no la velocidad angular de las ruedas. Para pasar de la velocidad angular a la velocidad lineal:

$$velocidad_{robot} = rpm_{rueda} \cdot \frac{2\pi}{60} \cdot radio_{rueda}\ (m/s)$$

$$\boldsymbol{velocidad_{robot} = 0.625 \cdot \frac{\pi}{Ts \cdot 34}\ (m/s)}$$

5.3.3 Recepción de datos.

Como se pretende contralar el robot a distancia empleando un dispositivo Smartphone, la solución más fácil que se encontró fue la utilización de un módulo de comunicaciones inalámbricas bluetooth. El dispositivo que se va a utilizar es un módulo bluetooth HC-05 [32], el cual se puede controlar mediante el uso de la comunicación vía puerto de serie.

Figura 65. Modulo Bluetooth HC-05. Fuente

En la comunicación serie [33], la información se transmite empleando dos conductores principales, aunque pueden existir otros como sincronización de reloj:

- Rx: Recepción de datos.
- Tx: Transmisión de datos.

Como un canal de comunicación es de recepción de datos (Rx), este se debe conectar con el encargado de la transmisión de datos (Tx) del otro dispositivo. De esta manera, la conexión que se realiza entre el microcontrolador y el modulo es de la siguiente forma:

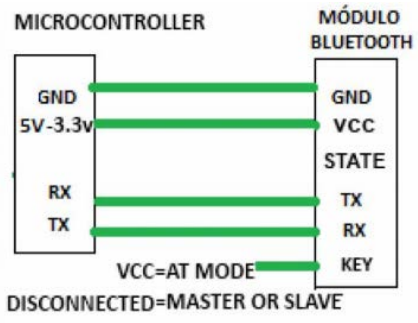


Figura 66. Comunicación serie. Fuente

En Arduino existe disponible un puerto de serie hardware asociado a los pines 0 y 1. Sin embargo, en nuestro caso se optó por no usar este hardware, ya que su uso puede dar problemas al programar el microcontrolador. Por ello se emplea la librería de Arduino

SoftwareSerial [34], la cual permite definir un elemento de comunicación serie por software en unos pines definidos por el usuario.

Tras haber realizado la conexión del módulo bluetooth, se puede realizar la comunicación de dos maneras diferentes:

- Empleando los comandos AT para realizar la configuración del dispositivo
- Desde un terminal serie, que en el caso de un dispositivo Smartphone será una aplicación de terceros descargada desde *GooglePlay* con la cual se pueden enviar y recibir datos

Dado que en nuestro caso deseamos enviar datos al dispositivo, se empleará el segundo método para realizar la comunicación con el Smartphone, no modificándose la configuración del dispositivo.

5.4 Bloque control

Este apartado se centra en el bloque de control, el cual es capaz de tratar toda la información que proviene de los diferentes sensores del robot en tiempo real, y mediante la programación, generar la acción que se debe realizar con dicha información. Este bloque está compuesto por un microcontrolador y su programación, que incorporará un sistema de control en lazo cerrado capaz de mantener la posición vertical del robot.

5.4.1 Arduino

Arduino es una plataforma de electrónica de código abierto basada en hardware y software fácil de usar, pensado para que cualquier persona que desee, pueda crear proyectos sin necesidad de la elaboración de PCBs complejas y permitiendo el prototipado rápido de la solución deseada. Esta plataforma está fundamentada en la combinación de dos soluciones diferentes. Por un lado, tenemos el hardware, el cual está basado en diferentes placas de desarrollo de muy bajo coste basadas en microcontroladores ATMEGAxxx [35]. Estos son una familia de microcontroladores RISC de 8 bits de la empresa *ATMEL* con arquitectura Harvard, siendo el principal competidor de los PIC de 8 bits. Por otro lado existe un IDE (Integrated Develpment Enviroment) gratuito que, mediante el uso de librerías de funciones muy sencillas, permite el desarrollo de software en C para estos microcontroladores. Por tanto, dada la facilidad de acceso tanto al hardware como al software necesario, Arduino se ha empleado en multitud de proyectos electrónicos tanto a nivel amateur o de estudiantes como en complejos instrumentos científicos. Su gran difusión y el hecho de ser de código abierto han resultado en una gran comunidad mundial de desarrolladores y su uso en multitud de proyectos diferentes. Por tanto es muy fácil encontrar soluciones a los posibles errores que se puedan encontrar, lo cual facilita mucho el aprendizaje tanto de electrónica como de programación.

Figura 67. Logotipo de Arduino. **Fuente**

5.4.2 Entorno de desarrollo

El entorno de desarrollo es extremadamente sencillo ya que el programa se escribe en la zona del sketch y para compilarlo y cargarlo a cualquier placa solo es necesario pulsar un botón.

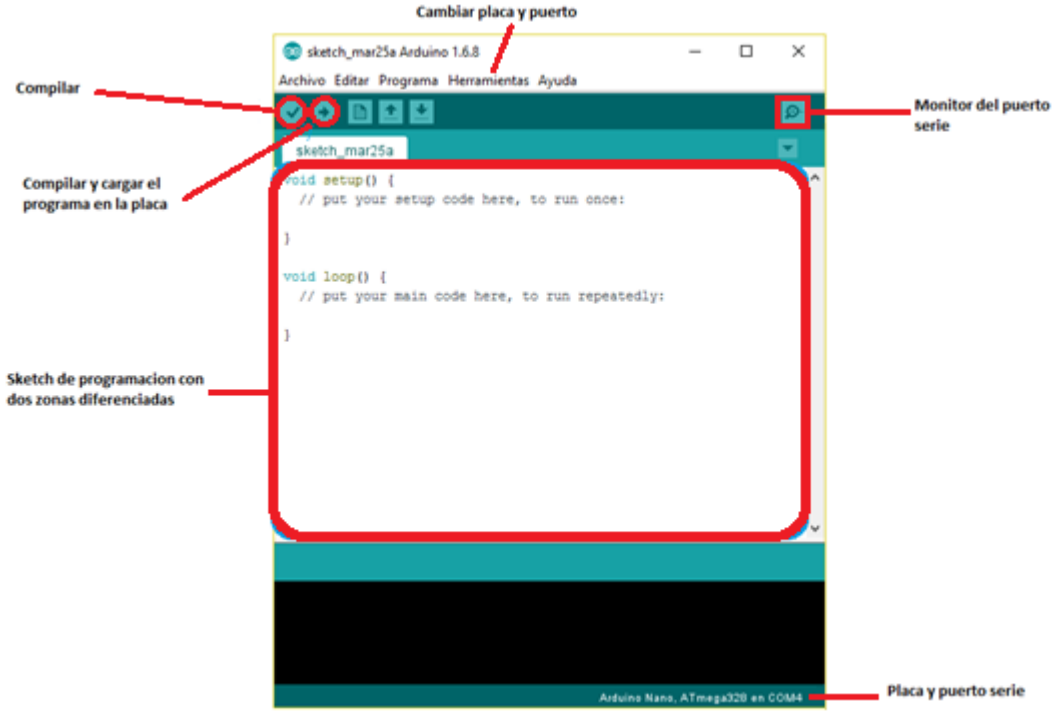


Figura 68. Entorno de desarrollo de Arduino.

El software de desarrollo de Arduino es publicado bajo una licencia libre, por lo tanto gratuita que se puede descargar desde la página oficial [36] que incluye los drivers necesarios para cualquier placa Arduino, así como las librerías necesarias para su programación. A pesar de esto, el IDE presenta una desventaja bastante importante, ya que no incorpora un debugger, complicándose la detección de errores en proyectos complejos.

5.4.3 Ventajas de Arduino

Arduino presenta varias ventajas que son decisivas a la hora de seleccionarlo para la elaboración del proyecto y no utilizar un PIC como microcontrolador:

1. Una placa de Arduino incorpora ya el microcontrolador y todos los elementos necesarios para su funcionamiento y programación. En caso de usar un PIC sería necesario diseñar la PCB necesaria y fabricarla, lo que conlleva muchas probabilidades de cometer un error. Además, debido a la naturaleza del proyecto es recomendable asegurarse de disponer de una placa cuyas soldaduras sean resistentes.

2. El software de desarrollo de Arduino está publicado bajo una licencia libre, por lo que es totalmente gratuito. Desde su página web se puede descargar tanto el programa, como los drivers y librerías que podamos necesitar, también de manera totalmente gratuita.

3. A la hora de volcar un programa en la tarjeta Arduino, este no necesita de un una tarjeta de programación como pasa con los PIC, ya que este se conecta vía serie al ordenador mediante un puerto USB.

4. Existe una gran cantidad de documentación disponible, tanto en libros como en internet, donde se puede encontrar desde diferentes ejemplos de programas hasta librerías completas para el uso de diferentes dispositivos que se le pueden conectar. Además, todas estas librerías son de código abierto.

5. Es relativamente simple poder conectarle un dispositivo como puede ser teclados, pantallas LCD, sensores y muchos más de una manera simple, y con una programación simplificada gracias a los ejemplos y librerías gratuitas disponibles.

Por estos motivos, se ha optado por emplear Arduino en lugar de otro microcontrolador. Sin embargo, aún queda un problema a resolver: ¿cuál de todas las diferentes placas de Arduino que hay en el mercado vamos a escoger? Como ya se ha dicho, existen diferentes placas de desarrollo disponibles, siendo las más importantes:

- Arduino Uno R3: basada en Atmega328P que tiene 14 entradas/salidas digitales (6 de las cuales puede ser usadas como salidas PWM), 6 entradas analógicas, con un reloj de 16MHz [37].

Figura 69. Arduino UNO. **Fuente**

- Arduino Nano: similar al Arduino Uno, pero con un empaquetado mucho más pequeño. También está basado en Atmega328P, pero este no cuenta con un regulador de tensión de 5V y la conexión con el ordenador se realiza mediante un USB del tipo Mini-B [38].

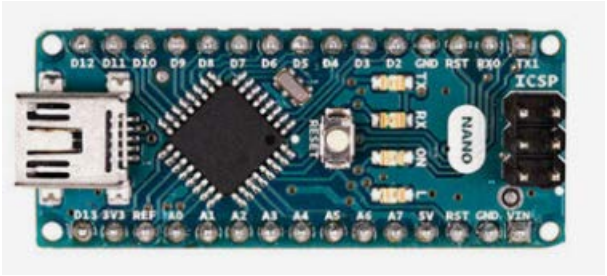


Figura 70. Placa Arduino nano. **Fuente**

- Arduino Mega: versión expandida del Arduino Uno, que tiene un procesador ATmega2560. Al es una versión expandida del Arduino Uno, dispone de 54

entradas/salidas digitales (15 como salidas PWM), 16 entradas analógicas y 4 UART (hardware serial ports) [39].

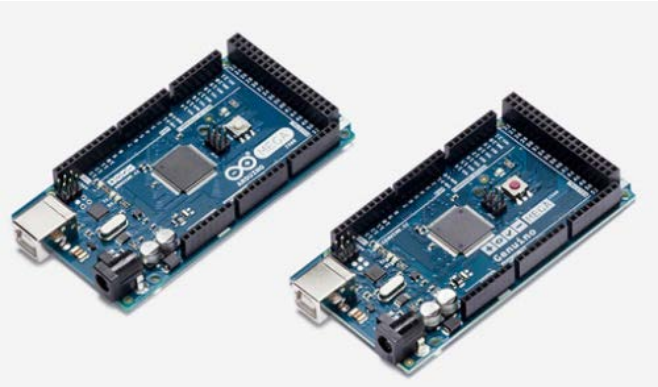


Figura 71. Placa Arduino Mega. **Fuente**

- Arduino Yún LininoOS: basado en Atmega32u4 y en Atheros AR9331. Esta placa de desarrollo soporta una distribución Linux basados en OpenWRT llamado Linino OS, ofreciendo soporte Ethernet, Wifi, USB y micro SD [40].

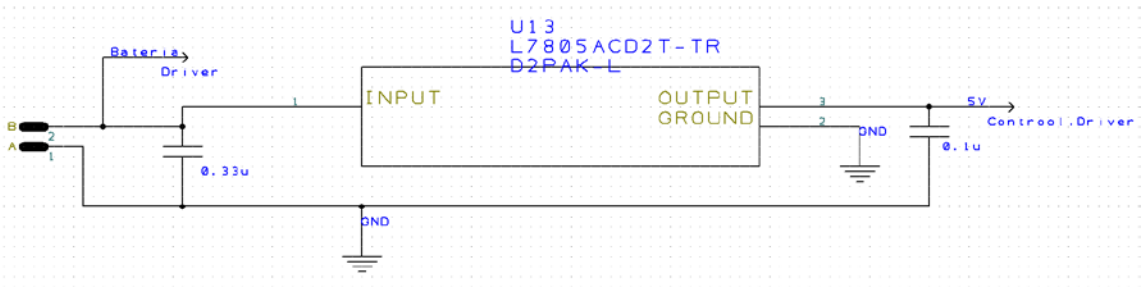
Figura 72. Placa Arduino Yún. **Fuente**

Finalmente, tras analizar las diferentes opciones que ofrece Arduino para la elaboración del proyecto, se optó por utilizar el Arduino nano. La principal ventaja de esta placa de desarrollo es que ofrece las mismas características que el Arduino Uno pero con un tamaño mucho más reducido, disponiendo sin embargo entradas y salidas digitales suficientes para poder conectar todos nuestros sensores y actuadores. Además presenta un coste muy bajo.

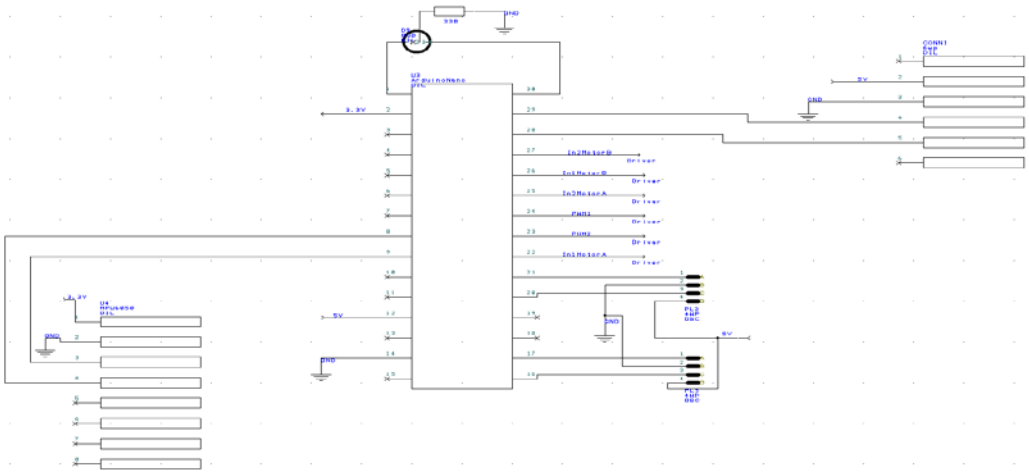
5.5 Diseño de la PCB

Una vez conocidos todos los elementos que se van a necesitar para cumplir con las condiciones de control, se pasó a diseñar una PCB en la que se recojan todos estos elementos. Para ello se ha empleado el programa de diseño DesignSpark PCB. DesignSpark PCB es un software gratuito desarrollado por RS Components y que permite realizar el diseño de tarjetas electrónicas de una forma sencilla y amigable. Antes de realizar el diseño de cada una de las partes, se verificó el correcto funcionamiento de todos los circuitos en una placa de prototipado. A continuación se presentarán los esquemáticos para los diferentes elementos del circuito.

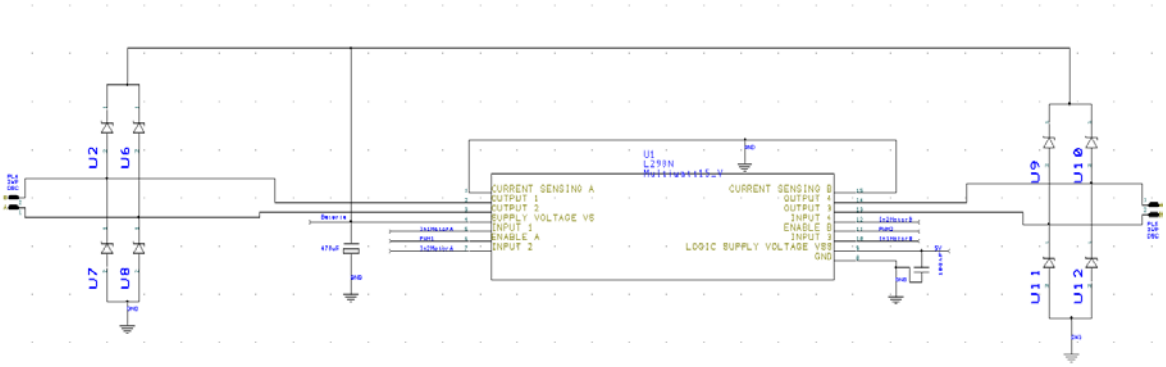
Circuito de alimentación



Circuito del driver



Circuito de control



Todos estos esquemáticos pueden observarse en más detalle en el anexo “Esquema eléctrico”. Así pues, una vez se verificó la corrección de todos los esquemáticos, se procedió al diseño de la PCB mediante la herramienta *Translate To PCB*. Los pasos que se dieron para el diseño de la placa fueron los siguientes:

1. Distribución de los elementos entre las capas de la tarjeta. En este caso se tienen dos capas, la capa superior (verde) y la inferior (azul).

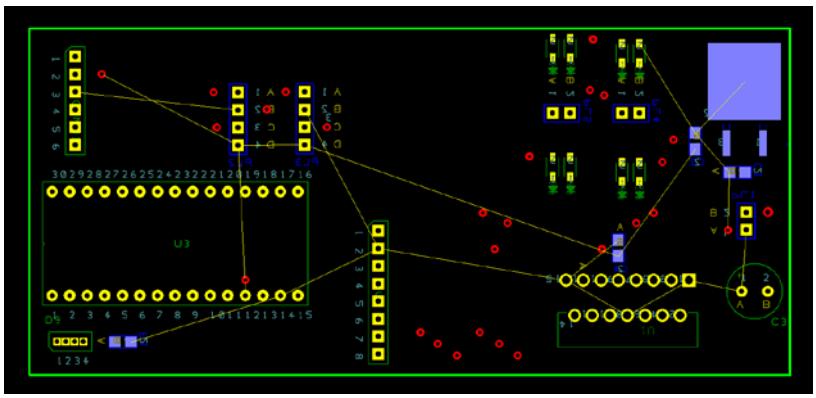
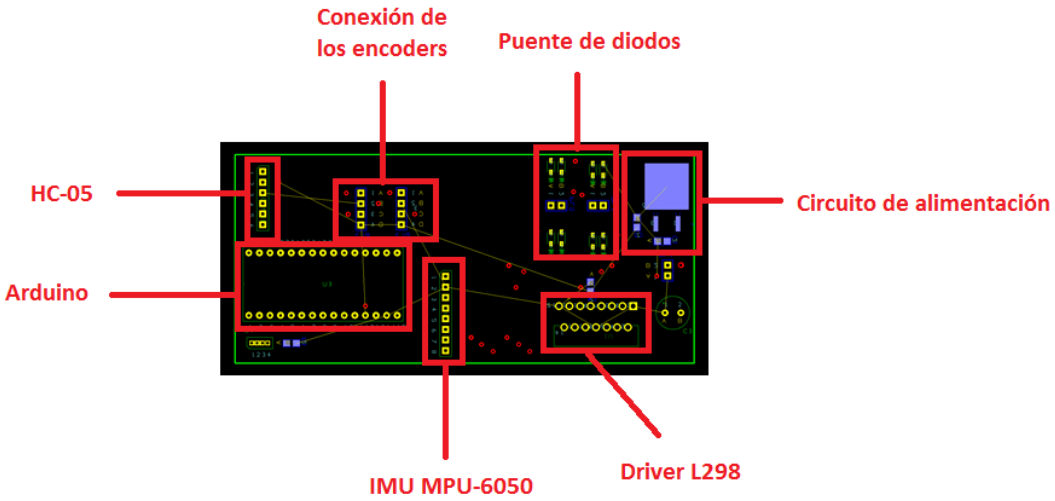


Figura 73. Distribución de los componentes.

La distribución de los elementos que se ha realizado, es la que se puede ver en la siguiente figura:



2. Una vez la distribución está hecha, se crean las diferentes conexiones entre los diferentes dispositivos. Para ello se define la tecnología de fabricación asociada a la máquina de fresado que se usará, en la cual se indica el tamaño de las pistas, tamaño de los pads…

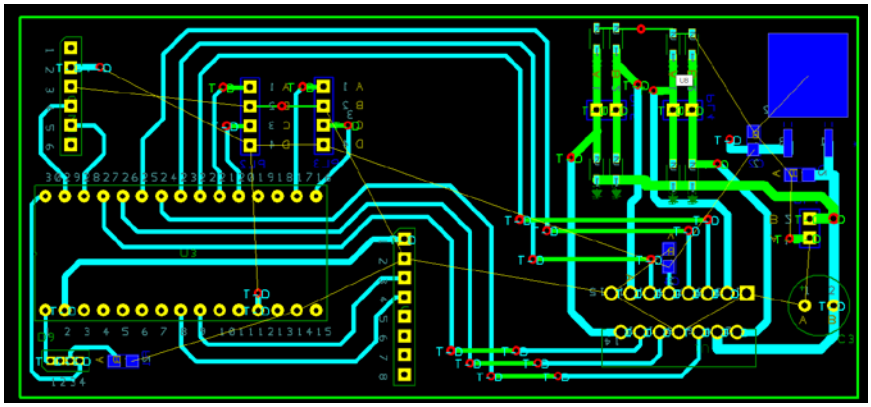


Figura 74. Distribución de las pistas por la PCB.

3. Tras haber completado todas las conexiones entre los componentes, se crearon los diferentes planos de alimentación y masa. En el plano de masa es recomendable separar la masa de la parte de potencia, de la parte de señal.

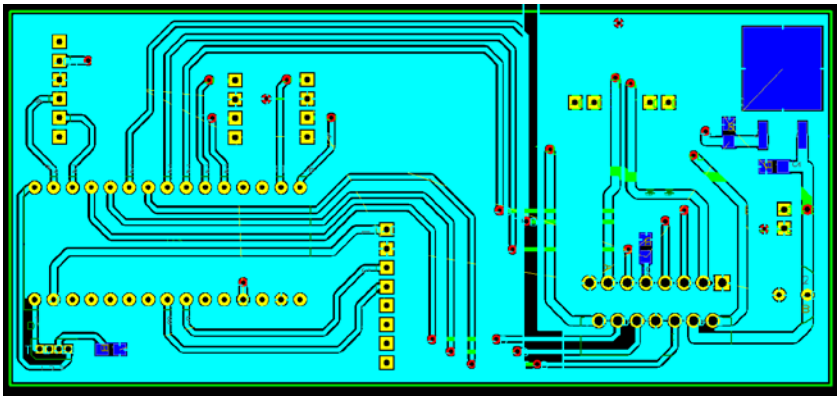


Figura 75. Plano de masa.

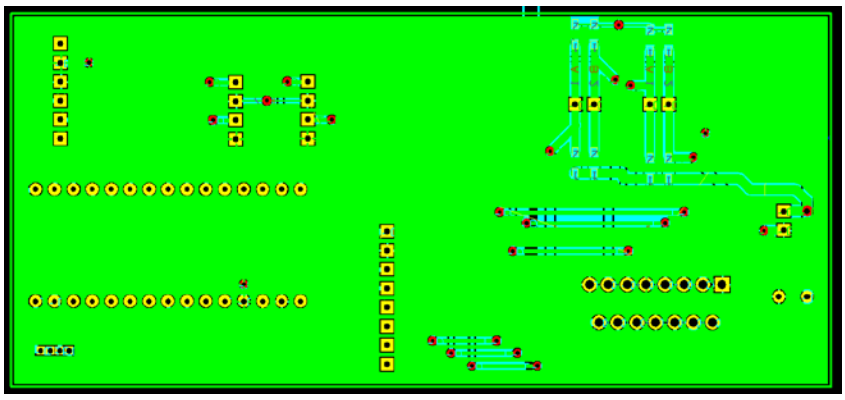


Figura 76. Plano de alimentación.

4. DesignSpark PCB tiene incorporado una herramienta, con la cual se puede crear un diseño en 3D de la PCB diseñada. De esta manera permite ver cómo será el diseño final y comprobar de forma visual que la tarjeta no presente ningún error.

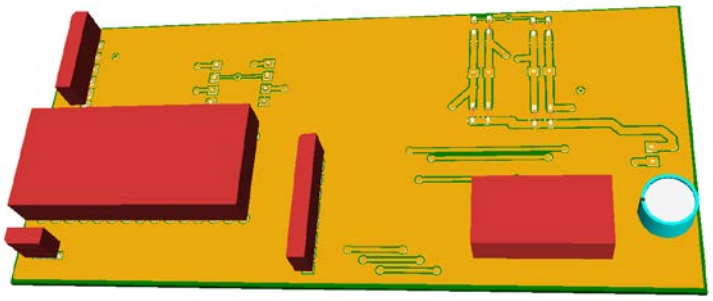


Figura 77. Capa superior de la PCB.

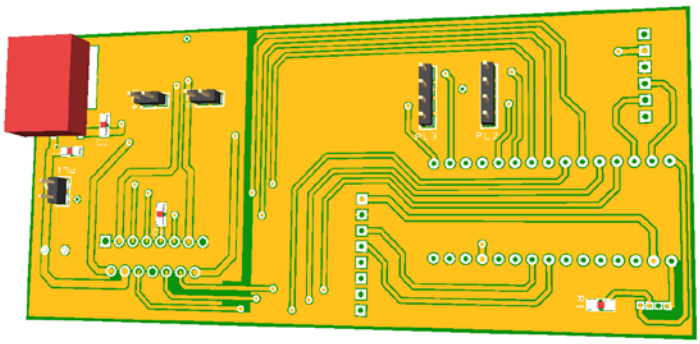


Figura 78. Capa inferior de la PCB.

Una vez la placa estaba fabricada, se procedió a soldar los diferentes elementos en ella. En primer lugar, se soldaron todas las vías, las cuales nos permiten la conexión de una de las caras de la PCB con la otra. Tras acabar de soldarlas, el siguiente paso fue soldar los componentes más pequeños de nuestra tarjeta, que eran las resistencias, condensadores y diodos de montaje superficial. Tras estos, se procedió a soldar el circuito de alimentación añadiendo el regulador, el cual también es de montaje superficial y al que se le añadió pasta térmica para mejorar le disipación de calor. Se decidió que estos elementos fuesen superficiales ya que nos garantizábamos ahorrar espacio en la placa, y tener más superficie a la hora de colocar las diferentes pistas.

Tras haber soldado los componentes más complicados, se pasó a soldar los componentes más sencillos al ser de agujero pasante, que son el driver y el condensador. Al driver se le añadió un disipador de calor, ya que se calienta bastante debido a la gran cantidad de corriente que soporta. Por último se soldaron los zócalos del Arduino, la unidad de medida inercial y el modulo bluetooth HC-05, así como los conectores de los motores y la fuente de alimentación.

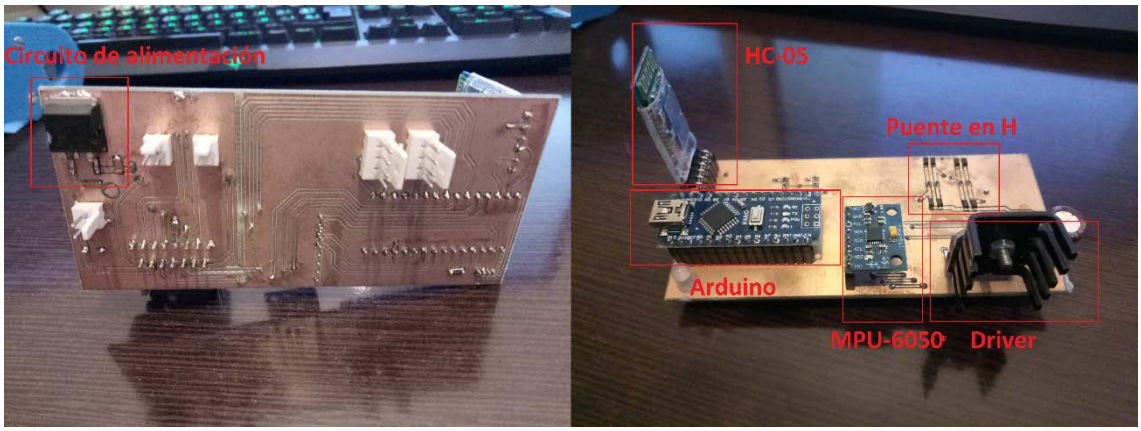


Figura 79. PCB final.

6 Software

6.1 Introducción

En este apartado se procederá a explicar la programación con la cual se ha conseguido realizar el control del robot. Para ello, el software cuenta con diversos apartados diferenciados, entre los que destacan los siguientes:

- adquisición de datos desde el exterior, que incluye el ángulo de inclinación, la velocidad del robot y los comandos enviados vía bluetooth
- implementación de un algoritmo de control con un doble lazo de control digital
- actuación sobre los motores DC para conseguir el equilibrio

Para implementar el código se utilizaran varias herramientas de programación, como pueden ser las librerías que se emplearan para las diferentes comunicaciones que hay que realizar. Otras herramientas que se van a usar son las subrutinas e interrupciones, mediante las cuales se llevaran a cabo diversas operaciones.

6.2 Flujo del programa

Para poder tener una visión global del programa, se verá en primer lugar un diagrama de flujo gracias al cual será posible seguir el orden lógico del mismo. Además, el uso de un diagrama de flujo simplificará posteriormente la programación al tener un esquema claro del mismo. El diagrama de flujo empleado a la hora de realizar este proyecto puede verse en la Figura 80. Puede observarse que, en primer lugar, será necesario inicializar los diferentes registros y ajustar los parámetros necesarios para el funcionamiento de los diferentes elementos del robot. Una vez se ha realizado esta inicialización, se pasará al bucle principal, el cual se ejecutará continuamente y funcionara de la siguiente manera:

1. Se inicia la comunicación con el sensor de medida inercial y se obtienen los datos del acelerómetro y giroscopio en los tres ejes. Tras recibir estos datos, se realizan las operaciones matemáticas necesarias para obtener el ángulo de inclinación.
2. Se comprueba la comunicación bluetooth. Si se ha recibido un dato, el cual será una carácter entre a y e, un número o un salto de línea, se llevaran a cabo dos operaciones distintas. Si el dato es un número, se cambiaran los coeficientes del controlador PID, mientras que si el dato es un carácter, se define una dirección de movimiento.
3. Conocidos los valores del PID y la dirección de movimiento, se calcula el PID de velocidad con el cual se obtiene el ángulo de referencia que se debe seguir para mantener una velocidad determinada.
4. Se calcula la acción de control mediante el PID de posición. la entrada de este control será la referencia calculada anteriormente, menos el ángulo medido por el sensor de medida inercial.
5. Conocida la acción de control, que será una PWM, esta se enviara al driver para poder controlar los motores, y en definitiva la posición del robot. Otra característica a tener en cuenta, es que si el robot se cae al suelo, aunque el PID genere una acción de control, esta no se tenga en cuenta y la velocidad del robot en este caso sea cero, así nos evitaremos que se escape corriendo cuando se caiga.

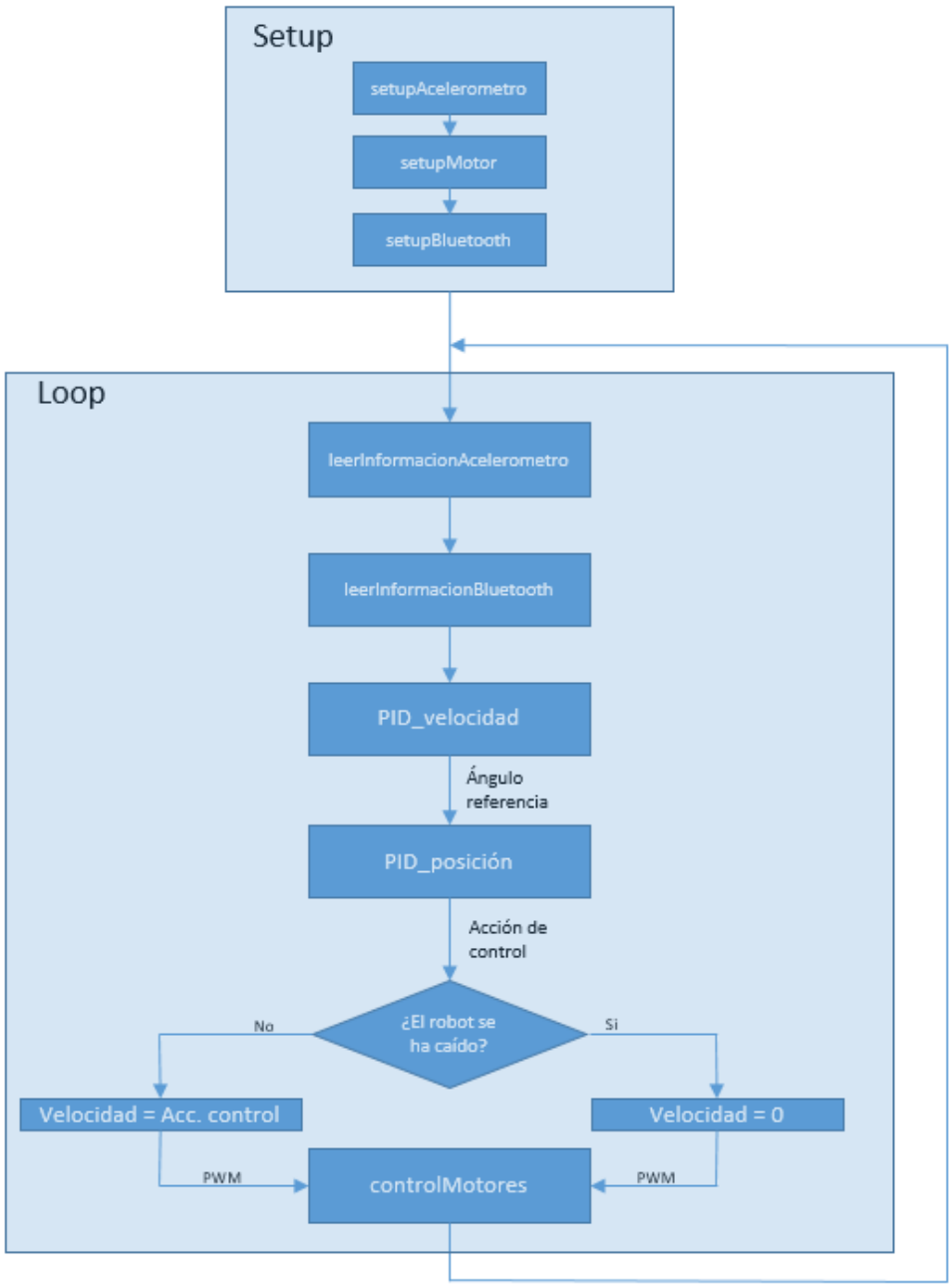


Figura 80. Diagrama de flujo del programa.

Este diagrama de flujo será el que habrá que implementar en Arduino para el control del robot. Para llevarla a cabo, se deberán conocer los diferentes apartados que hay en el sketch de programación:

- Setup(): La función setup se llama una sola vez cuando se inicia el sketch, y se emplea para la inicialización de variables, modos de funcionamiento de los pines, iniciar las librerías, etc. Solo se ejecuta una vez, después de encender o resetear la placa Arduino.
- Loop(): La función loop se trata de un bucle infinito en el cual se escribe el algoritmo de programación que se desee. Como es un bucle infinito, permite que el programa cambie y responda en función de las órdenes que reciba la tarjeta Arduino.
- Variables globales: Antes de la función setup, se pueden declarar todo tipo de variables, las cuales serán globales y se usaran en los diferentes bloques de programación.

- Subrutinas: Son funciones que resuelven una tarea específica. Son segmentos de código que se separan del bloque principal, pero pueden ser llamadas desde este bloque o desde otra subrutina.

6.3 Obtención de datos

Para la realización de este proyecto será necesario conectar el robot con diferentes dispositivos que permitan recibir la información del exterior. Más específicamente, el microcontrolador va a tener que comunicarse con el sensor IMU y con un dispositivo de control (Smartphone) a través del bluetooth. Por otro lado, será necesario medir la velocidad lineal del robot, para lo cual se empleara uno de los encoders que llevan incorporados los motores.

6.3.1 Obtención del ángulo

Como ya se ha comentado en el Apartado 5.3.1, para obtener la medida del ángulo de inclinación del robot se empleará un sensor IMU, más concretamente el MPU-6050. Empleando la información recibida a través del acelerómetro y giroscopio que incorpora será posible calcular la inclinación del robot. Para conseguir los datos deseados se empleará el bus de comunicación I2C que tiene incorporado el propio sensor, el cual se conectará con el módulo I2C de Arduino. Para la programación se empleará la librería Wire [41] que tiene las siguientes funciones implementadas:

- Wire.begin(): permite iniciar la comunicación I2C
- Wire.beginTransmission(): inicia la comunicación con el sensor, indicando también su dirección
- Wire.write(): permite al maestro (Arduino) escribir un byte en el bus de comunicación para que el esclavo pueda leerlo
- Wire.read():permite al esclavo escribir un byte en el bus de comunicación para que el maestro (Arduino) pueda leerlo
- Wire.endTransmission(): finaliza la comunicación

La comunicación I2C es una comunicación bidireccional basada en la topología de maestro-esclavo, en la que Arduino funcionara como maestro y el sensor de medida inercial hará la función de esclavo. La comunicación entre ambos será de escritura y lectura de los diferentes registros del sensor, en los cuales se guardan los datos. Dependiendo del tipo de dato que se quiera conocer, se debe indicar a que registro del sensor queremos acceder empleando las funciones anteriores. Igualmente, escribiendo la configuración adecuada en los registros del sensor es posible inicializar las características que se desean para el mismo [42].

Parámetros del sensor	
Acelerómetro	± 2g
Giroscopio	± 250º

Tabla 9. Parámetros del sensor.

Estos parámetros de definirán dentro de la función de inicialización del sensor, quedando la misma de la siguiente manera:

```
// Inicializamos la comunicación y los parámetros con el acelerómetro
void setupAcelerometro() {
  Wire.begin();
  Wire.beginTransmission(MPU_addr); //Iniciamos la comunicación con el
sensor (dirección 0x6B)
  Wire.write(0x6B);  //En el registro 0x6B (PWR_MGMT_1) => Modo de
energía, reloj interno y sensor de temperatura
  Wire.write(0);  //En el registro PWR_MGMT_1 se escribe 000000
  Wire.endTransmission(true); //Dejamos de escribir

  Wire.beginTransmission(MPU_addr); //Iniciamos la comunicación con el
sensor (dirección 0x6B)
  Wire.write(0x1B); //En el registro 0x1B (GYRO_CONFIG) ± 250 °/s
  Wire.write(0);  //En el registro 0x1B se escribe 000000
  Wire.endTransmission(true); //Dejamos de escribir

  Wire.beginTransmission(MPU_addr); //Iniciamos la comunicación con el
sensor (dirección 0x6B)
  Wire.write(0x1C); //En el registro 0x1C (ACCEL_CONFIG) ± 2g
  Wire.write(0);  //En el registro 0x1C se escribe 000000
  Wire.endTransmission(true); //Dejamos de escribir
}
```

Una vez que se ha definido el funcionamiento del sensor, se pasara a realizar la programación necesaria para calcular el ángulo de inclinación con las ecuaciones que ya se vieron anteriormente. Para ello se definirá una función *leerInformacionsAcelerometro* encargada de la lectura y procesado de los datos. El diagrama de flujo de esta función puede verse en la Figura 81.

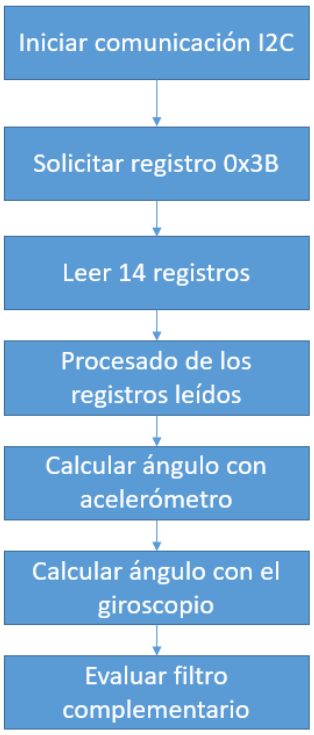


Figura 81. Diagrama de bloques de la subrutina leerInformacionAcelerometro().

Como puede observarse, el primer paso será iniciar la comunicación y pedirle al sensor que nos proporcione los datos del acelerómetro y giroscopio. Estos datos están en 14 registros diferentes, empezando por el registro 0x3B, y se corresponden a los valores de la información del acelerómetro y giroscopio en los 3 ejes así como de temperatura. La información de cada parámetro se almacena en dos registros de 8 bits ya que los valores que nos da el sensor para cada parámetro son de 16 bits. Por tanto será necesario aplicar una operación de desplazamiento de 8 bits a la izquierda al registro de los bits más significativos y combinar la información de los dos registros mediante la función OR. Así pues, el código para implementar esto queda de la siguiente manera:

```
//Nos conectamos al Sensor por comunicación Wire
Wire.beginTransmission(MPU_addr);
Wire.write(0x3B); //Empezamos con el registro 0x3B => ACCEL_XOUT_H
Wire.endTransmission(false);
Wire.requestFrom(MPU_addr, 14, true); //Le pedimos al sensor conectado
a la dirección 68 que nos de los 14 registros desde el indicado antes

//Cada registro nos da 8 bits pero la aceleración ocupa 16 hay que hacer
una operación de desplazamiento y suma

AcX = Wire.read() << 8 | Wire.read(); // 0x3B (ACCEL_XOUT_H) & 0x3C
(ACCEL_XOUT_L)
AcY = Wire.read() << 8 | Wire.read(); // 0x3D (ACCEL_YOUT_H) & 0x3E
(ACCEL_YOUT_L)
AcZ = Wire.read() << 8 | Wire.read(); // 0x3F (ACCEL_ZOUT_H) & 0x40
(ACCEL_ZOUT_L)
Tmp = Wire.read() << 8 | Wire.read(); // 0x41 (TEMP_OUT_H) & 0x42
(TEMP_OUT_L)
GyX = Wire.read() << 8 | Wire.read(); // 0x43 (GYRO_XOUT_H) & 0x44
(GYRO_XOUT_L)
GyY = Wire.read() << 8 | Wire.read(); // 0x45 (GYRO_YOUT_H) & 0x46
(GYRO_YOUT_L)
GyZ = Wire.read() << 8 | Wire.read(); // 0x47 (GYRO_ZOUT_H) & 0x48
GYRO_ZOUT_L)
```

Finalmente, ya se dispone la información en cada eje, por lo que se puede utilizar las ecuaciones para el cálculo del ángulo de inclinación. Con la información del acelerómetro se obtiene la inclinación en el eje X empleando la fórmula ya vista:

```
//Calcular ángulos con acelerómetro
float accel_ang_x = atan(AcY / sqrt(pow(AcX, 2) + pow(AcZ, 2))) *
(180.0/3.14);
```

Para el cálculo del ángulo con el giroscopio, será necesario medir el tiempo que pasa entre cada muestra. Además se puede implementar el filtro complementario a la vez que se calcula el ángulo con el giroscopio, por lo que el resultado final para el cálculo del ángulo de inclinación es:

```
dt = (millis() - tiempo_prev) / 1000;
tiempo_prev = millis();

//Calcular ángulos de rotación con giroscopio y filtro complementario
ang_x = FC * (ang_x_prev + (GyX / 131) * dt) + (1 - FC) * accel_ang_x;
ang_x_prev = ang_x;
```

6.3.2 Obtención de la velocidad

Para realizar la medida de la velocidad se emplean las ecuaciones que se vieron en el Apartado 5.3.2, siendo necesario calcular primero las revoluciones por minuto y luego pasar a velocidad lineal. Para ello se emplearán las interrupciones de Arduino, de forma que el cálculo de la velocidad se realizará cada vez que se detecte un flanco ascendente en una de las fases del encoder.

Dado que se van a emplear las interrupciones de Arduino el primer paso será inicializar las interrupciones, siendo necesario declarar los pines de la fase y la interrupción, además del tipo de interrupción, la cual será del tipo *flanco de subida*. Para ello, en la subrutina de inicialización de los encoders *setupMotor* se emplea la instrucción *attachInterrupt*, con la cual se define el pin de interrupción y su tipo. De esta forma, la función queda de la siguiente forma:

```
// Inicializarmos el control de los motores
void  setupMotor() {
  // Declaramos los pines de control PWM de los motores
  pinMode(Motor1, OUTPUT);
  pinMode(Motor2, OUTPUT);
  // Declaramos los pines para el control de giro de cada motor
  pinMode(M1_InA, OUTPUT);
  pinMode(M1_InB, OUTPUT);
  pinMode(M2_InA, OUTPUT);
  pinMode(M2_InB, OUTPUT);

  //  // Declaramos los pines del encoder y su interrupción
  pinMode(M1_PhaseA, INPUT);
  attachInterrupt(digitalPinToInterrupt(M1_PhaseA),
leerInformacionEncoderM1, RISING); //interrupcion (timer 0)patilla 2
flanco de subida
   }
```

Una vez inicializada la interrupción será necesario definir la función de *callback* de la misma, llamada *leerInformacionEncoderM1*. Esta función será llamada cada vez que se produzca una interrupción y su diagrama de flujo se puede ver en la Figura 82.

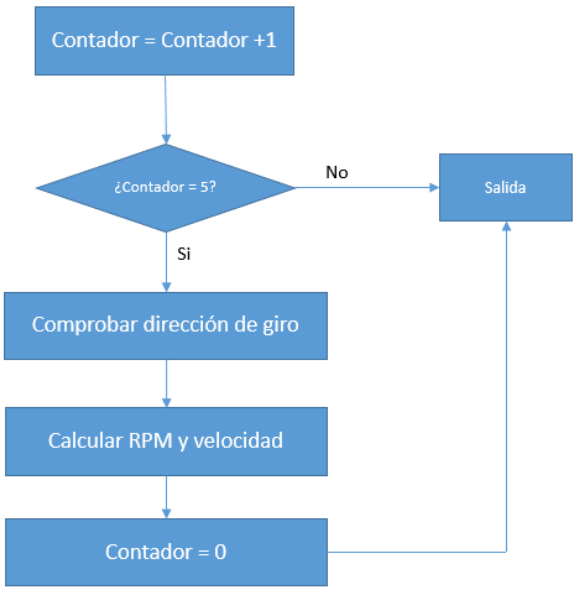


Figura 82. Diagrama de la interrupción LeerInformacionEncoderM1()

Como se puede ver, cada vez que salta la interrupción, se aumenta un contador y se mide el tiempo que ha transcurrido hasta que el contador llega a su valor límite. Conocido el tiempo se puede calcular las revoluciones por minuto a las que giran las ruedas. Además para conocer el sentido de giro se emplea la otra fase del encoder, de manera que si se comprueba el estado de ese pin, ya sea alto o bajo, se determina el sentido de giro. Por último se procederá a transformar el valor de revoluciones por minuto a una velocidad lineal. De esta forma, la programación final de esta función queda de la siguiente manera:

```
// Actualiza la velocidad del motor 1
void leerInformacionEncoderM1()
{
  float rpm = 0;
  float Ts_encoder = 0;
  int direc = 0;
  c = c + 1;
  // 10 o 5*2: 10 pulsos se corresponden a una vuelta del eje del
motor
  if (c >= 5) // se puede poner 10, o contar 5 y dividir entre dos
  {
    direc = digitalRead(M1_PhaseB);
    Ts_encoder = (millis() - tiempo_prev_encoder) / 1000;
    tiempo_prev_encoder = millis();
    // 34: relacion de la reductora (34 vueltas eje motor = 1 vuelta
eje salida)
    rpm = 60 / (2 * 34 * Ts_encoder);
    // Pasar a velocidad lineal (2*pi/60)*0.625 (radio de la rueda)
    velocidad = rpm * 0.06545;
    // Se comprueba el sentido de giro
    if (direc == 0)
    {
      velocidad = -velocidad;
    }
    c = 0;
  }
}
```

6.3.3 Comunicación bluetooth

Como ya se explicó, la comunicación bluetooth con el módulo HC-05 se realiza mediante un bus serie de dos hilos, uno de transmisión de datos, y otro de recepción de datos. La plataforma Arduino dispone de hardware dedicado en su microcontrolador asociado a los pines 0 y 1 para realizar esta comunicación. Sin embargo, el uso de estos pines puede dar lugar a problemas en la programación, por lo que en nuestro caso se optó por emplear un puerto de serie emulado por software asociado a unos pines diferentes a estos. Para ello se usará la librería *SoftwareSerial* [43]. Por otro lado, para la transmisión de datos desde un Smartphone será necesario disponer de un software en el mismo que permita la comunicación. Sin embargo, como ya se ha visto anteriormente, se desea emplear la comunicación bluetooth para dos funciones diferentes. Por un lado, se desea poder sintonizar el PID, ya que se podrá cambiar las constantes del control sin necesidad de cargar el sketch de programación desde la plataforma Arduino. Y por otro lado, se desea emplear la conexión para el control remoto del robot. En nuestro caso se optó por emplear dos aplicaciones diferenciadas de terceros, las cuales son:

- Bluetooth Terminal HC-05 [44]: aplicación que emula un terminal en el puerto de serie, permitiendo mandar cualquier tipo de dato, y recibirlo por pantalla. Se empleará para la sintonización del PID.

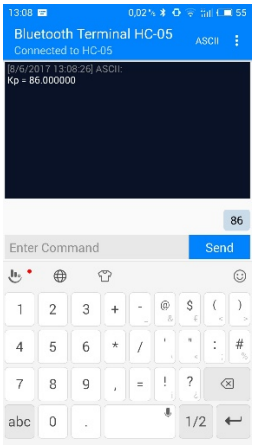


Figura 83. Bluetooth Terminal HC-05.

- Bluetooth Serial Controller [45]: esta aplicación permite crear una cruceta con la cual controlar el movimiento del robot. Se basa en la asignación de un carácter ASCII a cada uno de los botones que será transmitido al pulsar cada uno de los mismos.

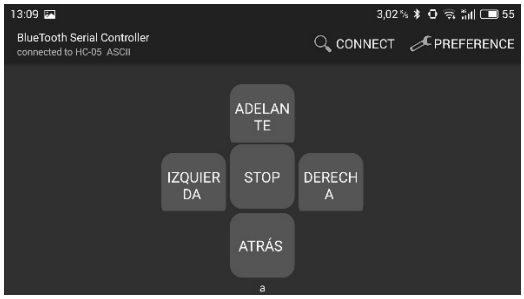


Figura 84. Bluetooth Serial Controller.

Como ya se ha dicho, para la comunicación serie se emplea una librería específica de este tipo de comunicaciones, la librería *SoftwareSerial,* en la cual se encuentra una función que

nos permite inicializar el puerto en los pines Rx y Tx deseados. Esta inicialización tendrá lugar en la función *setupBluetooth*, en la cual se crea el nuevo puerto de comunicación:

```
// Inicializarnos la comunicación bluetooth
void  setupBluetooth() {

  //Declaración pines HC-05
  pinMode(RXpin, INPUT);
  pinMode(TXpin, OUTPUT);

  // Inicio comunicación bluetooth
  bluetoothSerial = SoftwareSerial(RXpin, TXpin);
  bluetoothSerial.begin(9600);
}
```

Una vez que se tiene el nuevo puerto de comunicación serie creado y la comunicación abierta, se pasará definir el diagrama de flujo mediante el cual se puedan realizar dos acciones distintas, en una misma función. Las dos acciones que se deben llevar a cabo, son la de sintonización del PID, y la de controlar el robot de forma remota.

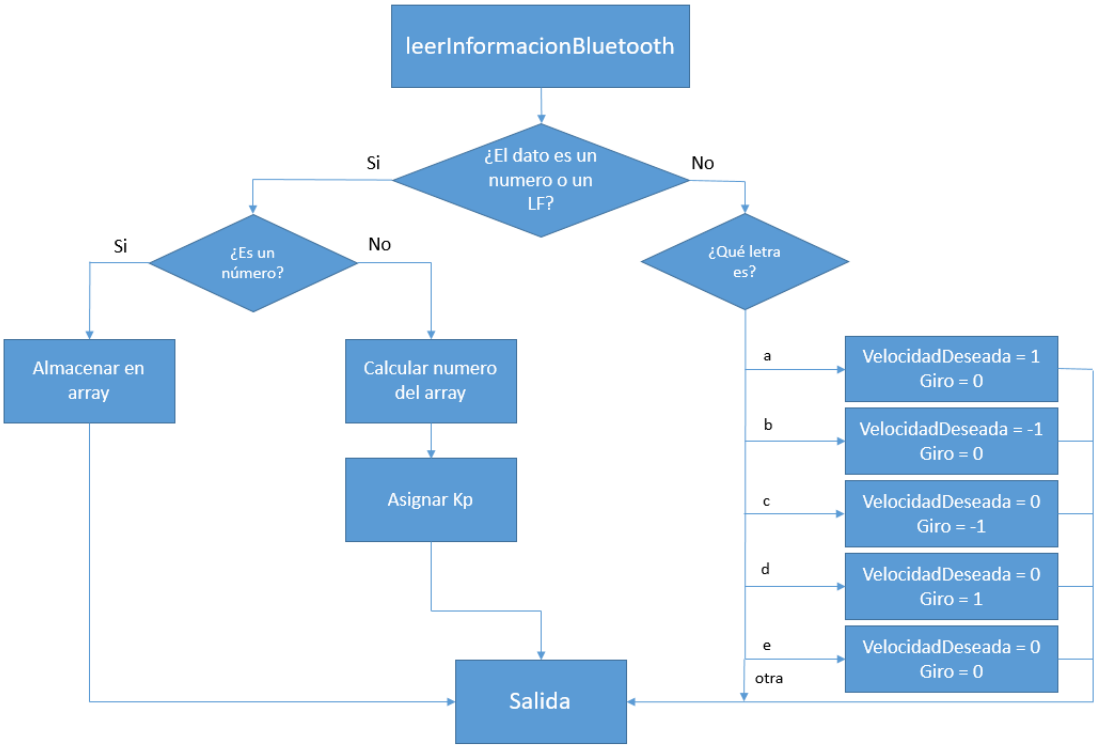


Figura 85.Diagrama de la sintonización del PID y el control remoto.

Cuando se llama a la subrutina, se comprueba si hay algún dato esperando para ser leído y, si es así, se almacena en una variable temporal. En el caso de que el dato leído corresponda al valor ASCII de un número, se entiende que se desea modificar algún valor del control PID. Por tanto, es necesario ir almacenando los diferentes caracteres recibidos en una variable array *auxiliar* hasta que el valor recibido se corresponda a un *final de línea*, ya que implica que el número se ha acabado. En caso de no recibirse un *LF,* se aumenta una variable contador y, si contador pasa de un cierto valor, se mandara un mensaje de error en el que se indique que el número es excesivamente largo. Por otro lado, al llegar un final de línea se considera que ya se

ha enviado el número completo, por lo que se convierte el array *auxiliar* en el número.Por otra parte, en el caso de que el dato leído sea una de las letras asociadas a una acción del robot, se modificarán de forma acorde las variables que determinan el sentido y giro del robot. En el caso en que el carácter recibido no sea ninguna de estas letras, no se realiza ningún cambio y la subrutina espera a recibir un nuevo dato.

Para implementar estas dos funciones se empleará la misma subrutina, cuya implementación queda de la siguiente manera:

```
// Lectura de la información recibida por bluetooth
void   leerInformacionBluetooth()
{
  byte state = 0; // Estado de la comunicación
  if (bluetoothSerial.available())
  {
    state = bluetoothSerial.read();
    //Serial.println(state, HEX);
    if (((((state >= 0x30) && (state <= 0x39)) || (state == 0x2E))) ||
        ((contador_auxiliar == 0) && (state == 0x2D)))
    {
      auxiliar[contador_auxiliar] = state;
      contador_auxiliar++;

      // Número demasiado largo
      if (contador_auxiliar > SIZE_ARRAY)
      {
        bluetoothSerial.println("Numero enviado demasiado largo");
        contador_auxiliar = 0;
        memset (auxiliar, 0x00, sizeof (auxiliar));
      }
    }
    // Al recibir un LF (suponemos comunicación con LF únicamente como
final de línea)
    else if (state == 0x0D)
    {
      Ki_posicion = atof(auxiliar);
      bluetoothSerial.print("Ki_posicion = ");
      bluetoothSerial.println(Ki_posicion, 6);

      contador_auxiliar = 0;
      // Poner todos los elementos del array a cero
      memset (auxiliar, 0x00, sizeof (auxiliar));
    }

    //Al recibir una "a"(0x61) significa que el robot se debe mover
hacia adelante
    else if (state == 0x61)
    {
      velocidadDeseada = 2;
      giro = 0;
    }
    //Al recibir una "b"(0x62) significa que el robot se debe mover
hacia atras
    else if (state == 0x62)
    {
      velocidadDeseada = -3;
      giro = 0;
    }
```

```
    //Al recibir una "c"(0x63) significa que el robot debe girar a la
izquierda
    else if (state == 0x63)
    {
      velocidadDeseada = 1;
      giro = -1;
    }
    //Al recibir una "d"(0x64) significa que el robot debe girar a la
derecha
    else if (state == 0x64)
    {
      velocidadDeseada = 1;
      giro = 1;
    }
    //Al recibir una "e"(0x65) significa que el robot se debe parar en
el sitio
    else if (state == 0x65)
    {
      velocidadDeseada = 0;
      giro = 0;
    }

    // Condición de error. Olvidar el valor y reiniciar. Esto no es
muy correcto, pero peor es nada
    else
    {
      bluetoothSerial.println("Error en la recepción");
      contador_auxiliar = 0;
      memset (auxiliar, 0x00, sizeof (auxiliar));
    }
  }
}
```

6.4 Control PID

Para conseguir que el robot se mantenga estable será necesario convertir la información del ángulo de inclinación proporcionada por el sensor MPU-6050 en una velocidad de giro de las ruedas tal, que corrija dicha inclinación. El sistema gracias al cual se consigue la relación entre una entrada, y una salida deseada, es un control basado en la realimentación de la salida del sistema. En nuestro caso, el esquema del sistema realimentado será el siguiente:

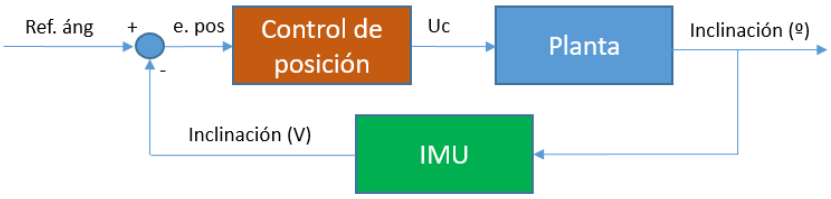


Figura 86. Sistema de control.

Aunque existen una gran variedad de controles diferentes, en nuestro caso se ha optado por diseñar un algoritmo de control PID digital e implementarlo en Arduino. Este cálculo se realiza dentro de la función *PID_posicion* y se encarga de combinar las tres partes del control (proporcional, integral y derivativa) en una acción de control. Además, dado que es necesario controlar el tiempo que pasa entre una iteración y la siguiente así como el error cometido en la iteración anterior, será necesario guardar estos valores en variables globales. De esta forma, la programación de la función para el PID de ángulo queda de la siguiente manera:

```
//Función de cálculo de la acción del PID
void PID_posicion(float error)
{
  t_posicion=millis();
  double tchange_posicion=t_posicion-lastTime_posicion;

  //Calculo de la parte proporcional
  float PID_prop = Kp*error;

  //Cálculo de la parte integral
  double
errorSum_posicion=errorSum_posicion+error_posicion*tchange_posicion/10
00;
  float PID_int_posicion = Ki_posicion*errorSum_posicion;

  //Calculo de la parte derivativa
  double errord_posicion=(error_posicion-
erroranterior_posicion)/(tchange_posicion/1000);
  float PID_der_posicion = Kd_posicion*errord_posicion;

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

  erroranterior=error;
  lastTime_posicion=t_posicion;
}
```

6.5 Doble lazo de control

Además de controlar que el robot se mantenga en equilibrio, en nuestro caso se desea poder controlar también la velocidad de movimiento del robot. Para ello será necesario incorporar un segundo lazo de cuya idea básica es la de proporcionar un ángulo de inclinación de referencia al control del equilibrio, de manera que el robot se incline para uno u otro lado y se regule de esta forma su velocidad. Así, si el ángulo de referencia que se busca está inclinado, el robot se moverá en un sentido para evitar que se caiga. Por tanto, la nueva estructura del lazo de control, es el siguiente:

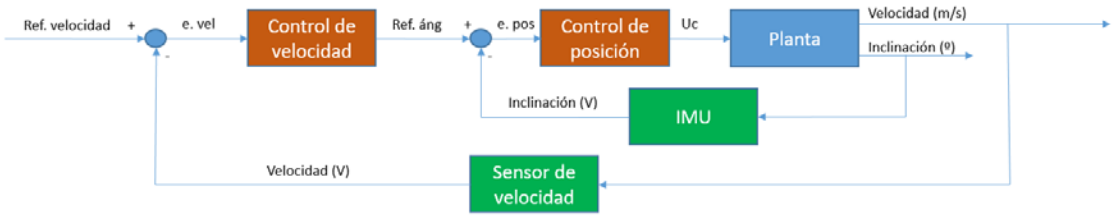


Figura 87. Doble lazo de control.

Para conseguir estabilidad, el lazo interno de control, que corresponde al control de la inclinación, debe ser más rápido que el lazo externo, que es el lazo de control de velocidad. Por tanto, se establecerá que el periodo de muestreo del lazo de control interno sea mucho menor que el externo, de esta manera será mucho más rápido.

Por otra parte, aunque inicialmente se planteó hacer un control PID completo también para la velocidad, al final se optó por emplear un control PI para el control de la velocidad. Esto se debe a que realmente no era necesario disponer de un control preciso de la velocidad al desearse controlar únicamente el sentido de movimiento. Con esto se consigue un controlador

más simple y más fácil de sintonizar. La función encargada, por tanto de este segundo bucle de control queda de la siguiente manera:

```
void PID_velocidad(float error_velocidad)
{
  float t_velocidad = millis();
  float tchange_velocidad = t_velocidad - lastTime_velocidad;
  if (tchange_velocidad >= Tmuestreo_velocidad)
  {
    //Calculo de la parte proporcional
    float PID_prop_velocidad = Kp_velocidad * error_velocidad;

    //Caluculo de la parte integral
    float errorSum_velocidad = errorSum_velocidad + error_velocidad *
Tmuestreo_velocidad / 1000; //Calculo del error integral
    float PID_int_velocidad = Ki_velocidad * errorSum_velocidad;
//Calculo del termino integral

    U_control_velocidad = PID_prop_velocidad + PID_int_velocidad;

    lastTime_velocidad = t_velocidad;
  }
}
```

6.6 Correcciones del algoritmo PID

La programación del PID presentada anteriormente funciona correctamente, pero tiene ciertas limitaciones cuando se implementa en una aplicación de un sistema real. Para que se comporte como un control PID de nivel industrial fue necesario realizar unas pequeñas modificaciones en la programación.

6.6.1 Periodo de muestreo

Un controlador PID tiene mejor comportamiento cuando se ejecuta a intervalos regulares, pero con la función anterior no se controla el tiempo entre iteraciones. Por tanto, el algoritmo se estaba ejecutando en periodos irregulares, lo que implica:

- un comportamiento inconsistente del PID debido a que se ejecuta regularmente y a veces no.
- la realización operaciones matemáticas extras para el cálculo de las partes derivada e integral, que son las componentes dependientes del tiempo.

La solución de este problema es simple, y basta con asegurarse de que la función que ejecuta el PID lo haga regularmente, incorporando el concepto de tiempo de muestreo al algoritmo. Para ello simplemente será necesario añadir una línea de código que compara el tiempo que transcurre cada vez que se llama a la función del PID y un tiempo de muestreo que nosotros seleccionamos. De esta forma, en la función PID_posicion hay que modificar:

```
//Función de cálculo de la acción del PID
void PID_posicion(float error)
{
  t_posicion=millis();
  double tchange_posicion=t_posicion-lastTime_posicion;
  …
}
```

por:

```
void PID_posicion(float error_posicion)
{
  float t_posicion = millis();
  float tchange_posicion = t_posicion - lastTime_posicion; //Calculo
del tiempo del PID

  //Condicion para conseguir que el PID se ejecute en periodos de
tiempo constantes
  if (tchange_posicion >= Tmuestreo_posicion)
  {

      …

  }
}
```

6.6.2 Derivtive Kick

Una vez que se empezaron a realizar pruebas con el control PID, se encontró que aparecía un error en el apartado derivativo del PID. El objetivo de este punto es eliminar ese fenómeno, conocido como "derivative kick".

Figura 88. Fenómeno del Derivative Kick. **Fuente**

Como vemos en la imagen anterior, el problema reside en el momento que aparece una variación brusca del error en la medida, entre la entrada y la referencia a seguir. Como se trata de una derivada, una variación grande del error en un periodo de tiempo pequeño provoca una acción de control infinita:

$$\frac{dError}{dt} = \frac{dSetPoint - DInput}{dt} = \infty;\ \ para\ dt\ pequeño\ y\ dError\ grande.$$

Este valor se introduce en la ecuación del PID, por lo que da como resultado un pico indeseable en la salida. Por suerte hay una forma bastante simple de corregir este problema:

$$\frac{dError}{dt} = \frac{dSetPoint}{dt} - \frac{dInput}{dt}$$

Cuando la referencia es constante y no varía a lo largo del tiempo, la derivada de esta (*SetPoint*) es cero, de manera que desaparece este término de la ecuación, que es el causante de generar una acción muy grande al inicio del PID:

$$\frac{dError}{dt} = -\frac{dInput}{dt}$$

Figura 89. Solución del Derivative Kick. Fuente

El resultado es que la derivada del Error es igual a la derivada negativa del Input siempre que el SetPoint se mantenga con un valor constante. Esto resulta ser una solución perfecta que se implementa de manera sencilla: A la hora de calcular la acción de control, en lugar de sumar la acción derivativa "Kd*dError", se le resta "Kd*dInput". Para implementar esta solución se realizaron los siguientes cambios en el código del programa. Se cambia:

```
//Cálculo de la parte derivativa
double errord_posicion=(error_posicion-
erroranterior_posicion)/(tchange_posicion/1000);
float PID_der_posicion = Kd_posicion*errord_posicion;
```

por:

```
//Código con la solución del Derivative Kick implementado.
//Calculo de la parte derivativa
float inputd_posicion = (ang_x - angulo_x_anterior) * 1000 /
(Tmuestreo_posicion); //Calculo de la derivada de la entrada (input)
float PID_der_posicion = - Kd_posicion * inputd_posicion; //Calculo de
la parte derivativa de un PID
    angulo_x_anterior = ang_x;
```

6.6.3 WindUp

Este problema aparecer cuando en un sistema aparece un error grande durante un tiempo prolongado, lo que provoca que el apartado integral del PID aumente para eliminar el error. Pero si el actuador es limitado, la acción de control no será la misma que la que el PID cree estar enviando.

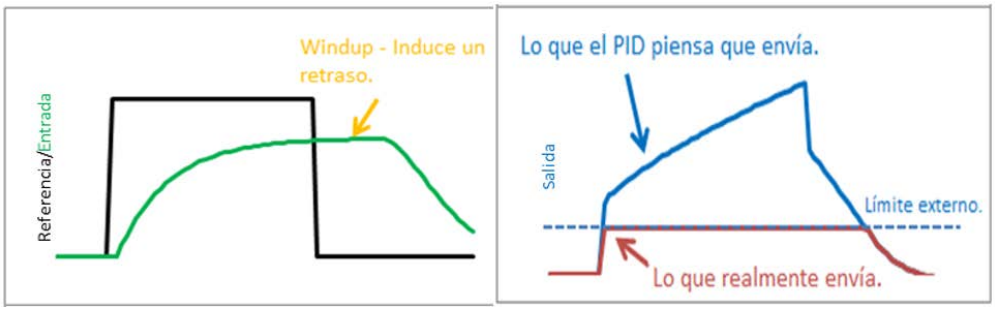


Figura 90. Error producido por el WindUp. Fuente

Nuestro controlador es un Arduino nano que tiene una salida PWM de 8 bits, por lo que la acción de control estará contenida entre 0 y 255. Si la acción de control se satura, esto nos provocará un retraso. La forma de corregir este problema es aplicar al PID los límites necesarios para evitar la saturación y el retraso que conlleva:

- Acotar la acción de control.
- Acotar la acción integral.

Para ello, en la acción de control se cambia:

```
//Generación de la acción de control
U_control_posicion=PID_prop_posicion+PID_int_posicion+PID_der_posicio;
```

por:

```
//Generación de la acción de control
U_control_posicion = PID_prop_posicion + PID_int_posicion +
PID_der_posicion;

//Condicion Anti WindUp en la acción de control
if (U_control_posicion >= 255.0)
  {
    U_control_posicion = 255.0;
    }
    else if (U_control_posicion <= -255.0)
    {
      U_control_posicion = -255.0;
    }
```

Y en el cálculo de la acción integral se sustituye:

```
//Cálculo de la parte integral
double
errorSum_posicion=errorSum_posicion+error_posicion*tchange_posicion/10
00;
float PID_int_posicion = Ki_posicion*errorSum_posicion;
```

por:

```
//cálculo de la parte integral
float errorSum_posicion = errorSum_posicion + error_posicion *
Tmuestreo_posicion / 1000; //Calculo del error integral
float PID_int_posicion = Ki_posicion * errorSum_posicion; //Calculo
del termino integral
//Condición Anti WindUp para el término integral
if (PID_int_posicion > 255)
    {
      PID_int_posicion = 255;
    }
    else if (PID_int_posicion < -255)
    {
      PID_int_posicion = -255;
    }
```

De esta manera se consigue eliminar el fenómeno del WindUp además de conseguir que la salida permanezca dentro del rango que deseamos.

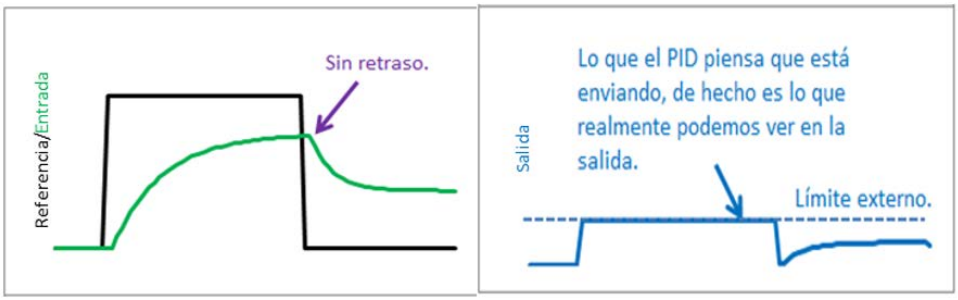


Figura 91. Corrección del WindUp. Fuente

Gracias a estas modificaciones que se han realizado al algoritmo de control, se consigue crear un control PID mucho más robusto que el primero, capaz de responder muy bien ante todos los problemas que pueda presentar nuestro sistema.

6.7 Actuación sobre los motores

Gracias al doble lazo de control visto anteriormente, a la salida se generará una acción de control PWM que controlará la velocidad de giro del motor. Esta señal se enviará al driver y su ciclo de trabajo tendrá un valor comprendido entre 0 y 255, ya que las salidas PWM que implementa Arduino son de 8bits.

Como se ha visto en el diagrama de bloques la función responsable del control de los motores será *controlMotores* y tomará como parámetros de entrada el valor de la señal PWM y, en caso de giro, el sentido de giro deseado. El diagrama de flujo de esta función puede verse en la Figura 92.

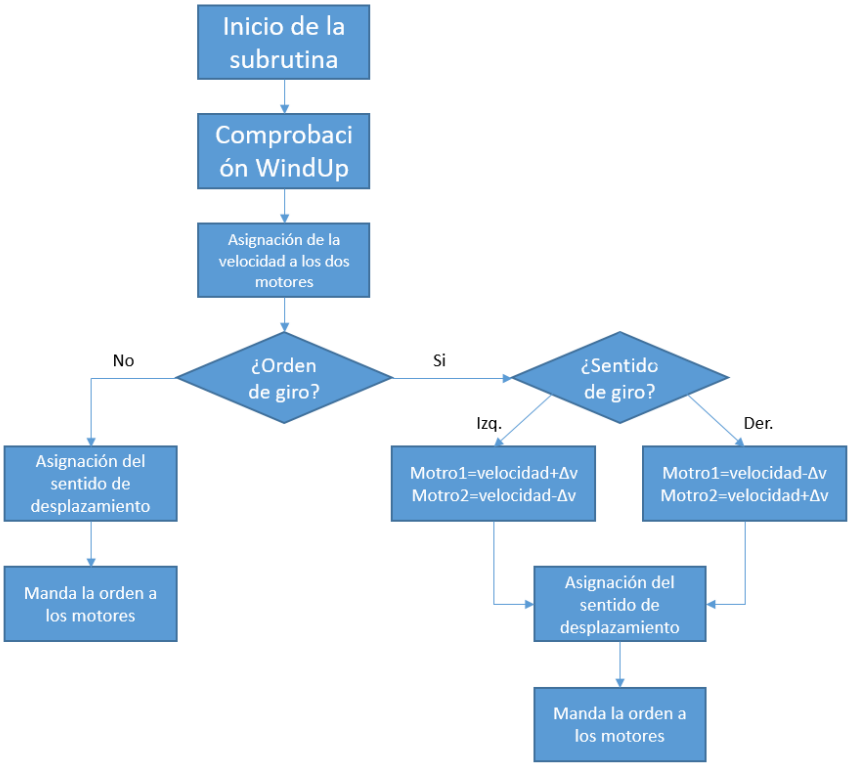


Figura 92. Diagrama de bloques del contorlMotores.

Como ya se ha dicho, el valor de la señal PWM tendrá como valor límite 255, por lo que habrá que garantizar que la acción nunca sobrepase este umbral. Lo mismo ocurre si la acción de control es negativa, que indicaría que los motores deban girar en el sentido contrario, no pudiendo permitirse que se sobrepase el valor de -255. Por tanto, y aunque en principio no sería necesario si se ha implementado el control WindUp en el PID de control, en primer lugar se realizará un chequeo de la velocidad deseada para verificar que se encuentra entre estos valores. Una vez se ha verificado que el valor de actuación es correcto, hay que tener en cuenta que el signo de la acción de control nos está determinado el sentido de giro. Para ello, cada motor cuenta con dos pines adicionales, gracias a los cuales se puede definir el sentido de giro, estableciéndose la velocidad de giro con el valor absoluto de la acción de control. Esto se recoge en la siguiente tabla:

	pin digital	estado	velocidad
Motor 1	4	0	positiva
	7	1	
	4	1	negativa
	7	0	

	pin digital	estado	velocidad
Motor 2	8	1	positiva
	9	0	
	8	0	negativa
	9	1	

Tabla 10. Sentido de giro de los motores.

Como se desea que el robot además de mantener el equilibrio, se pueda mover adelante, atrás y que gire sobre su eje, se deberá añadir al control de movimiento un algoritmo que nos permita girar. Sin embargo, el control del movimiento hacia adelante y hacia atrás no requiere de ninguna modificación especial ya que se hará simplemente cambiando la velocidad objetivo del doble lazo de control.

Para conseguir el giro, lo que se hace es que, en función del sentido de giro, a un motor se le aplica una señal de control mayor que la calculada por el PID, y al otro motor se le resta esa misma cantidad, de manera que se consiga girar sobre su propio eje. Por supuesto, habrá que tener en cuenta los límites de la señal PWM, no sobrepasando nunca los valores 255 y -255.

Por último, una vez se han establecido el sentido de giro de los dos motores y su velocidad, se procederá a poner estos valores en los pines de control del motor usando las funciones digitalWrite y analogWrite. De esta forma, el código que se emplea para actuar sobre los motores es:

```
// Controla los motores en función de U_control_posicion
void controlMotores(int velocidadMotor,int giro)
{
  //Protección que evita pasar los limites
  if (velocidadMotor > 255)
  {
    velocidadMotor = 255;
  }
  else if (velocidadMotor < -255)
  {
    velocidadMotor = -255;
  }

  int velocidadMotor1=velocidadMotor;
  int velocidadMotor2=velocidadMotor;
  //giro = 1 indica el giro a la izquierda
  if(giro==1)
  {
    velocidadMotor1-=15;
    velocidadMotor2+=15;

    if(velocidadMotor1>255)
    {
      velocidadMotor1=255;
    }
    else if(velocidadMotor1<-255)
    {
      velocidadMotor1=-255;
    }
    else if(velocidadMotor2>255)
    {
```

```
      velocidadMotor2=255;
    }
    else if(velocidadMotor2<-255)
    {
      velocidadMotor2=-255;
    }
  }

  //giro = -1 indica el giro a la derecha
  else if (giro==-1)
  {
    velocidadMotor1+=15;
    velocidadMotor2-=15;

    if(velocidadMotor1>255)
    {
      velocidadMotor1=255;
    }
    else if(velocidadMotor1<-255)
    {
      velocidadMotor1=-255;
    }
    else if(velocidadMotor2>255)
    {
      velocidadMotor2=255;
    }
    else if(velocidadMotor2<-255)
    {
      velocidadMotor2=-255;
    }
  }

  //se colocan los motores para que giren en un sentido o en otro,
además de indicar la velocidad.
  if (velocidadMotor > 0)
  {
    analogWrite(Motor1, velocidadMotor1);
    digitalWrite(M1_InA, HIGH);
    digitalWrite(M1_InB, LOW);
    analogWrite(Motor2, velocidadMotor2);
    digitalWrite(M2_InA, HIGH);
    digitalWrite(M2_InB, LOW);
  }
  else if (velocidadMotor < 0)
  {
    analogWrite(Motor1, -velocidadMotor1);
    digitalWrite(M1_InA, LOW);
    digitalWrite(M1_InB, HIGH);
    analogWrite(Motor2, -velocidadMotor2);
    digitalWrite(M2_InA, LOW);
    digitalWrite(M2_InB, HIGH);
  }
  else if (velocidadMotor == 0)
  {
    analogWrite(Motor1, 0);
    analogWrite(Motor2, 0);
  }
}
```

7 Conclusiones y líneas futuras

7.1 Conclusiones

El reto con el que se empezó a realizar este trabajo fue el de emplear y demostrar los conocimientos adquiridos a lo largo del Grado y aplicarlos a un caso práctico. El interés personal que tengo en el área de la tecnología, más concretamente en la robótica, electrónica y el control, me llevaron a realizar este proyecto. De esta manera, se decidió profundizar más en estas materias realizando un TFG basado en combinar multitud de estas disciplinas, entre las que se incluyen:

- Diseño de tarjetas electrónicas
- Microprocesadores
- Electrónica
- Control Automático/Digital
- Laboratorio de proyectos de control
- Comunicaciones industriales

Empleando todo lo estudiado durante el grado asociado a estas disciplinas se ha conseguido realizar un TFG con las siguientes características:

- Arduino como base principal de la programación
- Control mediante un doble lazo que incluye dos controladores (PI y PID) digitales
- Diseño mecánico empleando un software de diseño 3D, SolidWorks, y montaje de la estructura diseñada
- Diseño eléctrico empleando un software de diseño de PCB, DesignSpark, y montaje y soldadura de diferentes componentes en la placa diseñada
- Comunicación y manejo vía bluetooth desde un Smartphone

Para la elaboración de este proyecto se ha tenido que repasar la teoría vista en varias asignaturas de la carrera como son las asignaturas de control Analógico/Digital al diseño de tarjetas electrónicas y microprocesadores, además de comunicaciones industriales, electrónica y electrónica de potencia.

7.2 Líneas futuras

A pesar de la amplia bibliografía existente sobre robots autobalanceados, durante la elaboración del proyecto han surgido diversas complicaciones que, como ya se han visto, dieron lugar a la necesidad de rediseñar la PCB. Sin embargo, y con la experiencia adquirida durante la elaboración del proyecto, se ha detectado que en el diseño final existen diversas posibilidades de mejora que habría que tener en cuenta a la hora de diseñar una nueva versión del robot. Las principales dificultades encontradas han sido:

1. Batería LiPo: Estas son unas baterías de polímero de iones de litio, que a pesar del peligro que tienen (pueden arder si no se tratan de manera adecuada), son indicadas para este tipo de aplicaciones. Con esta batería se busca sustituir las dos pilas, ya que es capaz de suministrar la energía necesaria para la demanda de consumo de los motores, que es muy elevada. Este cambio viene justificado ya que las pilas en muchas ocasiones no tenían potencia suficiente para mover los motores, lo que resultaba en

bajadas de tensión que provocaban reinicios en el microcontrolador. Se consiguió una de estas baterías en el último momento, y el problema al que nos enfrentamos fue el poco tiempo del que se disponía y la falta de espacio para colocarla. Se colocó en una primera versión en la base superior que tiene el robot, por lo que se complica la sintonización del control PID, ya que es un sistema mucho más inestable.

Figura 93. Batería LiPo empleada.

2. Motores DC: Desde un principio se buscó unos motores DC que tuviesen integrados encoders para la medición de la velocidad. Aunque los motores empleados ofrecen un par bastante elevado, la velocidad que pueden alcanzar es muy baja y presentan una zona muerta ante valores PWM bajos muy grande. Por tanto, sería necesario reemplazarlos por unos motores cuya velocidad sea mayor, con lo cual, el margen de control del que disponemos fuese mayor al actual y que funcionarán también con ciclos de trabajo bajos. Para esto sería necesario emplear la batería LiPo, ya que sin ella solo se disponían de unos pocos grados de inclinación antes de que la acción de control fuese máxima (PWM = 255).

3. Dimensiones de las diferentes piezas que componen el robot: A pesar de que las piezas diseñadas, las cuales se pueden ver en el *Anexo Planos*, proporcionan al robot una rigidez y dureza frente a golpes muy alta, lo ideal sería rediseñar algunas piezas, concretamente los laterales y la base superior, a un grosor reducido. La idea detrás de realizar estos cambios sería reducir el peso y bajar el centro de gravedad del robot, ya que en estos momentos está muy por encima del eje de rotación de los motores, lo que hace que la inestabilidad sea mayor.

4. Diseño de la PCB: Se cometieron algunos errores antes de comenzar con el diseño de la PCB, como fue no comprobar el consumo de los motores, lo que nos obligó a tener que diseñar dos PCB completamente diferentes. Ahora que se tienen más conocimientos, se podría rediseñar la PCB con diferentes componentes, como podría ser un driver más eficiente que el que se emplea actualmente, u otra distribución de I/O digitales del Arduino.

Teniendo en cuenta las mejoras propuestas anteriormente, se puede evolucionar el robot de varias maneras. Algunas de estas posibles mejoras podrían ser:

1. Reemplazar el control PID por alguna técnica más moderna de control que permita mejorar la estabilidad, como son las lógicas difusas, implementando un controlador *fuzzy* o borrosos.

2. Reemplazar los motores empleados por otro tipo de motores (brushless, stepper) que no tengan las limitaciones observadas, permitiendo un control mucho más preciso de la velocidad de los mismos.

3. Realizar un modelado preciso del sistema que permita realizar una simulación del mismo que se asemeje lo máximo posible con la realidad. Con esto se podría ajustar de manera más eficiente cualquier tipo de control. El problema de esta modelización es que es muy compleja, y sería un Trabajo Fin de Grado completo solo realizar la modelización y simulación del sistema, por eso no se realizó en este proyecto.

4. Realizar un circuito de control de la batería LiPo ya que es un elemento muy delicado y peligroso. También se puede diseñar un sistema de recarga de la batería, de manera que no fuese necesario quitarla del robot cada vez que se tiene que recargar.

5. Incorporar la opción de ver datos de telemetría tales como carga de la batería, regulación PWM, corriente consumida… bien sea a través de una pantalla LCD o mediante una aplicación en un Smartphone.

6. Comunicar el robot con diferentes dispositivos y ampliar la posibilidad de comunicación añadiendo un módulo Wifi, además del módulo Bluetooth que ya tiene incorporado. De esta manera se podría conectar un mando de Xbox en el ordenador y controlar el movimiento del robot con él.

7. Rediseñar el robot completamente y buscar convertirlo en un robot del estilo BB-8 de la película *Star Wars.* Para ello se deberían emplear 3 motores con ruedas omnidireccionales, este tipo de rueda permite el desplazamiento en cualquier dirección, las cuales estarían sobre una esfera, de manera que se buscaría mantener el equilibrio sobre esta.

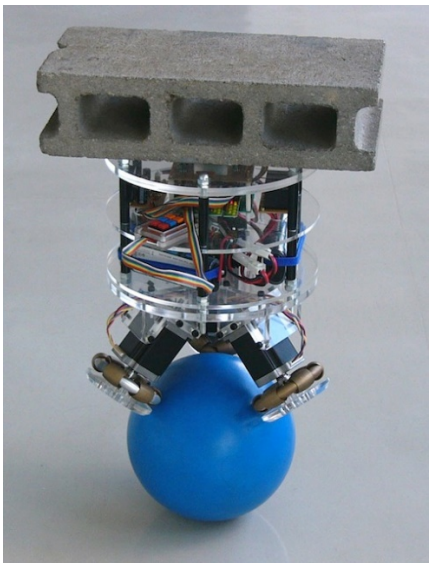


Figura 94. Robot auto-balanceado sobre una bola. Fuente

8 Referencias

[1] Yamafuji, K., Miyakawa, Y., Kawamura, T., "Synchronous Steering Control of a Parallel Bicycle", Transactions of the Japan Society of Mechanical Engineers Series C, 55 (513), pp. 1229-1234 (1989).

[2] Feng, Q., Yamafuji, K., "Design and simulation of control systems of an inverted pendulum", Robotica, 6 (3), pp. 235-241 (1988)

[3] Grasser, F., D'Arrigo, A., Colombi, S., Rufer, A.C., "JOE: A mobile, inverted pendulum", IEEE Transactions on Industrial Electronics, 49 (1), pp. 107-114 (2002)

[4] http://www.segway.es/es/

[5] Valera, A., Vallés, M., Cardo, M., "Desarrollo y control de un péndulo de Furuta", https://www.researchgate.net/publication/228855086_Desarrollo_y_control_de_un_pendulo_de_Furutahttps://www.researchgate.net/publication/228855086_Desarrollo_y_control_de_un_pendulo_de_Furuta

[6] van der Blonk, K., "Modeling and Control of a Ball-Balancing Robot", Master Thesis, Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS), University of Twente (2014).

[7] http://www.spacex.com/falcon9

[8] http://asimo.honda.com/

[9] https://upcommons.upc.edu/handle/2099.1/26340?locale-attribute=es

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

[11] https://en.wikipedia.org/wiki/Inverted_pendulum

[12] Bennett, S, "Nicolas Minorsky and the Automatic Steering of Ships", IEEE Control Systems Magazine, 4 (4), pp. 10-15 (1984).

[13] Gaydou, D., Redolfi, J., Henze, A., "Filtro complementario para estimación de actitud aplicado al controlador embebido de un cuatrirrotor", Simposio Argention de Sistemas Embebidos (SASE 2011) (2011).

[14] https://es.wikipedia.org/wiki/Filtro_de_Kalman

[15] http://dep.fie.umich.mx/~camarena/FiltroKalman.pdf

[16] https://es.wikipedia.org/wiki/%C3%81ngulos_de_navegaci%C3%B3n

[17] https://es.wikipedia.org/wiki/Sistemas_microelectromec%C3%A1nicos

[18] http://www.solidworks.es/

[19] http://we.easyelectronics.ru/part/modul-pitaniya-ywrobot-breadboard-power-supply-mb-v2-mb102.html

[20] http://www.advanced-monolithic.com/pdf/ds1117.pdf

[21] http://pdf1.alldatasheet.es/datasheet-pdf/view/329387/CHENG-YI/IN4007.html

[22] https://en.wikipedia.org/wiki/Motor_controller

[23] https://es.wikipedia.org/wiki/Modulaci%C3%B3n_por_ancho_de_pulsos

[24] http://www.ti.com/lit/ds/symlink/l293.pdf

[25] http://www.mouser.com/ds/2/389/l78-974043.pdf

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

[27] https://www.invensense.com/products/motion-tracking/6-axis/mpu-6050/

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

[29] https://www.arduino.cc/en/reference/wire

[30] https://www.invensense.com/products/motion-tracking/6-axis/mpu-6050/

[31] https://github.com/jrowberg/i2cdevlib

[32] http://cdn.makezine.com/uploads/2014/03/hc_hc-05-user-instructions-bluetooth.pdf

[33] http://cnx.org/contents/ul3tl2dl@1/Serial-Port-Communication

[34] https://www.arduino.cc/en/Reference/SoftwareSerial

[35] http://www.atmel.com/products/microcontrollers/avr/megaavr.aspx

[36] https://www.arduino.cc/

[37] https://www.arduino.cc/en/Main/ArduinoBoardUno

[38] https://www.arduino.cc/en/Main/ArduinoBoardNano

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

[40] https://www.arduino.cc/en/Main/ArduinoBoardYun

[41] https://www.arduino.cc/en/reference/wire

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

[43] https://www.arduino.cc/en/Reference/SoftwareSerial

[44] https://play.google.com/store/apps/details?id=project.bluetoothterminal&hl=es

[45] https://play.google.com/store/apps/details?id=nextprototypes.BTSerialController&hl=es_419

[46] http://brettbeauregard.com/blog/wp-content/uploads/2012/07/Gu%C3%ADa-de-uso-PID-para-Arduino.pdf

Anexo Presupuesto

En este presupuesto se pasará a detallar los costes que ha tenido los diferentes componentes del robot, no teniendo en cuenta el coste de la mano de obra para el diseño y la fabricación. Por tanto, únicamente se van a tener en cuenta los dispositivos electricos y una aproximación al coste de la estructura impresa, aunque en nuestro caso se imprimió en el laboratorio de electronica basica de la UPNA.

Dado que se deseaba que los componentes unidos a la placa se pudieran quitar, se emplearon unos conectores especiales que tienen los siguientes costes:

Aplicación	**Conector de 4 pines**			
	Referencia RS	**Precio unitario**	**Nº**	**Precio Total**
Conector macho 4 pines	483-8483	0,278 €	1	0,278 €
Carcasa conector 4 pines	296-4956	0,260 €	1	0,260 €
Contacto de crispada	233-3009	0,124 €	4	0,496 €

Precio conector 4 pines	**1,034 €**

Tabla 11. Presupuesto del conector de 4 pines.

Aplicación	**Conector de 2 pines**			
	Referencia RS	**Precio unitario**	**Nº**	**Precio Total**
Conector macho 2 pines	483-8461	0,130 €	1	0,130 €
Carcasa conector 2 pines	296-4934	0,190 €	1	0,190 €
Contacto de crimpado	233-3009	0,124 €	2	0,248 €

Precio conector 2 pines	**0,568 €**

Tabla 12. Presupuesto del conector de 2 pines.

Con esto definido, se procede con el desglosamiento del presupuesto final, en el cual se incluyen los tres apartados eléctricos que hay en el proyecto, así como una estimación del coste de la estructura impresa.

Aplicación	Circuito de Alimentación			
	Referencia RS	Precio unitario	Nº	Precio Total
Regulador de tensión (L7805ACD2T)	686-9530	0,41 €	1	0,41 €
Batería recargables	-	11,55 €	2	23,10 €
Condensador 0,1µF	264-4416	0,02 €	1	0,02 €
Condensador 0,33µF	698-3557	0,02 €	1	0,02 €
Conector 2 pines	-	0,54 €	1	0,54 €
		Total control		24,09 €

Aplicación	Circuito de control			
	Referencia RS	Precio unitario	Nº	Precio Total
Arduino nano	-	6,39 €	1	6,39 €
Sensor MPU-6050	-	2,91 €	1	2,91 €
Módulo HC-05	-	4,61 €	1	4,61 €
LED RGB SODIAL®	-	0,06 €	1	0,06 €
Tiras de pines	267-7400	0,48 €	1	0,48 €
Resistencia SMD 330Ω	618-3195	0,00 €	1	0,00 €
Conector 4 pines	-	0,97 €	2	1,94 €
		Total control		16,39 €

Aplicación	Circuito del driver			
	Referencia RS	Precio unitario	Nº	Precio Total
Motores DC con encoder [46]	-	9,27 €	2	18,54 €
Driver L298	636-384	3,49 €	1	3,49 €
Diodos PMEG2020EH	485-521	0,07 €	8	0,53 €
Condensador 100nF	264-4416	0,02 €	1	0,02 €
Condensador electrolítico 470µF	711-1110	0,09 €	1	0,09 €
Conector 2 pines	-	0,54 €	2	1,08 €
Disipador de calor	234-2564	1,01 €	1	1,01 €
		Total control		24,75 €

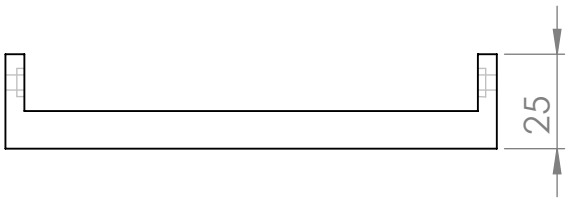
Aplicación	Coste de Impresión estimada. Fuente		
	Precio unitario	Nº	Precio
Base 1	94,31 €	1	94,31 €
Base 2	6,16 €	1	6,16 €
Base 3	21,06 €	1	21,06 €
Lateral 1	23,53 €	1	23,53 €
Lateral 2	23,10 €	1	23,10 €
Motores	50,84 €	1	50,84 €
Ruedas	8,16 €	2	16,32 €
Abrazaderas	5,59 €	2	11,18 €

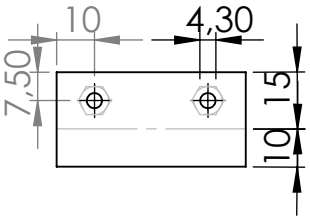
Total impresión	246,50 €

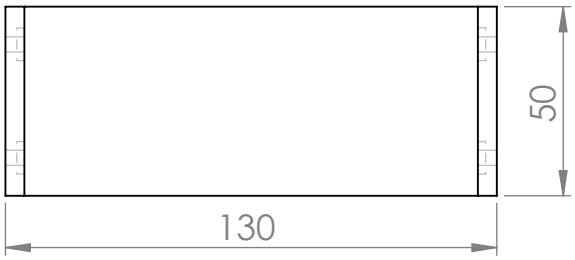
Coste total del robot	311,73 €

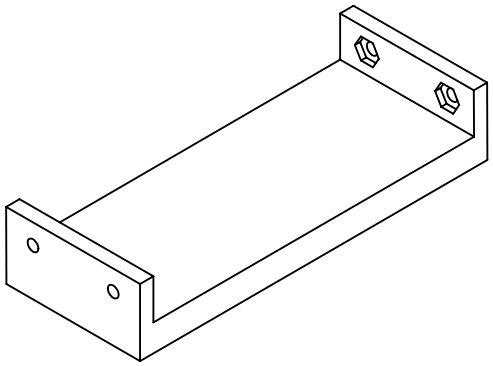
Tabla 13. Colección de tablas del presupuesto.

Anexo Planos

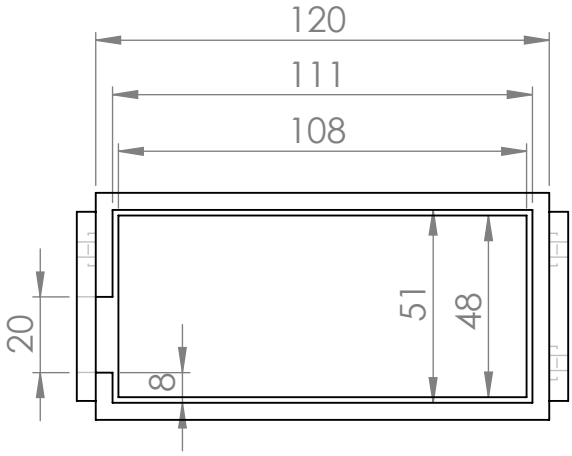


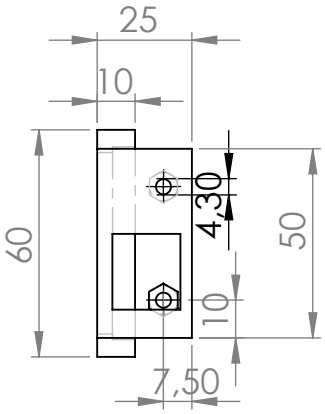


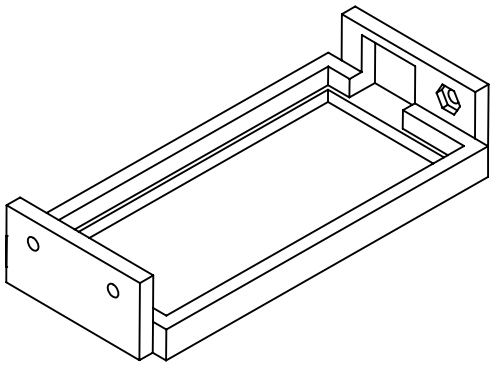




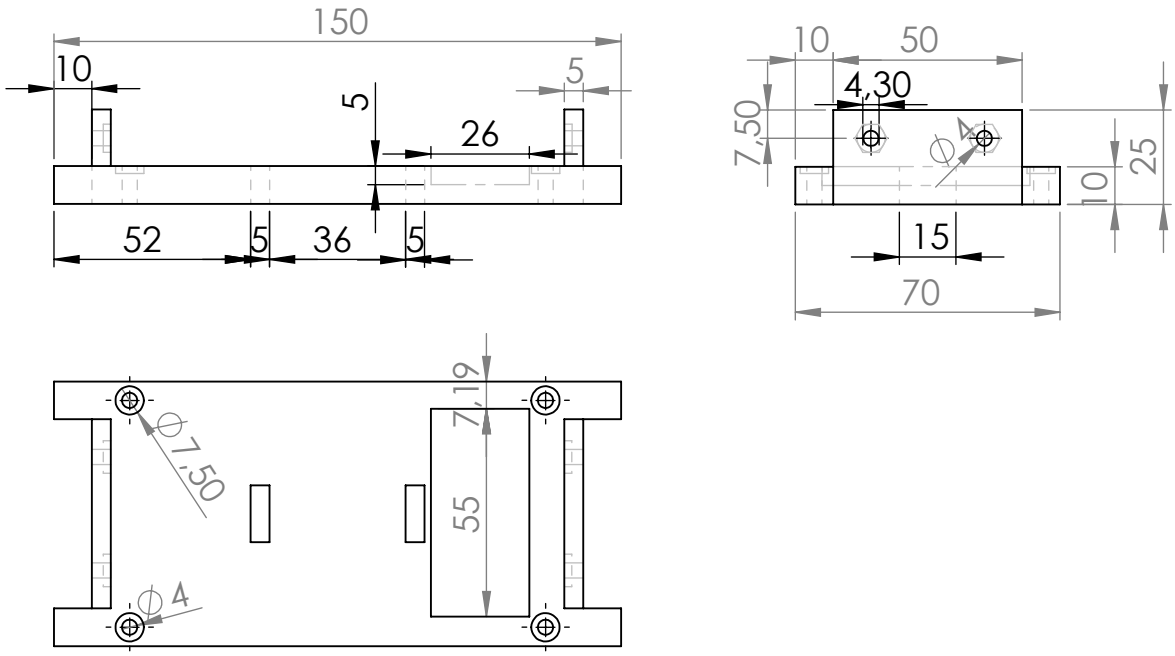
Universidad Pública de Navarra *Nafarroako Unibertsitate Publikoa*	**E.T.S.I.I.T.** **INGENIERO TECNICO INDUSTRIAL ELÉCTRICO.**	DEPARTAMENTO: **DEPARTAMENTO DE INGENIERIA ELÉCTRICA Y ELECTÓNICA**
PROYECTO: **Robot auto-balanceado basado en Arduino**		REALIZADO: **Gracia Moisés, Ander** FIRMA:
PLANO: Base 1		FECHA: 23/05/17 | ESCALA: | Nº PLANO: 1

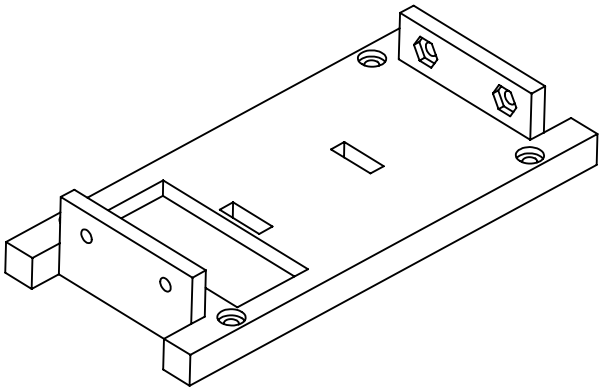




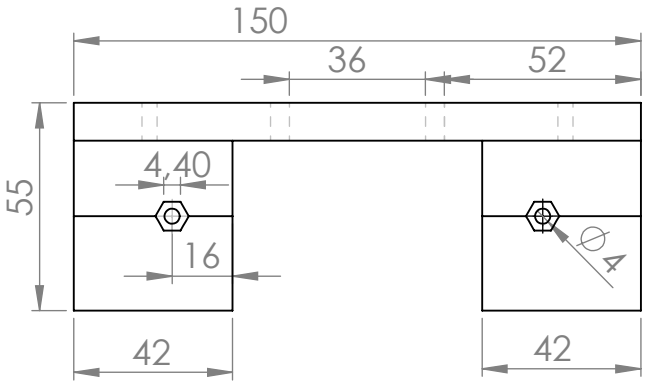


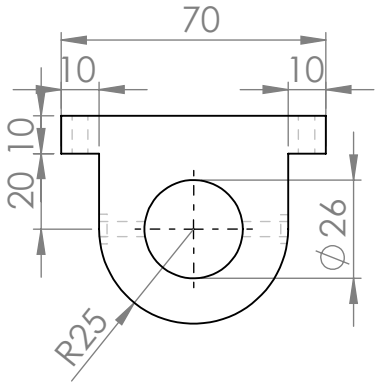
Universidad Pública de Navarra *Nafarroako Unibertsitate Publikoa*	**E.T.S.I.I.T.** **INGENIERO TECNICO INDUSTRIAL ELÉCTRICO.**	DEPARTAMENTO: **DEPARTAMENTO DE INGENIERIA ELÉCTRICA Y ELECTÓNICA**
PROYECTO: **Robot auto-balanceado basado en Arduino**		REALIZADO: **Gracia Moisés, Ander** FIRMA:
PLANO: Base 2	FECHA: 23/05/17	ESCALA:
		Nº PLANO: 2

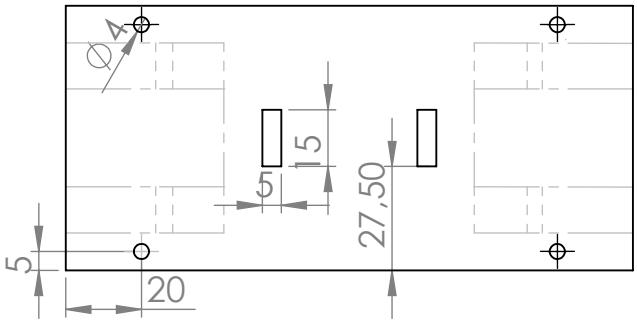


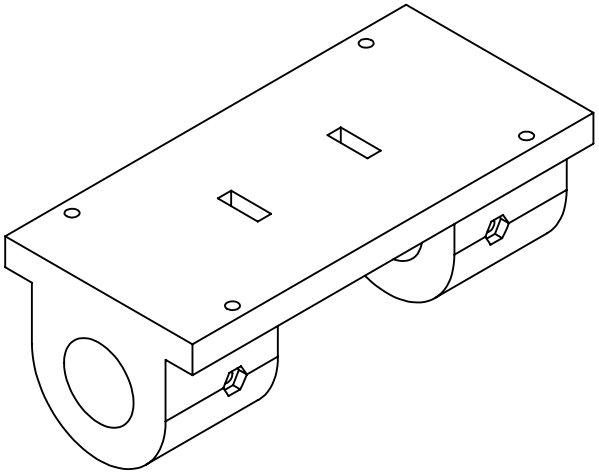


Universidad Pública de Navarra *Nafarroako Unibertsitate Publikoa*	**E.T.S.I.I.T.** **INGENIERO TECNICO INDUSTRIAL ELÉCTRICO.**	DEPARTAMENTO: **DEPARTAMENTO DE INGENIERIA ELÉCTRICA Y ELECTÓNICA**
PROYECTO: **Robot auto-balanceado basado en Arduino**		REALIZADO: **Gracia Moisés, Ander** FIRMA:
PLANO: Base 3		FECHA: 23/05/17 | ESCALA: | Nº PLANO: 3

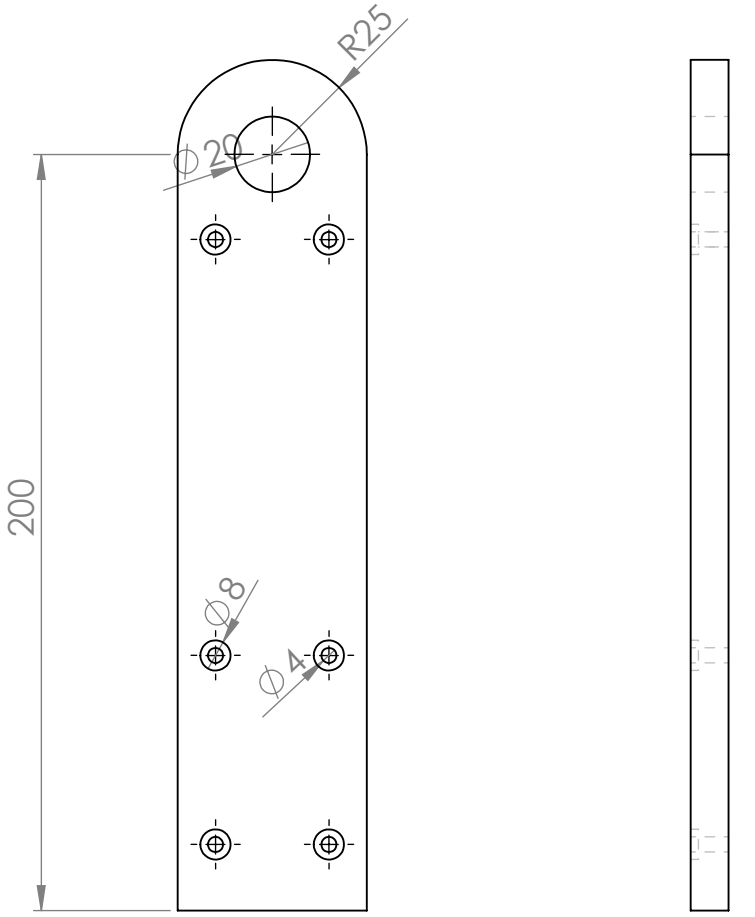


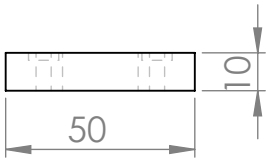


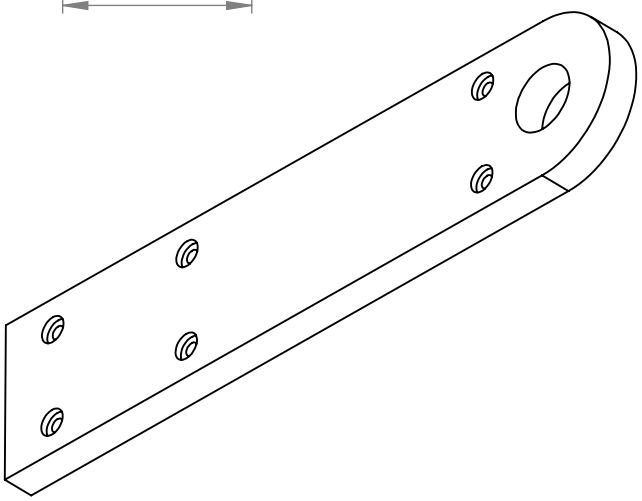




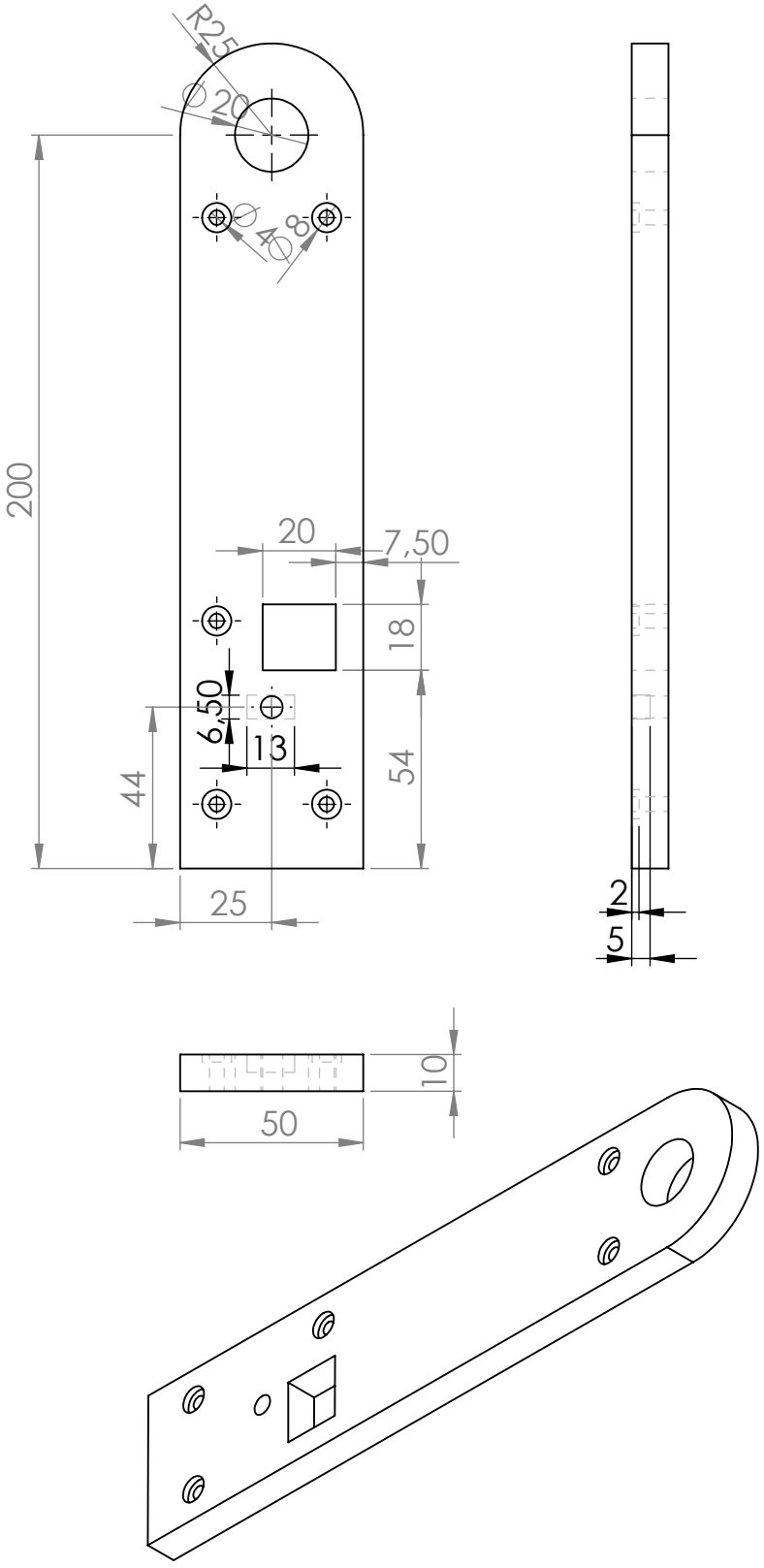
Universidad Pública de Navarra *Nafarroako Unibertsitate Publikoa*	**E.T.S.I.I.T.** **INGENIERO TECNICO INDUSTRIAL ELÉCTRICO.**	DEPARTAMENTO: **DEPARTAMENTO DE INGENIERIA ELÉCTRICA Y ELECTÓNICA**
PROYECTO: **Robot auto-balanceado basado en Arduino**		REALIZADO: **Gracia Moisés, Ander** FIRMA:
PLANO: Sujección de los motores	FECHA: 23/05/17 / ESCALA: / Nº PLANO: 4	



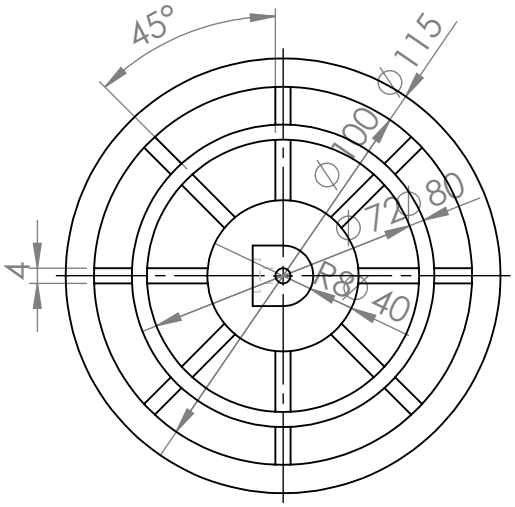


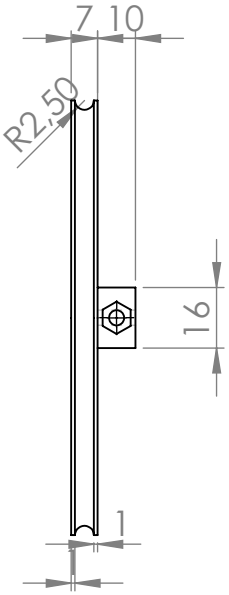


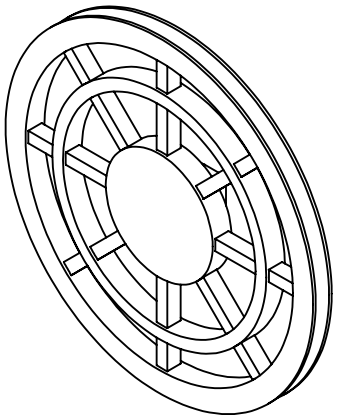
Universidad Pública de Navarra *Nafarroako Unibertsitate Publikoa*	**E.T.S.I.I.T.** **INGENIERO TECNICO INDUSTRIAL ELÉCTRICO.**	DEPARTAMENTO: **DEPARTAMENTO DE INGENIERIA ELÉCTRICA Y ELECTÓNICA**
PROYECTO: **Robot auto-balanceado basado en Arduino**		REALIZADO: **Gracia Moisés, Ander** FIRMA:
PLANO: Lateral 1	FECHA: 23/05/17	ESCALA:
		Nº PLANO: 5



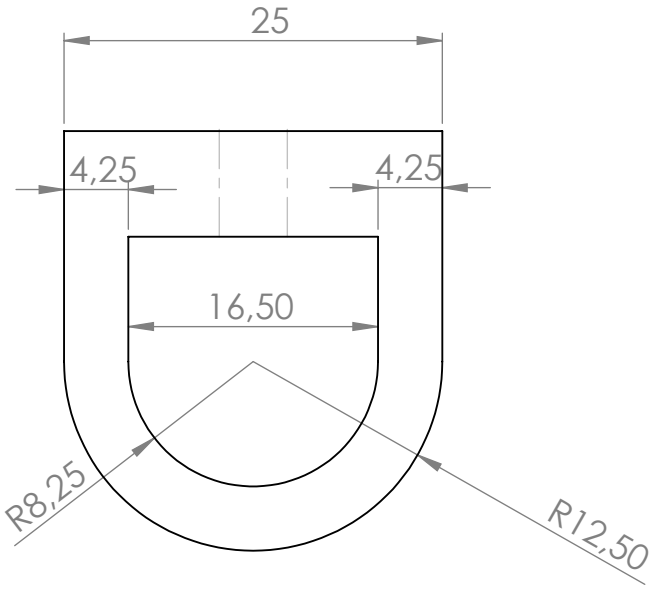
Universidad Pública de Navarra *Nafarroako Unibertsitate Publikoa*	**E.T.S.I.I.T.** **INGENIERO TECNICO INDUSTRIAL ELÉCTRICO.**	DEPARTAMENTO: **DEPARTAMENTO DE INGENIERIA ELÉCTRICA Y ELECTÓNICA**
PROYECTO: **Robot auto-balanceado basado en Arduino**		REALIZADO: **Gracia Moisés, Ander** FIRMA:
PLANO: Lateral 2	FECHA: 23/05/17	ESCALA:
		Nº PLANO: 6

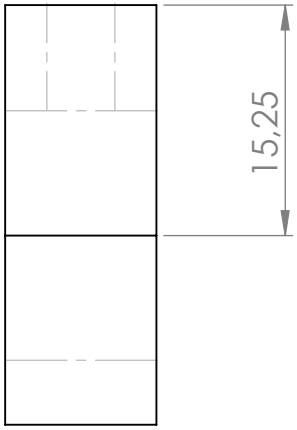


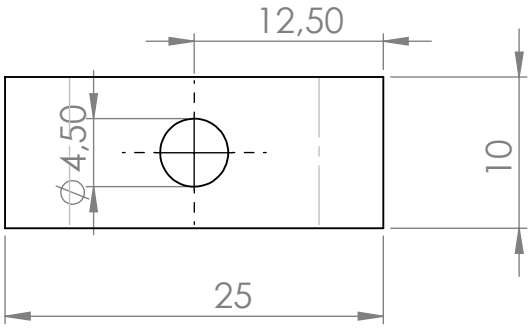


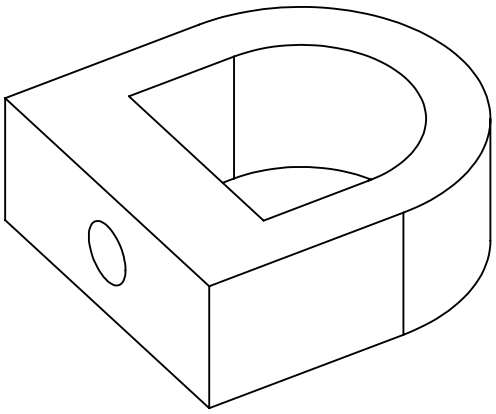


Universidad Pública de Navarra *Nafarroako Unibertsitate Publikoa*	**E.T.S.I.I.T.** **INGENIERO TECNICO INDUSTRIAL ELÉCTRICO.**	DEPARTAMENTO: **DEPARTAMENTO DE INGENIERIA ELÉCTRICA Y ELECTÓNICA**
PROYECTO: **Robot auto-balanceado basado en Arduino**		REALIZADO: **Gracia Moisés, Ander** FIRMA:
PLANO: Ruedas		FECHA: 23/05/17 | ESCALA: | Nº PLANO: 7





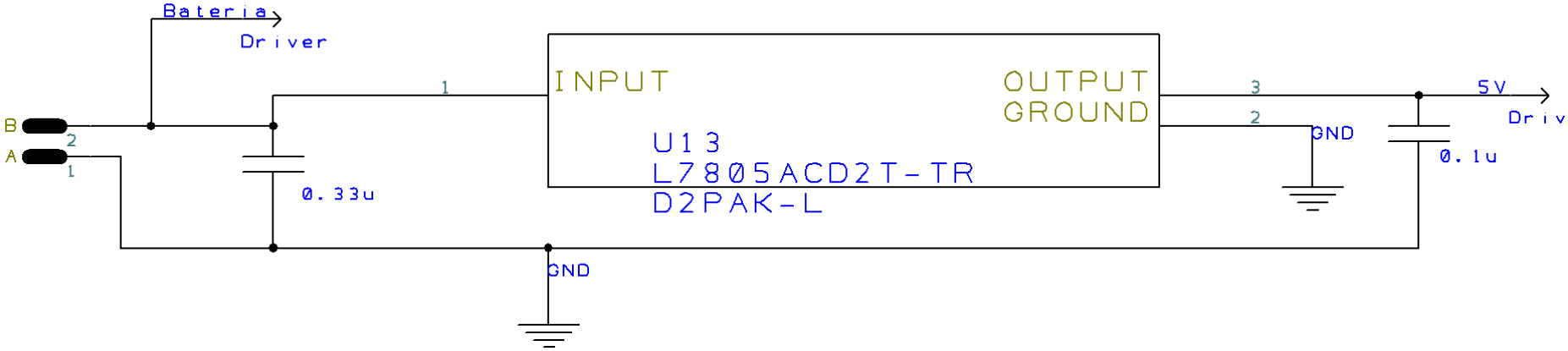




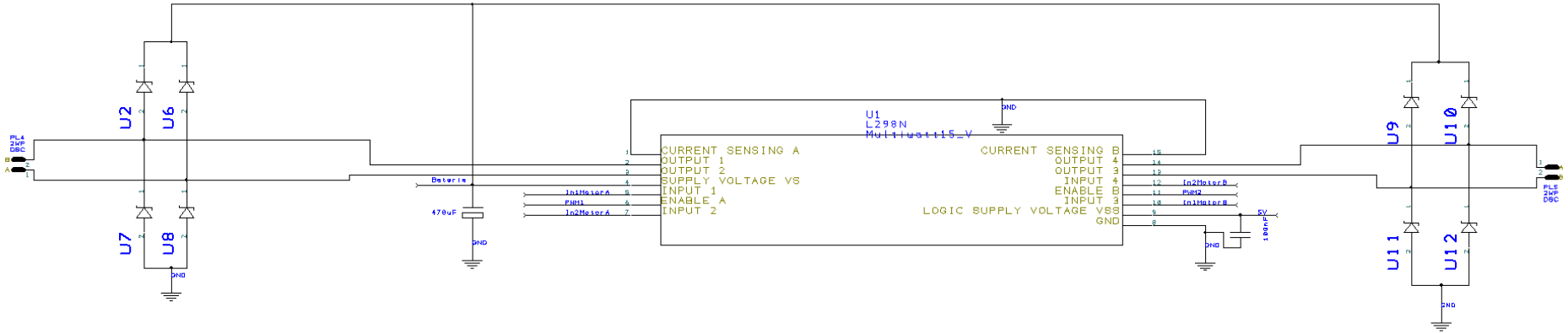
Universidad Pública de Navarra *Nafarroako Unibertsitate Publikoa*	**E.T.S.I.I.T.** **INGENIERO TECNICO INDUSTRIAL ELÉCTRICO.**	DEPARTAMENTO: **DEPARTAMENTO DE INGENIERIA ELÉCTRICA Y ELECTÓNICA**
PROYECTO: **Robot auto-balanceado basado en Arduino**		REALIZADO: **Gracia Moisés, Ander** FIRMA:
PLANO: Abrazadera para la sujección de las ruedas	FECHA: 23/05/17	ESCALA:
		Nº PLANO: 8

Anexo Esquemas Eléctricos

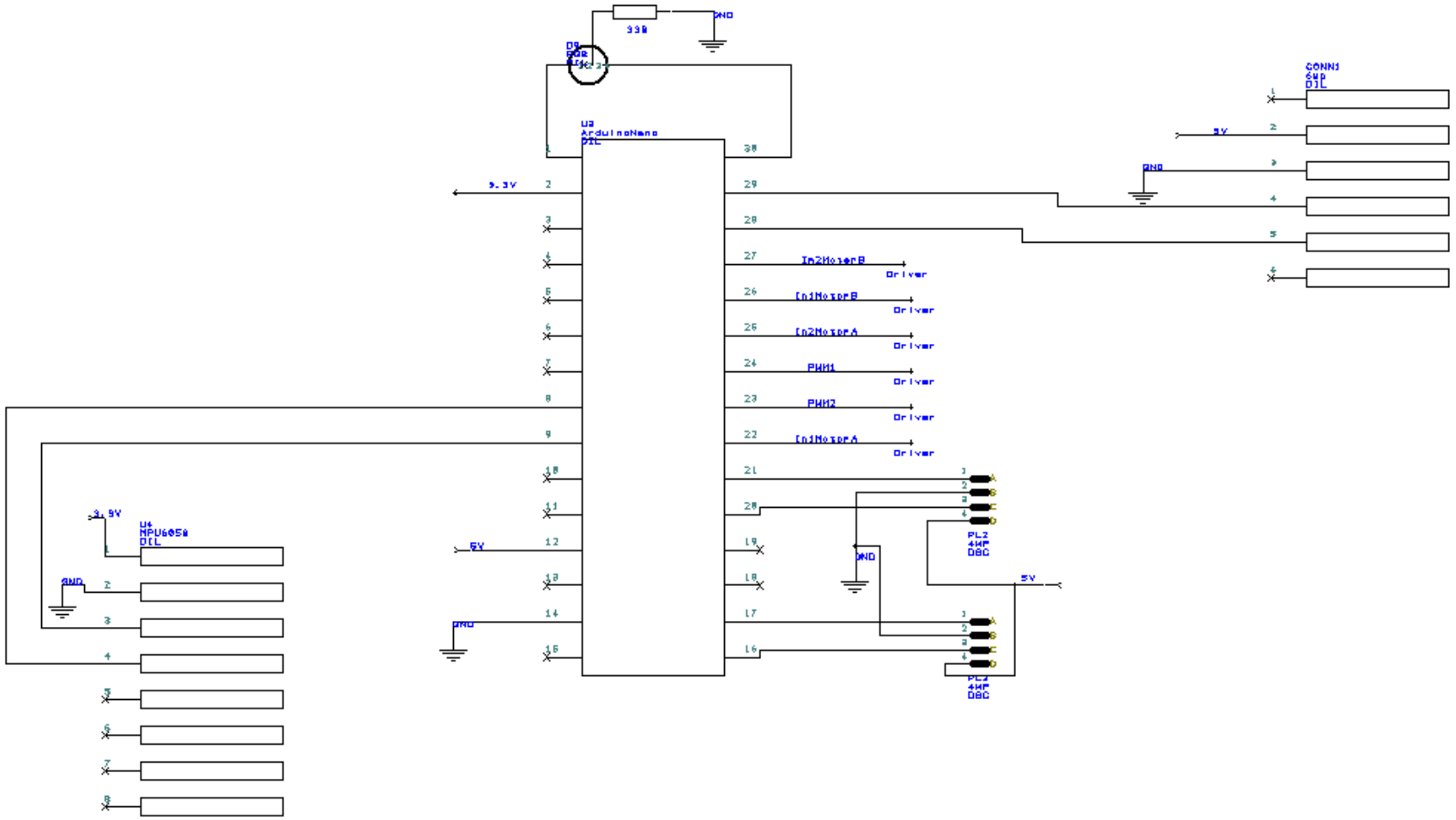
Circuito de alimentación



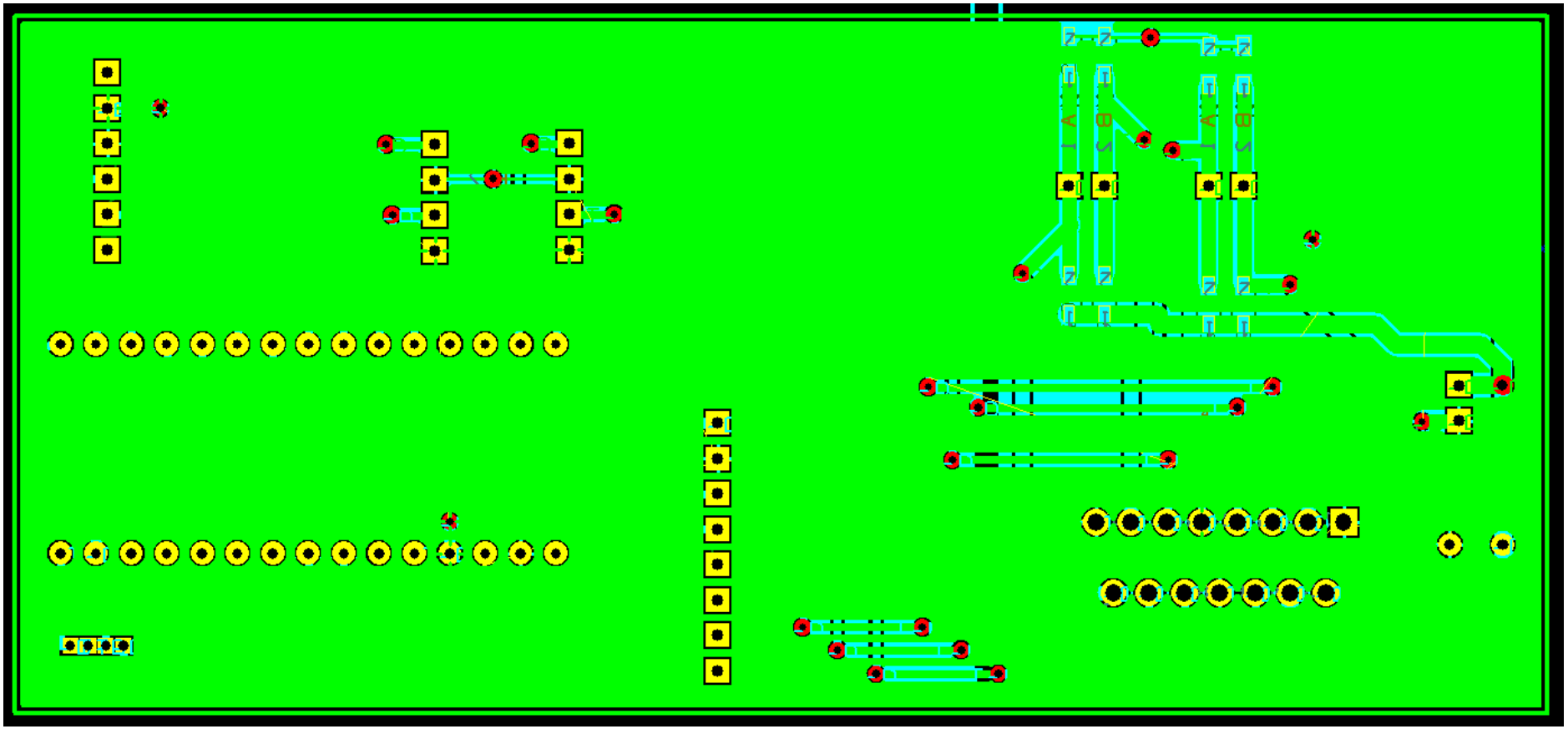
Circuito de Driver



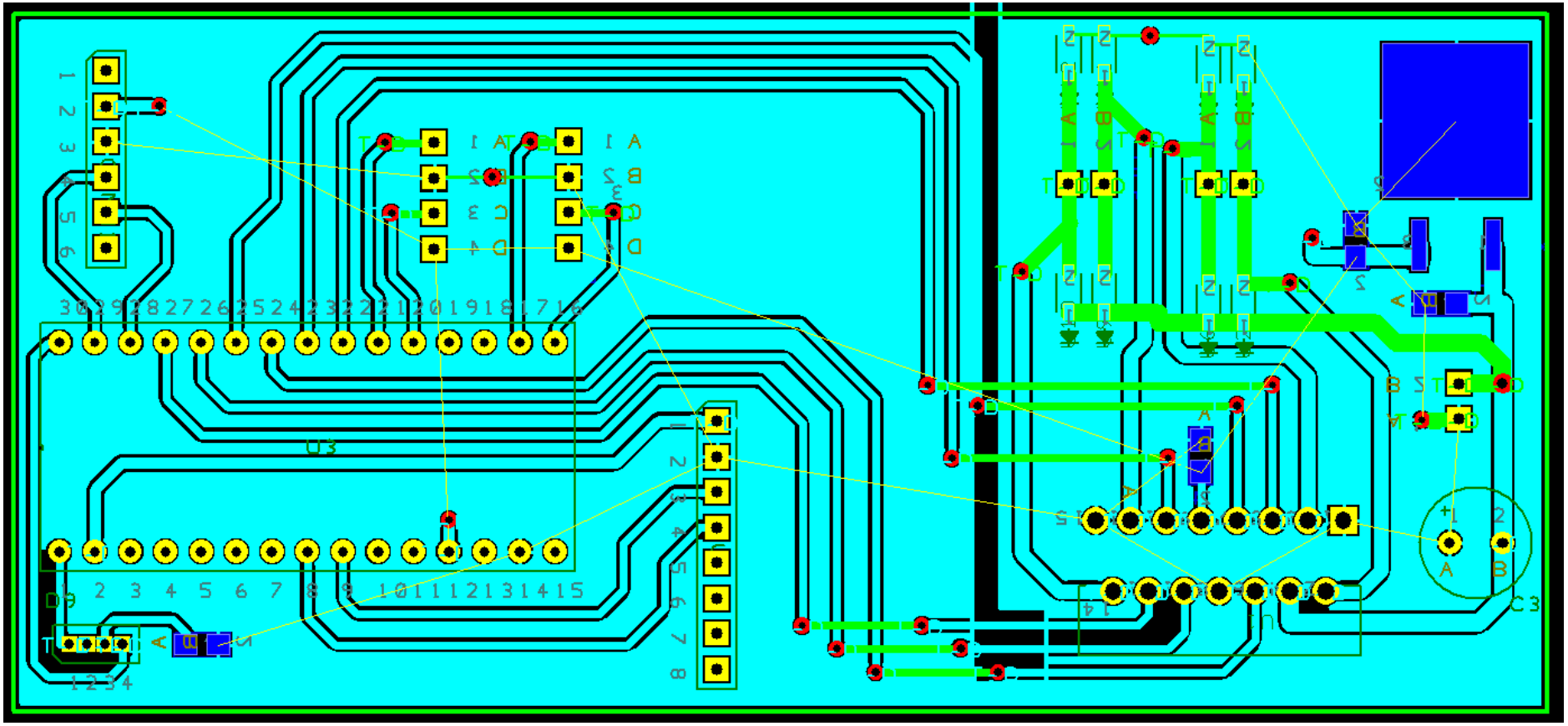
Circuito de control



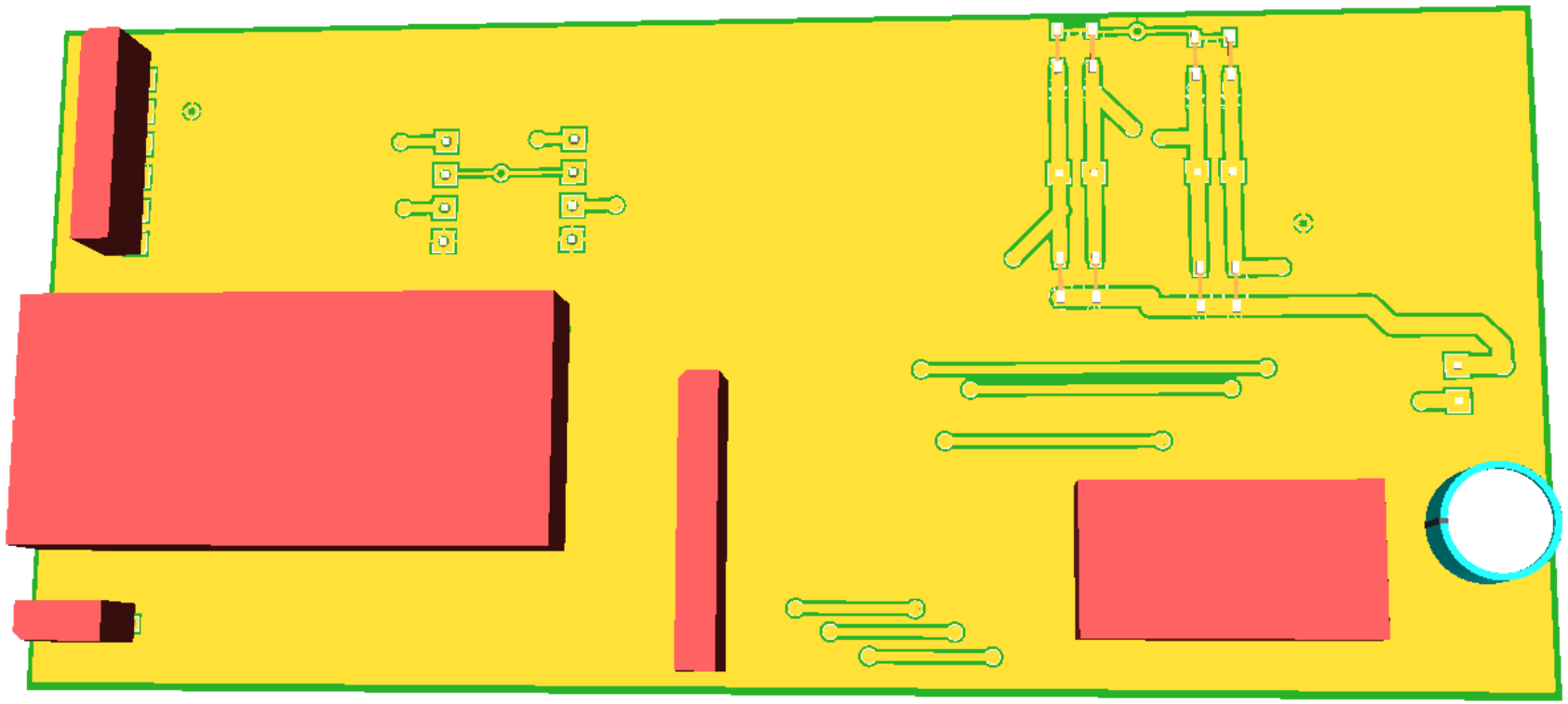
Capa superior PCB.



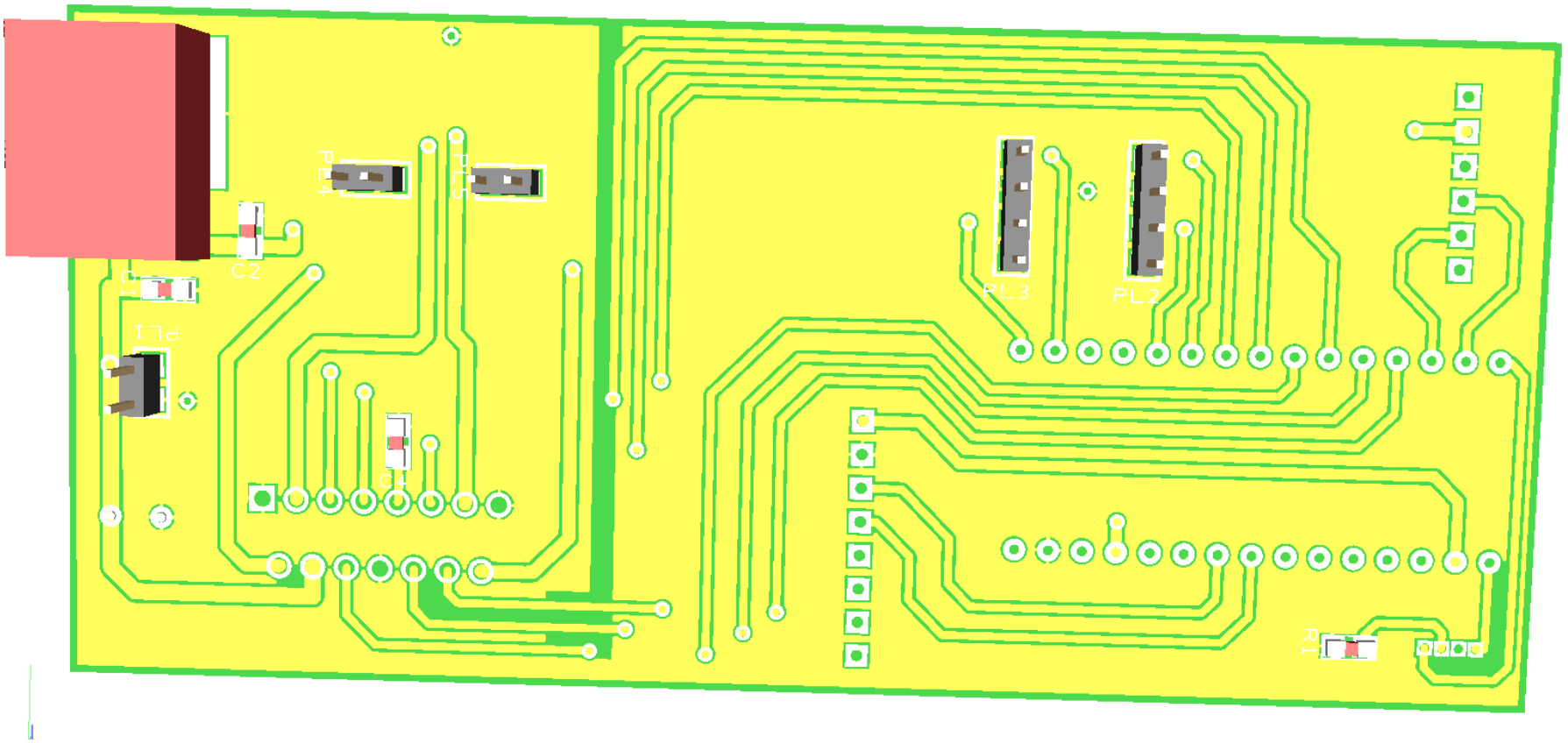
Capa inferior PCB.



Vista 3D superior.



Vista 3D inferior.



Anexo Software

```
////////////////////////////////////////////////////////////////////
//////////INICIALIZACION DE LIBRERIAS Y VARIABLES GLOBALES//////////
////////////////////////////////////////////////////////////////////

//Incluimos la librería Wire.h
#include<Wire.h>

//Incluimos la librería SoftwareSerial para comunicarnos con el modulo
bluetooth HC-05
#include <SoftwareSerial.h>

//Pines de comunicación Bluetooth
#define RXpin 11
#define TXpin 10

// Pines de control PWM de los motores
#define Motor1 6
#define Motor2 5

// Pines de control del giro del motor 1
#define M1_InA  7
#define M1_InB  4
// Pines de control del giro del motor 2
#define M2_InA  8
#define M2_InB  9

// Pines del encoder del motor 1
#define M1_PhaseA  2
#define M1_PhaseB  3

// Pines del encoder del motor 2
#define M2_PhaseA  0
#define M2_PhaseB  1

// Pines del led
#define Rojo  12
#define Azul  13

//Se define el valor de la histéresis que queremos poner
#define Ang_Histeresis 1.5

// Dirección I2C del sensor MPU6050
#define MPU_addr 0x68
// Tamaño array recepción float de bluetooth
#define SIZE_ARRAY 9

//Variable de tiempo muestreo del PID
#define Tmuestreo_posicion 3
#define Tmuestreo_velocidad 50

// Variables globales para la comunicación bluetooth
SoftwareSerial bluetoothSerial(RXpin, TXpin); // Puerto serie de
comunicación
byte contador_auxiliar = 0; // Variable contador para la recepción de
datos
char auxiliar[SIZE_ARRAY] = {0x00}; // Array de recepción de datos

// Variables globales para el acelerómetro MPU6050
#define FC  0.95 // Constante para el filtro combinacional
long tiempo_prev; // Tiempo en que se ha producido la medida anterior
del acelerómetro
```

```
float ang_x_prev, ang_y_prev; // Ángulos en los ejes x e y en la
medida anterior
float ang_x, ang_y; // Ángulos en los ejes x e y en la medida actual

// Variables globales para PID_posicion
float U_control_posicion = 0;
float erroranterior_posicion = 0;
float angulo_x_anterior;
float lastTime_posicion = 0; // Tiempo en que se realizó la
actualización anterior del PID
// Se pueden hacer con define
float Kp_posicion = 25, Ki_posicion = 0, Kd_posicion = 1.5; //
Constantes proporcional, integral y derivativa

//Variable globales PID_velocidad
float U_control_velocidad = 0;
float lastTime_velocidad = 0;
float Kp_velocidad = 0.15, Ki_velocidad = 0.8;

// Variables globales para el encoder
int c = 0;
float tiempo_prev_encoder = 0;
float velocidad = 0;

// Variables para el control remoto del robot
float velocidadDeseada = 0;
int giro = 0; //variable que puede tener tres valores diferentes: "-1"
= giro izquierda, "1" = giro derecha y "0" = no gira

////////////////////////////////////////////////////////////////////////
//////////////////////////////////SetUp/////////////////////////////////
////////////////////////////////////////////////////////////////////////

void setup() {
  setupAcelerometro();
  setupMotor();
  setupBluetooth();
  setupLED();
  //Serial.begin(115200);
}

// Inicializamos la comunicación y los parámetros con el acelerómetro
void setupAcelerometro() {
  Wire.begin();
  Wire.beginTransmission(MPU_addr); //Iniciamos la comunicación con el
sensor (dirección 0x6B)
  Wire.write(0x6B);  //En el registro 0x6B (PWR_MGMT_1) => Modo de
energía, reloj interno y sensor de temperatura
  Wire.write(0);  //En el registro PWR_MGMT_1 se escribe 000000
  Wire.endTransmission(true); //Dejamos de escribir

  Wire.beginTransmission(MPU_addr); //Iniciamos la comunicación con el
sensor (dirección 0x6B)
  Wire.write(0x1B); //En el registro 0x1B (GYRO_CONFIG) ± 250 °/s
  Wire.write(0);  //En el registro 0x1B se escribe 000000
  Wire.endTransmission(true); //Dejamos de escribir

  Wire.beginTransmission(MPU_addr); //Iniciamos la comunicación con el
sensor (dirección 0x6B)
  Wire.write(0x1C); //En el registro 0x1C (ACCEL_CONFIG) ± 2g
  Wire.write(0);  //En el registro 0x1C se escribe 000000
```

```
  Wire.endTransmission(true); //Dejamos de escribir
}

// Inicializamos el control de los motores
void  setupMotor() {
  // Declaramos los pines de control PWM de los motores
  pinMode(Motor1, OUTPUT);
  pinMode(Motor2, OUTPUT);
  // Declaramos los pines para el control de giro de cada motor
  pinMode(M1_InA, OUTPUT);
  pinMode(M1_InB, OUTPUT);
  pinMode(M2_InA, OUTPUT);
  pinMode(M2_InB, OUTPUT);

  //  // Declaramos los pines del encoder y su interrupción
  pinMode(M1_PhaseA, INPUT);
  attachInterrupt(digitalPinToInterrupt(M1_PhaseA),
leerInformacionEncoderM1, RISING); //interrupción (timer 0)patilla 2
flanco de subida
}

// Inicializamos la comunicación bluetooth
void  setupBluetooth() {

  //Declaracion pines HC-05
  pinMode(RXpin, INPUT);
  pinMode(TXpin, OUTPUT);

  // Inicio comunicación bluetooth
  bluetoothSerial = SoftwareSerial(RXpin, TXpin);
  bluetoothSerial.begin(9600);
}

//Inicializamos el LED
void setupLED(){
  pinMode(Rojo, OUTPUT);
  pinMode(Azul, OUTPUT);
}

////////////////////////////////////////////////////////////////////////
/////////////////////////////////Loop///////////////////////////////////
////////////////////////////////////////////////////////////////////////

void loop() {

  leerInformacionsAcelerometro();
  leerInformacionBluetooth();
  RGB(ang_x);

////Variables para el calculo del PID mediante Ziegler-Nichols
//  float TiempoZiegleNichols = millis();
//  Serial.print(ang_x);Serial.print("|");
//  Serial.println(TiempoZiegleNichols);

  //Calculo de la U_control_velocidad mediante la velocidad de
referencia
  float error_velocidad=velocidadDeseada-velocidad;  //Calculo del
error de velocidad, depende de la velocidad a la que se quiere que se
desplace el robot y la velocidad que lleva.
  PID_velocidad(error_velocidad);
```

```
  //Calculo de la U_control_posicion mediante el control PID
  float error_posicion = U_control_velocidad - ang_x;
  PID_posicion(error_posicion);

  // Si el robot está de pie, y no tiene que girar ajustamos el
control
  if (((ang_x > (U_control_velocidad+Ang_Histeresis)) && (ang_x < 30))
|| ((ang_x < (U_control_velocidad-Ang_Histeresis)) && (ang_x > -30) ))
  {
     controlMotores(-U_control_posicion,giro);
  }
  // Si se ha caído, paramos los motores
  else
  {
    U_control_posicion = 0;
    controlMotores(0,giro);
  }
}

////////////////////////////////////////////////////////////////////////
/////////////////OBTENER INFORMACION DEL BLUETOOTH//////////////////////
////////////////////////////////////////////////////////////////////////


// Lectura de la información recibida por bluetooth
void   leerInformacionBluetooth()
{
  byte state = 0; // Estado de la comunicación
  if (bluetoothSerial.available())
  {
    state = bluetoothSerial.read();
    //Serial.println(state, HEX);
    if (((((state >= 0x30) && (state <= 0x39)) || (state == 0x2E))) ||
        ((contador_auxiliar == 0) && (state == 0x2D)))
    {
      auxiliar[contador_auxiliar] = state;
      contador_auxiliar++;

      // Número demasiado largo
      if (contador_auxiliar > SIZE_ARRAY)
      {
        bluetoothSerial.println("Numero enviado demasiado largo");
        contador_auxiliar = 0;
        memset (auxiliar, 0x00, sizeof (auxiliar));
      }
    }
    // Al recibir un LF (suponemos comunicación con LF únicamente como
final de línea)
    else if (state == 0x0D)
    {
      Ki_posicion = atof(auxiliar);
      bluetoothSerial.print("Ki_posicion = ");
      bluetoothSerial.println(Ki_posicion, 6);

      contador_auxiliar = 0;
      // Poner todos los elementos del array a cero
      memset (auxiliar, 0x00, sizeof (auxiliar));
    }

    //Al recibir una "a"(0x61) significa que el robot se debe mover
hacia adelante
```

```
    else if (state == 0x61)
    {
      velocidadDeseada = 2;
      giro = 0;
    }
    //Al recibir una "b"(0x62) significa que el robot se debe mover
hacia atrás
    else if (state == 0x62)
    {
      velocidadDeseada = -3;
      giro = 0;
    }
    //Al recibir una "c"(0x63) significa que el robot debe girar a la
izquierda
    else if (state == 0x63)
    {
      velocidadDeseada = 1;
      giro = -1;
    }
    //Al recibir una "d"(0x64) significa que el robot debe girar a la
derecha
    else if (state == 0x64)
    {
      velocidadDeseada = 1;
      giro = 1;
    }
    //Al recibir una "e"(0x65) significa que el robot se debe parar en
el sitio
    else if (state == 0x65)
    {
      velocidadDeseada = 0;
      giro = 0;
    }

    // Condición de error. Olvidar el valor y reiniciar. Esto no es
muy correcto, pero peor es nada
    else
    {
      bluetoothSerial.println("Error en la recepción");
      contador_auxiliar = 0;
      memset (auxiliar, 0x00, sizeof (auxiliar));
    }
  }
}

////////////////////////////////////////////////////////////////////
//////////////////OBTENCION DE DATOS DEL MPU-6050///////////////////
////////////////////////////////////////////////////////////////////

// Lectura información del acelerómetro e integración de la velocidad
de giro
void leerInformacionsAcelerometro()
{
  int16_t AcX, AcY, AcZ, Tmp, GyX, GyY, GyZ; // Datos capturados con
el acelerómetro
  float dt; // Intervalo de tiempo que ha pasado entre la medida
anterior del acelerómetro y la actual


  //Nos conectamos al Sensor por comunicación Wire
  Wire.beginTransmission(MPU_addr);
```

```
  Wire.write(0x3B); //Empezamos con el registro 0x3B => ACCEL_XOUT_H
  Wire.endTransmission(false);
  Wire.requestFrom(MPU_addr, 14, true); //Le pedimos al sensor
conectado a la dirección 68 que nos de los 14 registros desde el
indicado antes

  //Cada registro nos da 8 bits pero la aceleración ocupa 16 hay que
hacer una operación de desplazamiento y suma
  AcX = Wire.read() << 8 | Wire.read(); // 0x3B (ACCEL_XOUT_H) & 0x3C
(ACCEL_XOUT_L)
  AcY = Wire.read() << 8 | Wire.read(); // 0x3D (ACCEL_YOUT_H) & 0x3E
(ACCEL_YOUT_L)
  AcZ = Wire.read() << 8 | Wire.read(); // 0x3F (ACCEL_ZOUT_H) & 0x40
(ACCEL_ZOUT_L)
  Tmp = Wire.read() << 8 | Wire.read(); // 0x41 (TEMP_OUT_H) & 0x42
(TEMP_OUT_L)
  GyX = Wire.read() << 8 | Wire.read(); // 0x43 (GYRO_XOUT_H) & 0x44
(GYRO_XOUT_L)
  GyY = Wire.read() << 8 | Wire.read(); // 0x45 (GYRO_YOUT_H) & 0x46
(GYRO_YOUT_L)
  GyZ = Wire.read() << 8 | Wire.read(); // 0x47 (GYRO_ZOUT_H) & 0x48
(GYRO_ZOUT_L)

  //Calcular ángulos con acelerómetro
  float accel_ang_x = atan(AcY / sqrt(pow(AcX, 2) + pow(AcZ, 2))) *
(180.0 / 3.14);

  dt = (millis() - tiempo_prev) / 1000;
  tiempo_prev = millis();
  //Calcular ángulos de rotación con giroscopio y filtro
complementario
  ang_x = FC * (ang_x_prev + (GyX / 131) * dt) + (1 - FC) *
accel_ang_x;
  ang_x_prev = ang_x;

}

////////////////////////////////////////////////////////////////////
//////////////////////CALCULO DEL PID DE POSICION///////////////////
////////////////////////////////////////////////////////////////////

void PID_posicion(float error_posicion)
{
  float t_posicion = millis();
  float tchange_posicion = t_posicion - lastTime_posicion; //Calculo
del tiempo del PID

  //Condición para conseguir que el PID se ejecute en periodos de
tiempo constantes
  if (tchange_posicion >= Tmuestreo_posicion)
  {
    //Serial.println(tchange_posicion);
    //Calculo de la parte proporcional
    float PID_prop_posicion = Kp_posicion * error_posicion;

    //Cálculo de la parte integral
    float errorSum_posicion = errorSum_posicion + error_posicion *
Tmuestreo_posicion / 1000; //Calculo del error integral
    float PID_int_posicion = Ki_posicion * errorSum_posicion;
//Calculo del termino integral
    //Condicion Anti WindUp para el termino integral
```

```
    if (PID_int_posicion > 255)
    {
      PID_int_posicion = 255;
    }
    else if (PID_int_posicion < -255)
    {
      PID_int_posicion = -255;
    }

    //Código con la solución del Derivativer Kick implementado.
    //Calculo de la parte derivativa
    float inputd_posicion = (ang_x - angulo_x_anterior) * 1000 /
(Tmuestreo_posicion); //Calculo de la derivada de la entrada (input)
    float PID_der_posicion = - Kd_posicion * inputd_posicion;
//Calculo de la parte derivativa de un PID

    //Generación de la acción de control
    U_control_posicion = PID_prop_posicion + PID_int_posicion +
PID_der_posicion;

    //Condición Anti WindUp en la acción de control
    if (U_control_posicion >= 255.0)
    {
      U_control_posicion = 255.0;
    }
    else if (U_control_posicion <= -255.0)
    {
      U_control_posicion = -255.0;
    }

    erroranterior_posicion = error_posicion;
    lastTime_posicion = t_posicion;
    angulo_x_anterior = ang_x;

  }
}

////////////////////////////////////////////////////////////////////
//////////////////CALCULO DEL PID DE VELOCIDAD//////////////////////
////////////////////////////////////////////////////////////////////

void PID_velocidad(float error_velocidad)
{
  float t_velocidad = millis();
  float tchange_velocidad = t_velocidad - lastTime_velocidad;
  if (tchange_velocidad >= Tmuestreo_velocidad)
  {
    //Calculo de la parte proporcional
    float PID_prop_velocidad = Kp_velocidad * error_velocidad;

    //Cálculo de la parte integral
    float errorSum_velocidad = errorSum_velocidad + error_velocidad *
Tmuestreo_velocidad / 1000; //Calculo del error integral
    float PID_int_velocidad = Ki_velocidad * errorSum_velocidad;
//Calculo del termino integral

    U_control_velocidad = PID_prop_velocidad + PID_int_velocidad;

    lastTime_velocidad = t_velocidad;
  }
}
```

```
////////////////////////////////////////////////////////////////////////
////////////////CALCULO DE LA VELOCIDAD DEL ROBOT//////////////////////
////////////////////////////////////////////////////////////////////////

// Actualiza la velocidad del motor 1
void leerInformacionEncoderM1()
{
  float rpm = 0;
  float Ts_encoder = 0;
  float direc = 0;
  c = c + 1;
  // 10 o 5*2: 10 pulsos se corresponden a una vuelta del eje del
motor
  if (c >= 5) // se puede poner 10, o contar 5 y dividir entre dos
  {
    direc = analogRead(M1_PhaseB);
    Ts_encoder = (millis() - tiempo_prev_encoder) / 1000;
    tiempo_prev_encoder = millis();
    // 34: relacion de la reductora (34 vueltas eje motor = 1 vuelta
eje salida)
    rpm = 60 / (2 * 34 * Ts_encoder);
    // Pasar a velocidad lineal (2*pi/60)*0.625 (radio de la rueda)
    velocidad = rpm * 0.06545;
    // La entrada se usa como comparador digital con respecto al valor
medio del conversor AD
    if (direc < 512)
    {
      velocidad = -velocidad;
    }
    c = 0;
  }
}

////////////////////////////////////////////////////////////////////////
//////////////////////CONTROL DE LOS MOTORES//////////////////////////
////////////////////////////////////////////////////////////////////////

// Controla los motores en función de U_control_posicion
void controlMotores(int velocidadMotor,int giro)
{

  if (velocidadMotor > 255)
  {
    velocidadMotor = 255;
  }
  else if (velocidadMotor < -255)
  {
    velocidadMotor = -255;
  }

  // Según como definas la lógica de giro, irá de una forma u otra
  int velocidadMotor1=velocidadMotor;
  int velocidadMotor2=velocidadMotor;
  if(giro=1)
  {
      velocidadMotor1-=30;
      if(velocidadMotor1<-255)
      {
           velocidadMotor1 = -255;
      }
  }
```

```
  else if (giro==-1)
  {
      velocidadMotor2-=30;
      if(velocidadMotor2<-255)
      {
           velocidadMotor2 = -255;
      }
  }

  if (velocidadMotor > 0)
  {
    analogWrite(Motor1, velocidadMotor1);
    digitalWrite(M1_InA, HIGH);
    digitalWrite(M1_InB, LOW);
    analogWrite(Motor2, velocidadMotor2);
    digitalWrite(M2_InA, HIGH);
    digitalWrite(M2_InB, LOW);
  }
  else if (velocidadMotor < 0)
  {
    analogWrite(Motor1, -velocidadMotor1);
    digitalWrite(M1_InA, LOW);
    digitalWrite(M1_InB, HIGH);
    analogWrite(Motor2, -velocidadMotor2);
    digitalWrite(M2_InA, LOW);
    digitalWrite(M2_InB, HIGH);
  }
  else if (velocidadMotor == 0)
  {
    analogWrite(Motor1, 0);
    analogWrite(Motor2, 0);
  }
}

//Control del LED RGB
void RGB(float ang_x)
{
  if (((ang_x > 0) && (ang_x < 30)) || ((ang_x <= 0) && (ang_x > -
30)))
  {
    digitalWrite(Azul,HIGH);
    digitalWrite(Rojo,LOW);
  }
  else
  {
    digitalWrite(Azul,LOW);
    digitalWrite(Rojo,HIGH);
  }
}
```