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

Diseño e implementación en PCB de un robot auto-balanceado mediante Arduino con módulo inalámbrico

Grado en Ingeniería Eléctrica y Electrónica

Trabajo Fin de Grado

Víctor Esteban Falconi Loja

Ignacio Del Villar Fernández

Pamplona, 15 Junio de 2015

upna Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

RESUMEN

El presente Trabajo Fin de Grado consiste en el desarrollo e implementación de un pequeño robot basado en Arduino, capaz de auto-balancearse, y de este modo mantener el equilibrio sobre sus dos ruedas.

Esto se consigue empleando un sensor que nos informa en tiempo real sobre el ángulo que forma el robot con respecto al suelo. El ángulo obtenido por el sensor es gestionado por un microcontrolador (Arduino), el cual, mediante un algoritmo de control obtiene la señal de control que se transmite a los motores de las ruedas para su accionamiento. Además, se añadirá un módulo de conexión inalámbrica para poder monitorizar variables importantes del robot, o incluso llegar a controlarlo mediante cualquier Smartphone gracias a la tecnología bluetooth, y de esta manera conseguir desplazarlo siguiendo un recorrido deseado.

LISTA DE PALABRAS CLAVE

Robot Segway, Arduino, auto-balanceado, Bluetooth, algoritmo de control PID, giroscopio, acelerómetro.

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1 INTRODUCCIÓN

La robótica es la rama de la tecnología que se dedica al diseño, construcción, operación, disposición estructural, manufactura y aplicación de los robots [1] [2].

El interés personal y motivación hacia la robótica, por ser un área multidisciplinaria que involucra diferentes campos del conocimiento, tales como: electrónica, teoría de control, programación, mecánica, procesamiento de señales; y su gran desarrollo tecnológico en los últimos años, han sido factores fundamentales que han influido en la elección de un Trabajo Fin de Grado de estas características.

Muchos expertos en robótica dirían que es difícil dar una definición de robot universalmente aceptada. Distintas organizaciones y asociaciones internacionales ofrecen definiciones diferentes aunque, obviamente, próximas entre sí. La definición de robot que ofrece la Robot Industries Association (RIA), es: "Manipulador funcional reprogramable, capaz de mover material, piezas, herramientas o dispositivos especializados mediante movimientos variables programados, con el fin de realizar tareas diversas" [3].

Actualmente, la mayoría de robots móviles que existen en el mercado y en fase de investigación utilizan ruedas para su locomoción. Esto es debido en parte por su mayor eficiencia energética en comparación a los robots que usan orugas o patas.
Por otra parte, los robots que utilizan dos ruedas necesitan de menos partes y su construcción presenta una mayor facilidad, además que su control es menos complejo que la actuación de orugas o patas.

Figura 1: Locomoción por ruedas(a); por orugas (b); por patas (c) - fuentes: http://www.damngeeky.com/wp-content/uploads/2013/12/MiP-robot-by-WowWee.jpg; https://sites.google.com/site/irenerobotica/6-robots-moviles

Estos tipos de robots con dos ruedas ofrecen la oportunidad de desarrollar sistemas de control que son capaces de mantener la estabilidad en sistemas que de otro modo serían altamente inestables. Un ejemplo de este tipo de sistemas es el péndulo invertido [4].

1.1 Robots Auto-Balanceados De Dos Ruedas

Pertenece a esta categoría todo robot capaz de equilibrarse en posición vertical sobre sus dos ruedas. Las dos ruedas junto con sus respectivos actuadores, por lo general motores eléctricos, están situadas debajo de la base y permiten que el cuerpo del robot pueda mantener su posición vertical accionando los motores en la dirección de inclinación, ya sea hacia atrás o hacia delante.

Las dos ruedas también proporcionan un sistema de locomoción diferencial, permitiendo que el robot pueda ir recto, girar sobre sí mismo y trazar curvas a través de diversos terrenos y entornos.

Figura 2: Trayectorias locomoción diferencial

Una de las aplicaciones en la que más éxito ha tenido este tipo de robots es en el área de los medios de transportes unipersonales, como lo es el vehículo auto-balanceado SEGWAY® creado en el 2001, por Dean Kamen y que es la principal alternativa comercial en vehículos unipersonales; además, este tipo de transporte, da una solución para la descongestión de tráfico en las grandes ciudades. Se caracteriza por ocupar poco espacio, puede girar sobre sí mismo, es ecológico, sostenible y silencioso, con una buena autonomía gracias a sus motores eléctricos.

2 ESTADO DEL ARTE

A continuación se presentará el estado del arte actual de los robots existentes que se basan en el auto-balanceo mediante dos ruedas y se clasificarán según su aplicación principal.

2.1.1 Robots para servicio

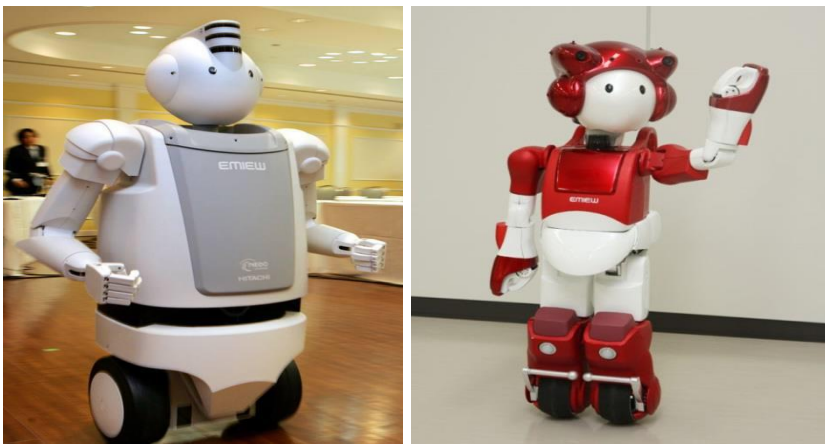


Figura 3: EMIEW 1 (izquierda), EMIEW 2 (derecha) - fuente: http://www.roboticstoday.com/robots/robots-a-to-z/e

EMIEW (Excellent Mobility and Interactive Existence as Workmate) [5] es un robot de dos ruedas auto-balanceable producido por Hitachi. Su versión mejorada es el EMIEW 2 [6] con aproximadamente la mitad de altura y muchísimo más liviano. Es capaz de reconocer la voz humana y entablar pequeñas conversaciones; también es capaz, de reconocer las formas de los objetos, evitar obstáculos utilizando un radar láser para derivar un mapa de su superficie envolvente y además es capaz de movilizarse en superficies irregulares con una velocidad máxima de 6 km/h. Está pensado para su utilización principalmente en viviendas y entornos de oficina.

Características		
Modelo	**EMIEW 1**	**EMIEW 2**
Altura	1,30 m	0,80 m
Peso	70 kg	13 kg
Velocidad	6 km/h	6 km/h
Autonomía	1 h aprox	1,5 h aprox
Año lanzamiento	2005	2007

Tabla 1: Características EMIEW

Figura 4: A.M.P. Bot - fuente: http://www.roboticstoday.com/robots/amp

A.M.P. Bot (Automated Musical Personality) [7] es también otro robot que basa su locomoción en dos ruedas, auto-balanceándose para mantenerse en equilibrio verticalmente gracias a su giroscopio piezoeléctrico. Fue lanzado por Hasbro's Tiger Electronics y Sega Toys.

Es un robot musical que tiene varios modos de funcionamiento. Estos van, desde un modo baile en el que el robot bailará a ritmo de la música que reproduce, a un modo de seguimiento en el que el robot seguirá al usuario reproduciendo música para su entretenimiento. El robot está equipado con sensores infrarrojos para detectar y evitar los objetos, además de sensores touch-pad y un control remoto. Para la reproducción de la música posee tres altavoces que ofrecen una potencia de salida de 12 vatios.

Funciona con seis pilas de tipo D aunque también es posible conectarlo a la red eléctrica. Su precio ronda los 500 dólares.

Características	
Altura	0,74 m
Peso	7 kg
Velocidad	6 km/h
Autonomía	10 h
Año lanzamiento	2008

Tabla 2: Características AMP Bot

2.1.2 Robots para transporte

Figura 5: Segway PT - fuente: https://co2calculator.files.wordpress.com/2008/10/segway.jpg

Segway PT [8] es un vehículo unipersonal de transporte de dos ruedas, con auto-balanceo controlado por ordenador, desarrollado por Dean Kamen en el 2001. Tiene un complejo conjunto sensorial del equilibrio basado en cinco giroscopios y dos sensores de inclinación. Su control de la trayectoria se basa en la dirección de inclinación del manillar. Además, es respetuoso con el medio ambiente gracias a sus motores eléctricos. Se puede adquirir por un precio aproximadamente de 7000 €.
Existen 2 líneas principales de modelos: la línea básica, llamada i2, y la línea todo terreno, llamada x2.

Características		
Modelo	**i2**	**x2**
Tamaño Ruedas	19”	33”
Peso	47 kg	54 kg
Máxima Velocidad	20 km/h	20 km/h
Autonomía	38 km aprox	19 km aprox
Motores	Brushless	Brushless

Tabla 3: Características Vehículos Segway PT

Figura 6: Segway P.U.M.A. - fuente: http://www.segway.com/segway-resources/puma/images/puma.png

Segway P.U.M.A. (Personal Urban Mobility and Accessibility) [9] es un vehículo lanzado conjuntamente por Segway Inc y General Motors en el 2009, con capacidad para transportar a dos pasajeros, este vehículo posee la misma tecnología de estabilización de Segway PT para mantener el equilibrio en dos ruedas paralelas durante la conducción. Además de las dos principales ruedas en cada lado del vehículo, hay ruedas estabilizadoras pequeñas en la parte delantera y trasera para apoyar el vehículo mientras está estacionado y sirven además como limitadores del ángulo máximo de inclinación.

Características	
Peso	140 kg
Velocidad máxima	56 km/h
Autonomía	56 km aprox
Año lanzamiento	2009

Tabla 4: Características Segway P.U.M.A.

2.1.3 Robots para uso didáctico o entretenimiento

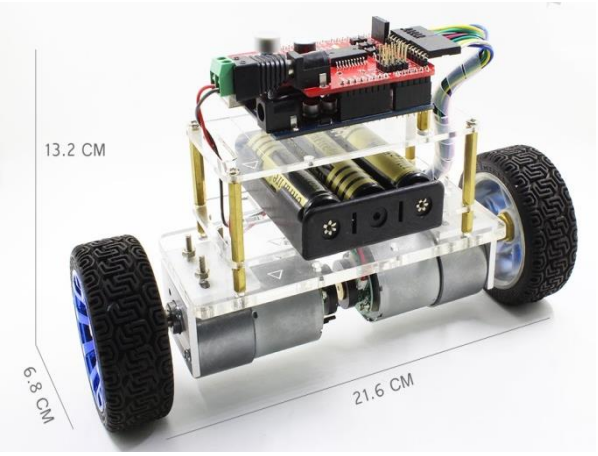


Figura 7: Balanbot Arduino Self-balancing Robot - fuente: https://images.indiegogo.com/file_attachments/962971/files/20141027093118-measure.jpg?1414427478

El Balanbot [10] es un pequeño robot educativo desarrollado por un grupo de ingenieros entusiastas (Steve Chang, Ryan Quin y Bruce Chen), el cual está montado en una estructura de acrílico, utiliza un sensor MPU6050 que combina un giroscopio y un acelerómetro para obtener el ángulo de inclinación y se conecta a la placa Arduino a través de la interfaz I2C. También integra un L298P como actuador de los motores y además permite una comunicación con otros dispositivos mediante tecnología Bluetooth y Wifi. Presenta un soporte para tres pilas, que es de donde se alimenta el robot. Pesa alrededor de medio kilo y se puede adquirir por aproximadamente unos 100 €.

Figura 8: nBot Balancing Robot - fuente: http://www.geology.smu.edu/~dpa-www/robo/nbot/

nBot Balancing Robot [11] es el diseño de un entusiasta llamado David Anderson en el 2003. Este robot utiliza un giroscopio y acelerómetro cuyas salidas se juntan por un filtro Kalman, proporcionando así una entrada precisa para el control de la estabilidad vertical. Una de las buenas propiedades de este robot es ser capaz de desplazarse por terrenos muy abruptos e irregulares.

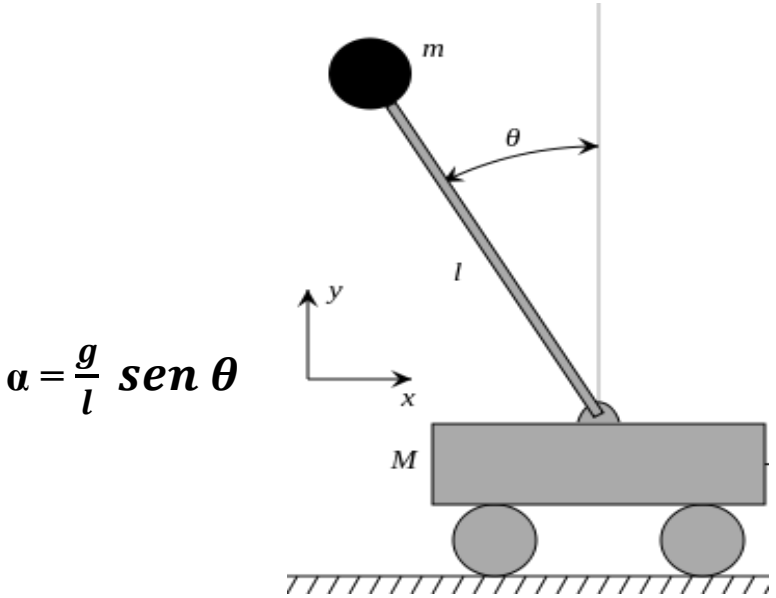
2.1.4 Modelo Teórico del Sistema

El sistema se asemejaría al modelo teórico carro y péndulo (cart and pole) [12] es decir, una variante del sistema de péndulo invertido cuyo centro de masas está por encima de su eje de rotación o pivotamiento.

Un sistema de estas características es por naturaleza inestable y debe ser equilibrado constantemente de manera externa para permanecer en una posición de equilibrio. En este tipo de sistema se utiliza una base móvil, que va desplazándose horizontalmente según convenga para conseguir mantener el equilibrio.

El ejemplo sencillo para entender la dinámica de este sistema es intentar mantener en equilibrio una varilla sujeta en un extremo por nuestro dedo. El péndulo invertido es uno de los problemas clásicos en la dinámica.

Las ecuaciones del movimiento de un péndulo invertido simple, sin considerar ningún tipo de fricción o restricciones al movimiento, y suponiendo el cuerpo como una varilla rígida ideal, con la masa concentrada en su extremo y con su eje de rotación fijo en el espacio, es muy similar a la de un péndulo clásico.



$$\alpha = \frac{g}{l}\ sen\ \theta$$

Figura 9: Esquema modelo teórico carro y péndulo - fuente: http://iimyo.forja.rediris.es/_images/Cart-pendulum.png

Donde **α** es la aceleración angular del péndulo, **g** es la aceleración de la gravedad, **l** es la longitud del péndulo, y **θ** es el ángulo medido desde la posición de equilibrio.

Es fácil comprobar que el péndulo se acelera e inestabiliza más rápido, cuanto más lejos está de la posición de equilibrio y que dicha aceleración es inversamente proporcional a la longitud del péndulo. De aquí se deduce una importante especificación constructiva, los péndulos altos caen más lentamente que los bajos.

Se podría considerar el péndulo como una varilla en donde la masa se reparte uniformemente a lo largo de su longitud, debido a todo esto, al realizar la modelización de este sistema aparecen términos de inercias relacionados con la dinámica de la varilla, y por ello las ecuaciones del sistema se complican demasiado con respecto a la consideración anterior del péndulo como masa puntual y no es objeto de este trabajo un estudio a fondo del modelo del sistema.

2.1.5 Conclusiones

Como podemos observar en los modelos analizados, los conceptos de diseño y funcionamiento de estos robots son muy similares. Normalmente utilizan un giroscopio para medir la inclinación, encoders en las ruedas para medir la posición de la base del robot y un microcontrolador para realizar los cálculos.

Estos componentes se combinan para conseguir la base principal de mantener la estabilidad sobre las dos ruedas. Inclinómetros o acelerómetros son a veces añadidos junto al giroscopio o lo que es lo mismo, utilizan dispositivos IMU (Inertial Measurement Unit) [13], permitiendo así una señal de entrada más precisa y mucho mejor para el sistema de control.

En lo que se refiere a los algoritmos de control, los sistemas de control lineales como un control PD o PID son los más comunes debido a una mayor disponibilidad y desarrollo de la literatura en este campo, además de que su proceso de implementación es menos complicado.

3 OBJETIVOS

El objetivo principal que se perseguirá es desarrollar un robot capaz de mantener el equilibrio auto-balanceándose con sus dos ruedas. Para llegar a conseguir el objetivo principal; se tienen que ir desarrollando otras metas complementarias:

- Implementación de algoritmos de control lineal con microcontroladores.
- Enfrentarse a problemas reales que se presentan a lo largo de la realización de un proyecto e intentar solucionarlos de la mejor forma posible.
- Permitir la monitorización de variables importantes del sistema.
- Permitir el control remoto desde cualquier Smartphone para desplazarse con aceleraciones moderadas.
- Conseguir que el robot tenga una buena estética con un gran acabado.
- Cumplir condiciones importantes en un trabajo como bajo consumo de energía, alta autonomía y reducido coste.

4 JUSTIFICACIÓN DEL TRABAJO

Las aplicaciones de los vehículos robóticos auto-balanceados a dos ruedas se encuentran en una etapa de crecimiento y desarrollo. En los próximos años este tipo de transportes serán cada día más comunes conforme la tecnología avance y los costos de construcción disminuyan.

El rápido crecimiento en la cantidad de automóviles y por consiguiente en la congestión de las vías, sobre todo en las grandes ciudades, son circunstancias necesarias para la búsqueda de nuevas soluciones en el sistema de transporte, por lo tanto, se están dedicando grandes esfuerzos en la investigación y el desarrollo de sistemas de transporte innovadores que sean eficaces, eficientes y amigables con el medio ambiente; estas características encajan con los citados vehículos auto-balanceados que además presentan una gran maniobrabilidad y ocupan menos espacio con una estructura mecánica más simple, compacta y con mayor precisión referente al centro de masas.

Por presentar estas características, estos vehículos se han convertido en una opción importante entre los investigadores y desarrolladores, que en un futuro harán posible contar con pequeños y ágiles vehículos eléctricos que se desplazarán de manera autónoma, estos estarán conectados entre sí, transmitiéndose constantemente información en una especie de red inalámbrica, y sabrán dónde otros objetos móviles se encuentran evitando coincidir en las mismas rutas.

La principal justificación de este Trabajo Fin de Grado es desarrollar un vehículo robótico auto-balanceado, que serviría como base de partida para un proyecto más grande, el cual desarrollaría un vehículo de transporte inteligente con las características ya antes mencionadas. Este robot podría ser un primer contacto experimental que ayudará a entender mejor el funcionamiento de este tipo de vehículos robóticos.

5 METODOLOGÍA EMPLEADA

5.1 Descripción Del Trabajo

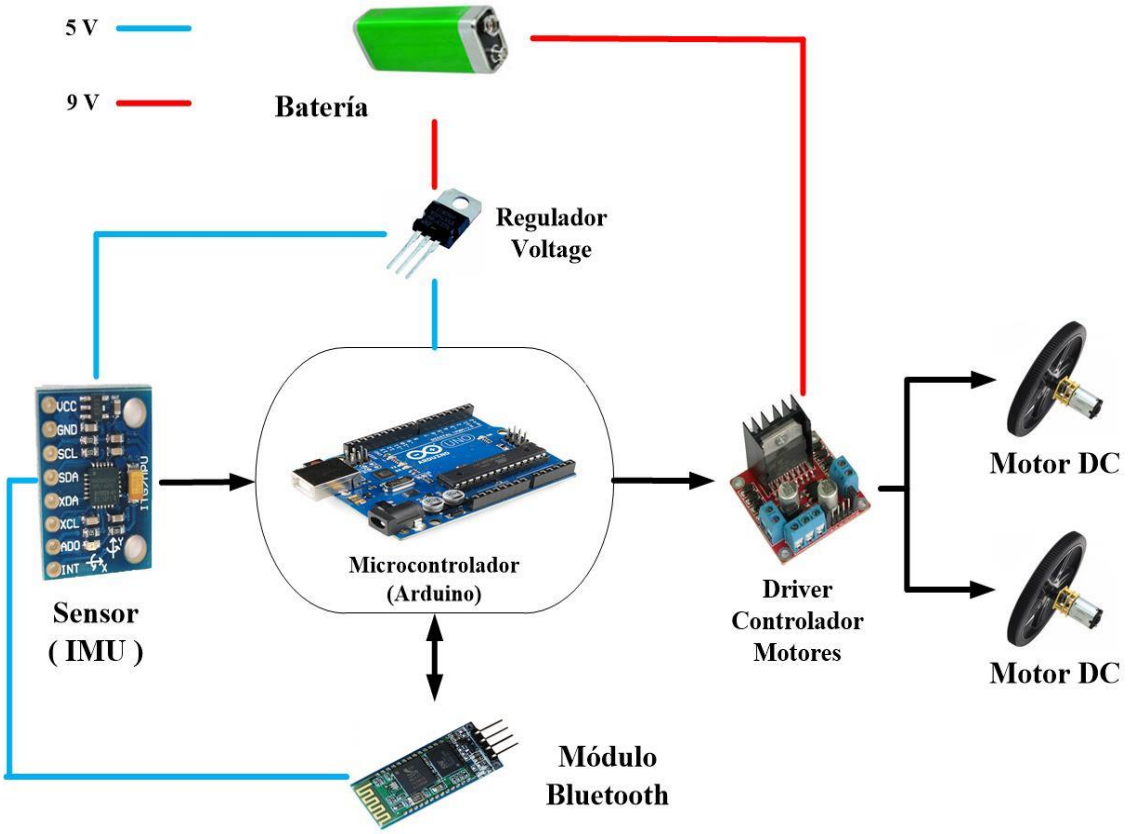


Figura 10: Esquema descriptivo del sistema

Como se aprecia en la Figura 10, el sistema va a estar compuesto básicamente por tres etapas:

1. **Etapa de Sensado:** con ayuda de sensores adecuados, se obtienen información acerca de la posición y orientación del robot con respecto al suelo.

2. **Etapa de Control:** un microcontrolador será el encargado de recibir y gestionar adecuando correctamente la información obtenida del sensado. Este micro también se encargará de generar la señal de control mediante un algoritmo de control PID.

3. **Etapa de Potencia:** recibe la señal de control generada y, con la ayuda de un driver controlador de motores, se transmitirá a los mismos la consigna de actuación adecuada y de este modo se conseguirá mantener el equilibrio.

El sistema robótico dispondrá de la suficiente autonomía gracias a que todas las citadas etapas estarán alimentadas por una batería recargable de 9V para los circuitos de potencia que, pasando por un regulador, se convertirá en 5V, que es el voltaje necesario para la tarjeta Arduino, el módulo sensor y el módulo Bluetooth.

Se diseñarán y fabricarán dos PCBs con ayuda del software "Designspark", utilizado durante el Grado. Una de las PCBs actuará como driver controlador de los motores perteneciente a la etapa de potencia del sistema y en la otra se situará el microcontrolador, así como la etapa de sensado y módulo bluetooth.

También se diseñará y se realizará el montaje de la estructura mecánica que va a albergar los distintos componentes del citado robot. Con ayuda del software "SolidWorks" utilizado durante el Grado, se diseñará y obtendrá los planos de la citada estructura.

A continuación, se realizará una explicación de los elementos fundamentales de los diferentes módulos de cada etapa que forma nuestro sistema a controlar y de las causas que han llevado a su elección.

5.2 Etapa De Sensado

Esta etapa proporciona al robot la capacidad de interpretar el entorno que le rodea. Estará compuesta por los dispositivos de instrumentación necesarios con su respectiva señal de salida correctamente acondicionada. Estos dispositivos, más conocidos como sensores, permiten conocer las variables del sistema a controlar y son uno de los elementos fundamentales de cualquier sistema de control en lazo cerrado, permitiendo realizar la realimentación.

En este caso las variables del sistema a controlar son ángulos de inclinación. El robot debe saber en todo momento cuál es su ángulo de inclinación con respecto a su posición vertical y enviárselo al microcontrolador en tiempo real para ejecutar la acción de control que se ha programado previamente y de esta manera conseguir el control de la planta.

La estimación del ángulo de inclinación presenta cierta complejidad en ciertas aplicaciones donde se necesita medir ángulos de inclinación de manera inercial, es decir, cuando no es posible tener una referencia externa para determinar la posición y orientación en el espacio. Para ello se utilizan los dispositivos ya citados, IMU [14].

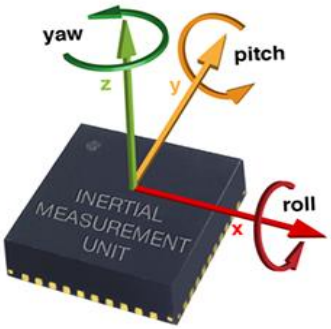


Figura 11: Ángulos de inclinación en un IMU, cabeceo (pitch), alabeo (roll), y guiñada (yaw) - fuente: http://www.rerace.com/img/device/imu.png

5.2.1 IMU (Inertial Measurement Unit)

Es un dispositivo electrónico capaz de medir e informar acerca de la velocidad, orientación y fuerzas gravitacionales de un cuerpo, usando una combinación de acelerómetros y giroscopios. Una IMU funciona detectando la actual tasa de aceleración usando uno o más acelerómetros, y detecta los cambios en atributos rotacionales tales como cabeceo (pitch), alabeo (roll), y guiñada (yaw) usando uno o más giróscopos. Por lo tanto, una IMU no mide ángulos, o por lo menos no directamente, sino que requiere de algunos cálculos para obtenerlos, cálculos realizados por un procesador que también forma parte de la estructura de un IMU. Tanto los acelerómetros, giroscopios y procesador pueden encontrarse todos implementados en un mismo encapsulado.

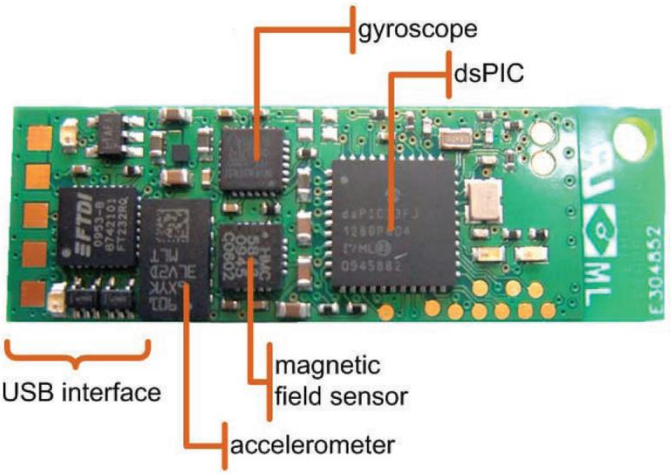


Figura 12: IMU (Inertial Measurement Unit) -fuente: http://en.wikipedia.org/wiki/Inertial_measurement_unit#/media/File:ETHOS_pcb.png

Existen IMUs de 6 grados de libertad, esto significa que lleva tres acelerómetros y tres giroscopios, uno de cada por cada eje ortogonal. Hay también IMUs de 9 grados de libertad, en este caso también llevan un magnetómetro. Otras pueden tener 5 grados, en cuyo caso el giroscopio sólo mide dos ejes, etc.

5.2.2 Acelerómetro

Los acelerómetros son dispositivos electromecánicos que detectan las fuerzas de aceleración, ya sea estática o dinámica. Las fuerzas estáticas incluyen la gravedad, mientras que las fuerzas dinámicas pueden incluir vibraciones y movimiento. Estos dispositivos pueden medir la aceleración en uno, dos o tres ejes [15].

Existen varias tipos de acelerómetros basados en distintas tecnologías (piezo-eléctricos, piezo-resistivos, mecánicos, capacitivos, térmicos, micro-electromecánicos) y varios diseños que aunque todos tienen el mismo fin, pueden ser muy distintos unos de otros según la aplicación a la cual van destinados y las condiciones en las que han de trabajar.

Los acelerómetros empleados por lo general en los dispositivos IMU, son los MEMS [16] (sistemas micro-electromecánicos) por su rapidez y estabilidad, pequeñísimo tamaño, cada vez mayor desarrollo y menor precio. Por otro lado, tienen como inconveniente que la señal obtenida es altamente ruidosa que se podría solucionar con ayuda de buenos filtros.

Se distinguen tres categorías principales de acelerómetros de MEMS: el capacitivo de silicio, el piezo-resistivo y, finalmente, los acelerómetros térmicos. Hasta el momento, los acelerómetros capacitivos de silicio dominaban ampliamente el mercado. Estos acelerómetros contienen placas capacitivas internamente. Algunos de estos son fijos, mientras que otros están unidos a resortes minúsculos que se mueven internamente conforme las fuerzas de aceleración actúan sobre el sensor. Como estas placas se mueven en relación el uno al otro, la capacitancia entre ellos cambia. A partir de estos cambios en la capacitancia, la aceleración se puede determinar [17].

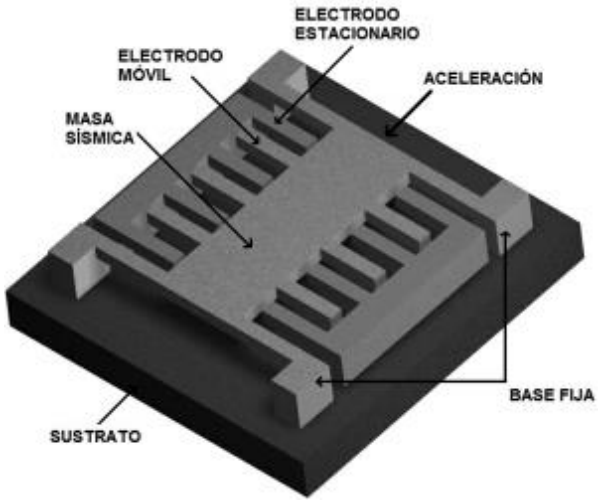


Figura 13: Acelerómetro capacitivo basado en MEMS - fuente: https://www.uv.mx/cienciahombre/revistae/vol22num2/articulos/microace/index.html

5.2.3 Giroscopio

El giroscopio electrónico es un dispositivo que proporciona información acerca de la velocidad angular de un cuerpo con respecto a un sistema de referencia estático. Al igual que el acelerómetro, estos dispositivos pueden medir velocidades angulares en uno, dos o tres ejes.

En un principio, los giroscopios eléctricos eran unos voluminosos artefactos con un altísimo coste empleados mayoritariamente para la tecnología militar. Por suerte, gracias a la revolución digital y la miniaturización de circuitos, hoy en día son de tamaños muy pequeños y se pueden adquirir con relativa facilidad y a un bajo coste.

El principio físico por el que se fundamentan estos tipos de giroscopios es que un objeto vibrante tiende a continuar vibrando en el mismo plano que gira su apoyo. Este tipo de dispositivos se conocen como "giroscopios de vibración de Coriolis", el término de Coriolis, viene porque la velocidad angular se obtiene de las ecuaciones del efecto Coriolis. Hay distintos tipos de giroscopios basados en este efecto entre los que se destacan: piezo-eléctricos, de resonador hemisférico, de rueda vibrante, micro-electromecánicos [18].

Los giroscopios son en general utilizados en objetos que no rotan muy rápido, es decir, tienen buena precisión cuando no se producen cambios bruscos de velocidad angular. Una gran desventaja de estos dispositivos es que debido a la necesidad de integrar la señal del giroscopio, pueden obtenerse derivas en las mediciones e ir sumando un error en cada medida.

Al igual que los acelerómetros, los giroscopios electrónicos empleados por lo general en los dispositivos IMU, son los MEMS por sus características ya comentadas.

Su funcionamiento básicamente se fundamenta en que cuando el giroscopio es girado, una pequeña masa se desplaza a medida que cambia la velocidad angular. Este movimiento se convierte en señales eléctricas de muy baja corriente que pueden ser amplificadas para convertirlas en señales medibles [19].

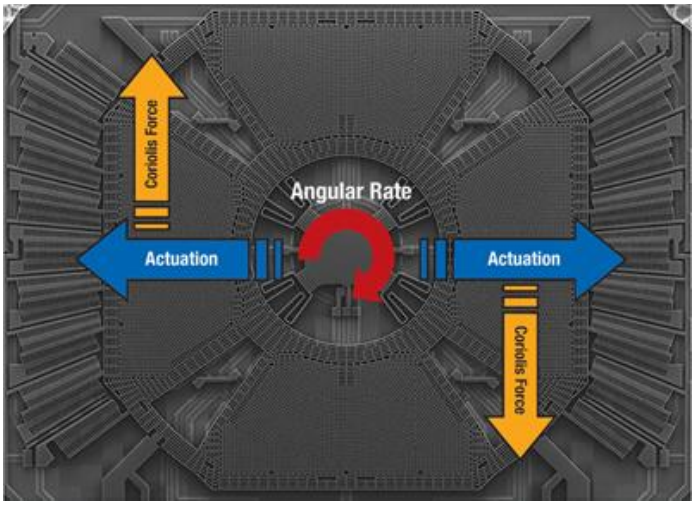


Figura 14: Vista operativa interna de un sensor giroscópico MEMS – fuente: http://www.digikey.com/Web%20Export/techzone/sensor/sensors-article-110411-fig1.jpg

5.2.4 Conclusiones

El dispositivo sensor que proporcionará el ángulo de inclinación va a ser un IMU de 6 grados de libertad, ya que son los más ampliamente distribuidos en el mercado actual y por consiguiente su precio será el más económico. Además, para la aplicación que se va a utilizar con 6 grados de libertad será suficiente.

Se analizaron los dos dispositivos IMU de 6 grados de libertad más populares del mercado: el GY-521 (MPU6050) [20] y el ITG3200/ADXL345 [21].

Características		
Modelo	GY-521 (MPU6050)	ITG3200/ADXL345
Alimentación	3.3 - 5VDC	3.3 VDC
Consumo	3.6 mA aprox.	6.5 mA aprox.
Rango Acelerómetro	2g – 16g	2g – 16g
Rango Giroscopio	±250 - ±2000°/s	±2000°/s
Dimensiones	21 mm x 16 mm	16.65 x 15.60 mm
Protocolo Comunicación	I2C	I2C
Coste	7.16 €	36.63 €

Tabla 5 : Comparativa IMUs de 6 grados de libertad

Finalmente se optó por el GY-521 (MPU6050) que fue elegido por su calidad precio, ya que claramente hay una considerable diferencia de precio y ambos tienen la misma calidad. Además este IMU incluye un procesador propio que puede descargar de trabajo al procesador principal en caso de ser necesario.

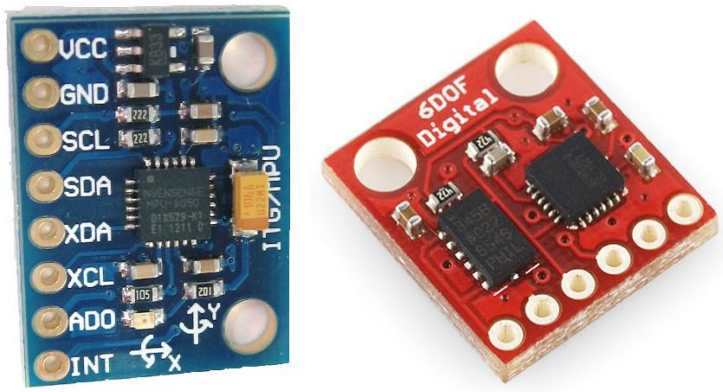


Figura 15: Izquierda (GY-521); Derecha (ITG3200/ADXL345) - fuentes: http://playground.arduino.cc/uploads/Main/mpu-6050.jpg

5.3 Etapa De Control

Esta etapa permitirá al robot tratar toda la información que le llega por parte de la etapa de sensado en tiempo real. Mediante la programación, que previamente se ha implementado, se decidirá qué operación realizar con dicha información. Está compuesta por un microcontrolador junto con su correspondiente programación y un sistema de control en lazo cerrado ayudándose del correspondiente algoritmo de control. En este caso se desea hacer un control para mantener la posición vertical del robot a dos ruedas.

5.3.1 Microcontrolador

Un microcontrolador es un circuito integrado programable, capaz de ejecutar las órdenes guardadas en su memoria. Está compuesto de todos los bloques funcionales de un sistema microprocesador en un único encapsulado, los cuales cumplen una tarea específica. Un microcontrolador incluye en su interior las tres principales unidades funcionales de una computadora: unidad central de procesamiento, memoria y periféricos de entrada/salida; los tres se encuentran conectados mediante buses [22].

Los microcontroladores por su reducido tamaño y costo, además del hecho de que son los dispositivos semiconductores más abundantes de todos en la actualidad, permiten la fácil implantación de sistemas con cierta "inteligencia" distribuida a lo largo de sistemas más complejos [23].

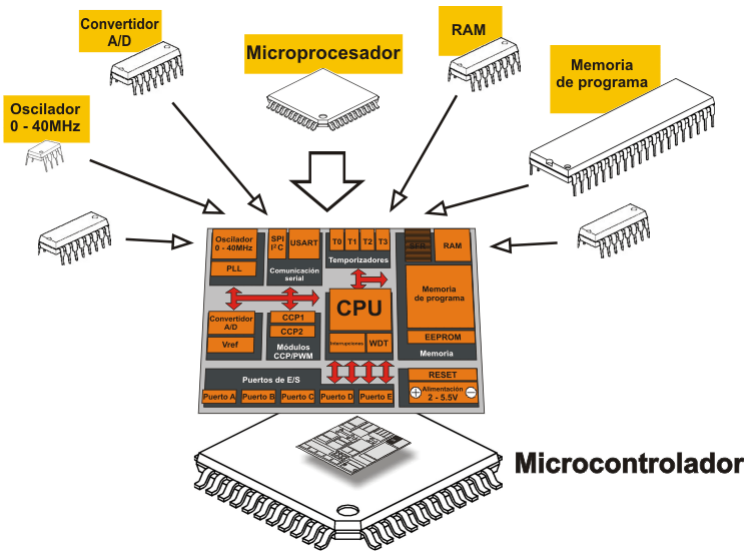


Figura 16: Esquema de un microcontrolador - fuente: www.mikroe.com/img/publication/spa/pic-books/programming-in-c/chapter/01/fig0-1.gif

Microchip Technology, denominada comúnmente Microchip es una de las empresas líderes en la fabricación de microcontroladores, debido a sus bajos costos, desempeño eficiente y rapidez, herramientas de desarrollo software y hardware abundantes y de bajo coste, además de disponer de gran cantidad de documentación.

Atmel, otra empresa líder en este campo es famosa por crear los microcontroladores sobre los que se basan los Arduinos (Atmega) y por tener las mismas características antes mencionadas sobre los micros de Micrichip.

Existen otras alternativas, ofrecidas por empresas como Texas Instruments, Freescale, entre otras. Sus productos pueden ser encontrados en tiendas de electrónica, aunque no siempre se consiguen fácilmente y se dispone de escasa documentación respecto a ellas y escaso desarrollo.

Se realiza un análisis con respecto a los dos fabricantes líderes mediante una comparativa en lo respectivo al hardware, y se puede apreciar que ambos fabricantes presentan características hardware muy similares, tales como:

- Ambos disponen de una unidad central de procesamiento o CPU.
- Presentan memoria tanto de programa como de datos. Se incluye memoria RAM, EPROM, EEPROM y/o FLASH.
- Tienen generador de reloj para el funcionamiento y sincronía de todo el micro.
- Periféricos de E/S tales como:
    - Puertos de Entrada/Salida.
    - Líneas de interrupción externa.
    - Conversores A/D y D/A.
    - Generadores de base de tiempo.
    - Comunicación serial mediante protocolos UART, SPI, I2C, USB, CAN, LIN, etc.
    - Temporizadores de propósito específico, captura de eventos y generadores de señal PWM.

Al estar más o menos empatados en lo que se refiere a hardware, hay que considerar una característica muy a tener en cuenta por parte de los microcontroladores de Atmel, su posibilidad de trabajar con la plataforma Arduino que abre un universo de posibilidades.

5.3.2 Arduino

Arduino es una plataforma de electrónica abierta y libre para la creación de prototipos basada en software y hardware flexibles y fáciles de usar. Se creó como alternativa a las costosas placas de desarrollo existentes hasta su aparición [24].

Esta plataforma ha ido ganado gran parte del mercado y posibilitado a que desarrolladores de software, así como aficionados o apasionados por la electrónica y la tecnología tengan una plataforma de desarrollo simple y potente sin la necesidad de ser expertos en electrónica. Se les da a los programadores la posibilidad de trabajar en sus programas sin profundizar demasiado en la electrónica y dedicarse solo a lo que es de su dominio, que es la programación. Este nuevo universo Arduino, dejó a los desarrolladores con microcontroladores una nueva herramienta para proyectos multidisciplinares, y a los fabricantes un nuevo mercado.

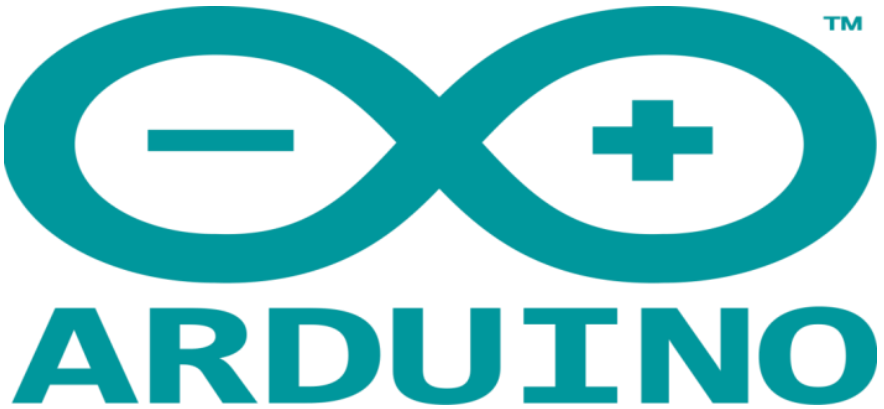


Figura 17: : Logo Arduino característico - fuente: http://www.arduino.cc/

Las placas Arduino pueden tomar información del entorno a través de sus pines de entrada, tanto analógicos como digitales, por parte de toda una gama de sensores y puede afectar aquello que le rodea controlando luces, motores y otros actuadores.

Los microcontroladores más usados por Arduino son el Atmega168, Atmega328, Atmega1280, y Atmega8 por su sencillez y bajo coste que permiten el desarrollo de múltiples diseños.

- **Algunas ventajas sobre la plataforma Arduino:**

El llamado IDE (Integrated Development Environment) o software de desarrollo Arduino está publicado bajo una licencia libre, es decir totalmente gratis, y preparado para ser ampliado por programadores experimentados. Además se puede descargar de su página oficial que ya incluye los drivers de todas las placas disponibles, lo que hace más fácil la carga de códigos desde cualquier ordenador.

Este software permite realizar la edición de nuestros programas fuente, la compilación, la programación del microcontrolador, la simulación y emulación o debugger en tiempo real. Arduino tiene la ventaja que no necesita ningún tipo de tarjeta de programación como pasa con los microcontroladores sino que la misma placa se conecta vía serial al ordenador usando un cable USB y se pueden cargar los programas totalmente en vivo, sin riesgo de dañar la tarjeta debido a su protección adicional. El microcontrolador en la placa Arduino se programa mediante el lenguaje de programación Arduino (basasdo en Wiring) y el entorno de desarrollo Arduino (basado en Processing).

Al ser Arduino una plataforma de código y hardware abierto, se puede acceder a todo aspecto del funcionamiento con respecto a los circuitos y algoritmo de las placas y, mucho mejor que eso, hay total disponibilidad de los archivos Eagle, diagramas y componentes para que cualquiera pueda crear su propia versión de Arduino. Esto hace que actualmente se puedan diseñar Arduinos basados en diferentes tipos de microcontroladores y tener la gran ventaja de no estar impuestos a un solo tipo de micro, por ejemplo con Pics existe el desarrollo de Pinguino, que implementan el hardware de Arduino con los PIC 18Fxx [25].

Algo muy bueno que tiene Arduino, además de sus numerosas ventajas, es que se dispone de muchísima documentación tanto por libros como por Internet. Desde su propia página web, el IDE que también viene con multitud de ejemplos y los incontables proyectos y tutoriales difundidos en la red sobre esta plataforma la hacen una de las más fáciles de desarrollar.

Arduino también ofrece una gran gama de Shields o placas extras que cumplen funcionalidades específicas como Ethernet, GSM, Control de Reles, Wi-fi y pueden ser acopladas a las placas de forma sencilla, aumentando considerablemente el rango de aplicaciones disponibles. Esta plataforma incluso tiene compatibilidad con infinidad de

periféricos de otras marcas como Xbee, teclados, LCD, sensores digitales, dispositivos de Sparkfun, serial, 1-Wire, SD-Card entre muchos otros [26].

Gracias a su versatilidad, Arduino se ha convertido en la placa de desarrollo con la que prácticamente se puede hacer de todo, desde domotizar un hogar u oficina, aplicaciones en robótica como cerebro de un robot o ser utilizado como nodos de tecnologías WSN (Redes de sensores inalámbricos). Entre sus aplicaciones más conocidas están:

- Control y monitoreo de sensores
- Efectos con leds
- Transponders (Transmisores/receptores)
- Educación
- Monitoreo ambiental
- Biomedicina
- Telemedicina
- Domótica
- Robótica

Si en algo se diferencia Arduino de otras plataformas de desarrollo, es en la multitud de placas con distintas prestaciones que ofrecen al mercado. Dependiendo de las necesidades del usuario se optarán por placas diferentes que poseen distintos atributos. A continuación se presenta algunas de estas placas:

- **Arduino Uno R3**: la placa básica que ofrece Arduino. Está basada en un microcontrolador ATmega328. Posee 14 I/O digitales (6 de ellos soportan PWM), 6 entradas analógicas, conexión USB y opera a 5 V [27].

Figura 18: Arduino Uno R3 - fuente: http://www.arduino.cc/en/Main/ArduinoBoardUno

- **Arduino Due**: se presenta como una de las placas más robustas al ser la primera con un microcontrolador basado en un núcleo ARM de 32-bit. Posee 54 entradas/salidas digitales (12 PWM), 12 entradas analógicas, una conexión USB OTG, y sólo funciona con alimentación de 3.3V, a diferencia de las otras placas si se conecta a voltajes de 5V podría dañar la placa [28].

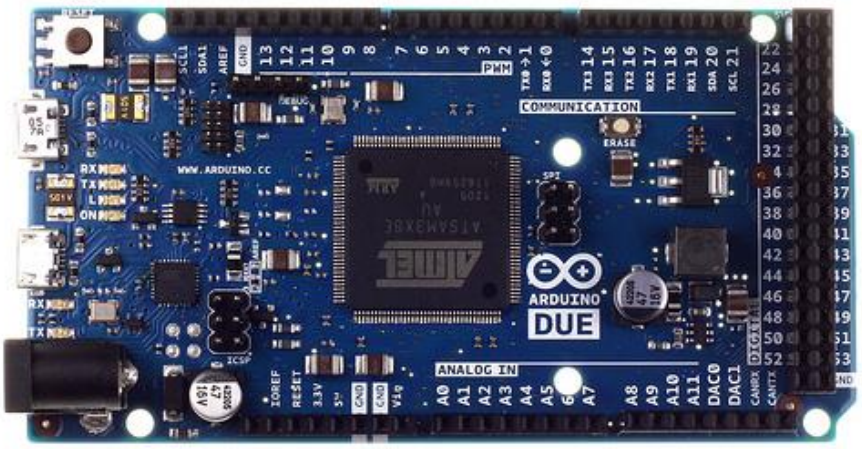


Figura 19: Arduino Due - fuente: http://www.arduino.cc/en/Main/ArduinoBoardDue

- **Arduino Yún**: Se distingue de las demás placas por su doble procesador, de parte de Arduino un ATmega32u4 y un Atheros AR9331 que soporta distribuciones Linux basados en OpenWRT. La placa provee de soporte Ethernet, Wifi, USB y Micro SD, 20 I/O terminales digitales (7 PWM) 12 entradas analógicas [29].

Figura 20: Arduino Yún - fuente: http://www.arduino.cc/en/Main/ArduinoBoardYun

- **Arduino Mega ADK**: placa muy robusta, basada en un microcontrolador ATmega2560 con una interfaz USB que permite conectar con teléfonos Android directamente. Posee 54 pines digitales (15 se pueden utilizar para PWM), 16 entradas Analógicas [30].

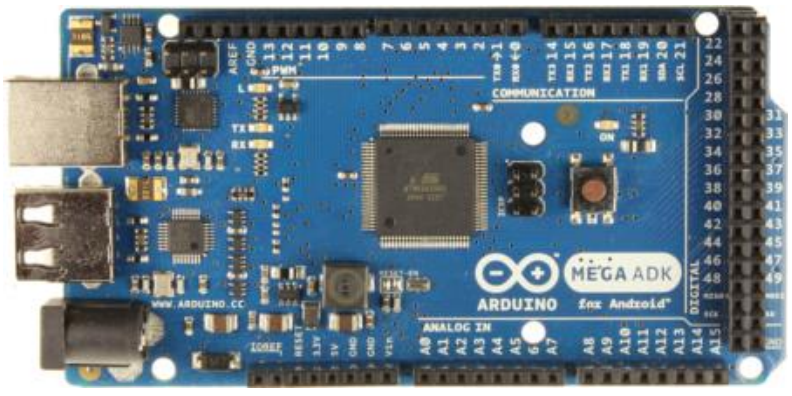


Figura 21: Arduino Mega ADK - fuente: http://www.arduino.cc/en/Main/ArduinoBoardMegaADK

- **Arduino Nano y Mini**: estas placas están pensadas para una conexión fácil de prototipado y para proyectos de robótica que requieran muy poco espacio; tienen el mismo procesador que la placa UNO pero se presentan compactas con muchos menos puertos [31] [32].

Figura 22: Arduino Nano - fuente: http://www.arduino.cc/en/Main/ArduinoBoardNano

5.3.3 Elección de la placa

Tras el profundo análisis y las numerosas ventajas citadas por parte de la plataforma Arduino, está claro que la elección que se toma será una placa Arduino con un microcontrolador Atmel, pero de entre toda la variedad de placas, hay que decantarse por la que se adapte al trabajo y a las necesidades existentes de la mejor manera posible. A continuación se presenta una tabla comparativa con las opciones analizadas:

Modelo	Uno R3	Due	Yún	Mega	Nano
Micro	**ATmega328**	AT91SAM3X8E	ATmega32u4	ATmega2560	**ATmega328**
Voltaje	**5 V**	3,3 V	5 V	5 V	**5 V**
Velocidad CPU	**16 MHz**	84 MHz	16 MHz	16 MHz	**16 MHz**
Digital E/S	**14**	54	20	54	**14**
Entradas Analógicas	**6**	12	12	16	**8**
PWM	**6**	12	7	15	**6**
USB	**Regular**	2 Micro	Micro	Regular	**Mini-B**
I2C	**Si**	Si	Si	Si	**Si**
Precio	**17,00 €**	36,00 €	52,00 €	39,00 €	**18,00 €**

Tabla 6: Comparativa placas Arduino

Ya que se ha decidido usar un IMU que presenta un protocolo de comunicación I2C, queda impuesto que la placa a elegir debe contener un microcontrolador capaz de soportar este protocolo. Se precisa una placa con las suficientes salidas y entradas, tanto analógicas como digitales, para satisfacer todas las aplicaciones que se van a utilizar, aunque tampoco se pretende que sobren demasiadas. Por otro lado, un factor muy importante también a tener en cuenta es el precio de la placa.

Considerando que nuestra placa debe tener las características mencionadas, se descartan tanto Arduino Due como el Arduino Mega ya que son placas que disponen de muchísimos pines de entradas y salidas y tienen un precio más o menos elevado en comparativa. El Arduino Yún también queda descartado debido a que dispone de muchas aplicaciones adicionales que no se precisan ni serán utilizadas. Estas

aplicaciones adicionales hace que su precio se dispare y resulte la opción más cara de todas. Tanto la placa Arduino Uno como Arduino Nano presentan características muy similares y son las opciones a tener en cuenta por ser las dos placas que mejor se adaptan a nuestras necesidades.

Se elige la placa Arduino Uno porque era de la que se tenía disponibilidad inmediata en el laboratorio, y al ser la placa principal de Arduino es de la que más documentación y conexionados existe. Además, si se precisa la adición de Shields o placas extras, a la Arduino Uno se pueden añadir con mayor facilidad por su estructura.

5.3.4 Sistema de control

Se entiende como sistema de control a la combinación de componentes que actúan juntos para influir en el funcionamiento del otro sistema a controlar. Su finalidad es conseguir, mediante la manipulación de las variables de control, un dominio sobre las variables de salida, de modo que estas alcancen unos valores prefijados (consigna) [33].

Los elementos básicos que forman parte de un sistema de control y permiten su manipulación son los siguientes:

- Sensores: permiten conocer los valores de las variables medidas del sistema.
- Controlador: utilizando los valores determinados por los sensores y la consigna impuesta, calcula la acción que debe aplicarse para modificar las variables de control en base a cierta estrategia.
- Actuador: es el mecanismo que ejecuta la acción calculada por el controlador y que modifica las variables de control.

Existen dos clases comunes de sistemas de control:

- Sistema de lazo abierto: en el que la salida no tiene efecto sobre la acción de control. Más sencillo, pero por lo general se comporta peor.

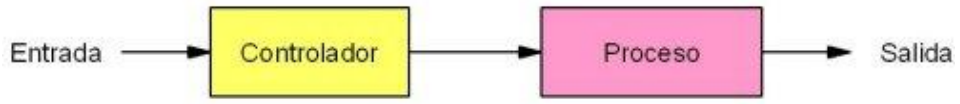


Figura 23: Sistema de control en lazo abierto - fuente: http://www.portaleso.com/usuarios/Toni/web_robot_3/imagenes/lazo_abierto.jpg

- Sistema de lazo cerrado: en el que la salida ejerce un efecto directo sobre la acción de control. Más complejo, pero por lo general se comporta mejor.

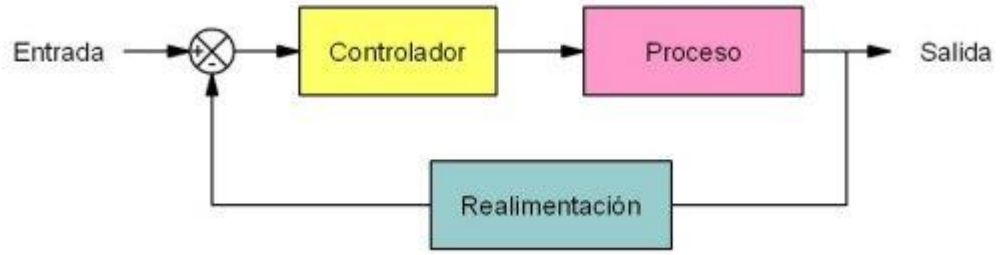


Figura 24: Sistema de control en lazo cerrado - fuente: http://www.portaleso.com/usuarios/Toni/web_robot_3/imagenes/lazo_cerrado.jpg

Como se ha mencionado antes en el modelo teórico, este sistema es por naturaleza altamente inestable y debe ser equilibrado constantemente. Por lo tanto, se necesitará un control en lazo cerrado, es decir, aquel en donde se tiene en cuenta la señal de salida mediante una realimentación.

Los sistemas de control en lazo cerrado obtienen su acción de control en función de una entrada de referencia o consigna. Primero se compara la consigna con la señal de salida del sistema, que en este caso se recoge por los sensores (IMU), obteniendo el error. En función de esa señal de error, el controlador (placa Arduino) elabora la acción de control que se envía a los actuadores (motores DC), todo esto con el fin de obtener la salida deseada, en este caso lograr el ángulo 0 de inclinación o, lo que es lo mismo, lograr la posición vertical en equilibrio del robot de dos ruedas.

Se optará por un controlador PID, ya que como se ha comentado antes en las conclusiones del estado del arte, son los más comunes debido a una mayor disponibilidad y desarrollo de su documentación, y además se han conseguido buenos resultados de estabilidad con el uso de estos controladores.

Controlador proporcional integral derivativo (PID): el algoritmo de cálculo del control PID viene dado por tres componentes distintos: parte **P**roporcional, acción **I**ntegral, y acción **D**erivativa. Se pretenderá lograr que el bucle de control corrija eficazmente y en el mínimo tiempo posible los efectos de las perturbaciones[34].

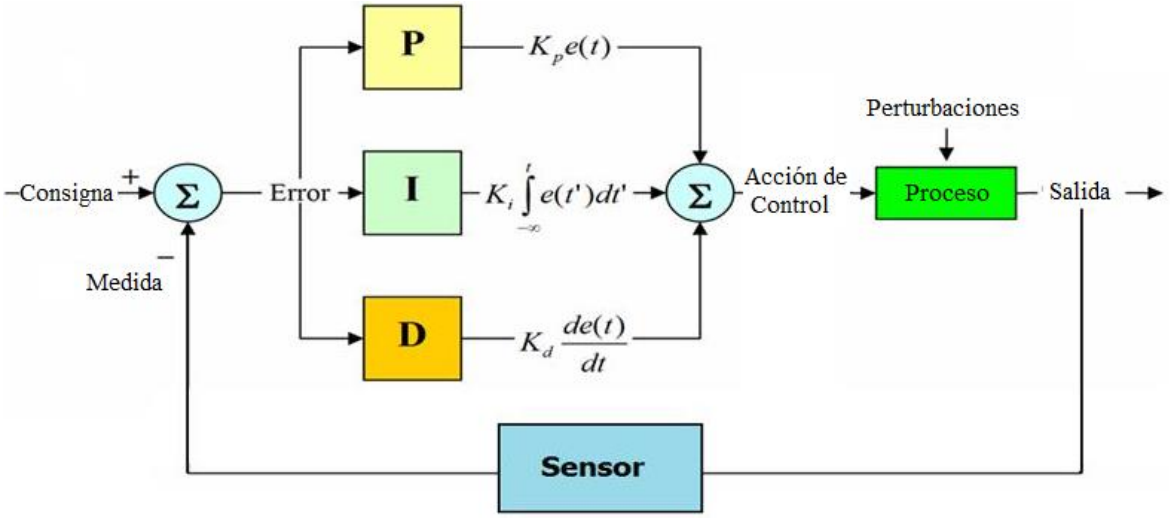


Figura 25: Diagrama de bloques de un sistema de control con controlador PID - fuente: http://www.e-ucenje-kel.ftn.uns.ac.rs/eSite/images/stories/General_descriptions/PID/7.jpg

- La parte proporcional (P) consiste en el producto entre la señal de error y la constante proporcional (Kp) para lograr que el error en estado estacionario se aproxime a cero, pero la parte proporcional no considera el tiempo, sino que solo determina la reacción del error actual; por lo tanto, la mejor manera de solucionar el error permanente y hacer que el sistema contenga alguna componente que tenga en cuenta la variación respecto al tiempo, es incluyendo y configurando las acciones integral y derivativa.

- La acción integral (I) tiene como propósito disminuir y eliminar el error en estado estacionario, provocado por la parte proporcional. La acción integral tiene en cuenta la historia pasada del error y se anula cuando se hace cero. Esta acción se presenta cuando hay cierto error. Este error es integrado y es multiplicado por la constante de integración (Ki). La acción integral es añadida a la parte proporcional para formar el control PI con el propósito de obtener una respuesta estable del sistema sin error estacionario.
- La acción derivativa (D) pretende mantener el error al mínimo, corrigiéndolo proporcionalmente con la misma velocidad que se produce. De esta manera evita que el error se incremente. La acción derivativa predice los cambios en el error y se anula cuando alcanza un valor estacionario. Esta acción se manifiesta cuando hay un error no constante. Este error se deriva con respecto al tiempo y se multiplica por la constante derivativa (Kd). La acción derivativa se suma a las otras dos acciones anteriores, la proporcional y la integral.

De una manera empírica, o empleando el algoritmo de Ziegler Nichols, se consigue ajustar las tres constantes de control, Kp, Ki, y Kd en el algoritmo del PID. Así el controlador puede proveer una señal de control adecuada para el proceso a realizar. Hay que destacar que por el simple hecho de usar un PID, no se garantiza un control óptimo del sistema o la estabilidad del mismo. Algunas aplicaciones pueden solo requerir de uno o dos modos de los que provee este sistema de control. Un controlador PID puede ser llamado también PI, PD, P o I en la ausencia de las acciones de control respectivas.

$$\boldsymbol{U\,(t) = Kp \cdot e(t) + Ki \int_0^t e(t) \cdot dt + Kd \cdot \frac{de(t)}{dt}}$$

Figura 26: Ecuación de la acción de control por parte del PID

El ajuste de las constantes de control Kp, Ki y Kd se podrá realizar desde el propio código del algoritmo de control que se va a implementar en el Arduino, pero debido a que es un poco incómodo y se pierde mucho tiempo haciéndolo de esta forma, también se incorpora al robot una forma de ajustar estas constantes en tiempo real de forma manual mediante unos potenciómetros.

Figura 27: Potenciómetro a utilizar – fuente: http://www.orbelec.es/images/tienda/pt10lv.jpg

5.4 Etapa De Potencia

En esta etapa se pretende dar al robot la fuerza externa o par que necesita para poder mantener el equilibrio sobre sus dos ruedas en posición vertical, y esto se consigue con la utilización de motores. Para aplicaciones de este tipo, como por ejemplo en la robótica, suelen ser motores eléctricos de pequeño tamaño y estos precisarán ser accionados por un driver motor adecuado. La acción de control obtenida en esta etapa será la encargada de decir cuándo y con qué fuerza tienen que accionarse los motores localizados en las ruedas para mantener el equilibrio. Por tanto, la etapa la forman los motores y su correspondiente driver.

5.4.1 Motor eléctrico para robótica

Este motor es una máquina eléctrica que convierte la energía eléctrica en mecánica, provocando un movimiento rotatorio, gracias a la acción de un campo magnético. Los motores eléctricos para aplicaciones de robótica suelen ser de tres tipos: [35].

- Motores DC o motores de corriente continua: pueden funcionar libremente en ambas direcciones de giro invirtiendo la polaridad de sus bornes. Es muy fácil controlar su velocidad y par mediante la corriente que se le suministra por sus dos bornes o mediante PWM, pero no su posición. Los motores DC tienen dos grandes inconvenientes (giran demasiado rápido) y, sobre todo, tienen muy poca fuerza o par. Entonces, lo que se hace es bajar la velocidad de giro y aumentarle la fuerza o par. Eso se consigue añadiéndole al motor una caja reductora, que no es más que un conjunto de engranajes [36].

Figura 28: Motor eléctrico DC – fuente: https://farm4.staticflickr.com/3712/12167636184_d4cbecc17f_o.png

- Motores paso a paso: pueden, a diferencia del motor DC, ser muy precisos tanto en posición como en velocidad. Su rotación completa se divide en pasos equitativos. Estos pasos se miden en grados y mientras más pequeños sean los pasos, más precisos serán. Los motores paso a paso son necesarios en aplicaciones donde prima la precisión; no sería el caso de este trabajo ya que aquí lo que prima es la fuerza o par generados y estos motores no presentan buen comportamiento para la aplicación que se precisa [37].

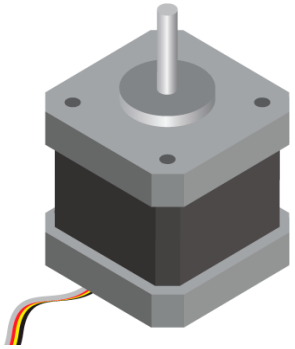


Figura 29: Motor paso a paso - fuente: https://farm9.staticflickr.com/8617/15666373503_8da6e084f5_o.png

- Servomotores: permiten controlar la posición en la que se ubican y mantenerse estables en dicha posición. Internamente un servomotor está formado por un motor de continua, una reductora y una circuitería electrónica junto con un potenciómetro, que se utilizan para conocer la posición del servomotor y controlar dicha posición aplicando una serie de pulsos. Generalmente este tipo de motores tiene un ángulo de giro limitado, que suele ser de 180º y esta característica hace que sean totalmente descartados para la aplicación que se va a necesitar [38].

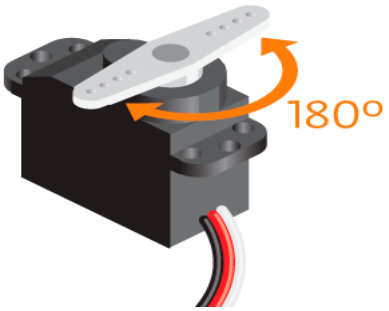


Figura 30: Servomotor – fuente: https://farm4.staticflickr.com/3701/12167450593_ee5d906dc1_o.png

Entonces se define que el motor a utilizar para el trabajo será el motor DC, y además tendrá que disponer de una caja reductora. Este motor es el más simple de entre las tres opciones y el que mejor relación presenta en lo respectivo al par que genera.

Los llamados Micro Metal Gearmotors, de la empresa Pololu, son unos micromotores de continua con reductora especialmente diseñados para robótica [39]. Además, ya se ha tenido contacto con este tipo de motores en un trabajo de Diseño de tarjetas electrónicas, asignatura cursada durante el Grado. Una ventaja de este tipo de motores es que disponen de varios accesorios, como soportes para engancharlos al chasis del robot, variedad de ruedas perfectamente acoplables, encoders. Es por todo esto que se elige esta clase de motores.

Figura 31: Micro Metal Gearmotors Pololu - fuente:https://www.pololu.com/product/1098

5.4.2 Driver motor

Un driver motor o controlador de motor [40] es un dispositivo conectado a una fuente de energía tal como una batería o fuente de alimentación, y a circuitos de control en forma de señales de entrada, analógicas o digitales. Sirve para gobernar de alguna manera el rendimiento de un motor eléctrico. En el caso de este trabajo es de un motor DC. Este dispositivo tendrá las siguientes funciones:

- Suministra la alimentación necesaria para el correcto funcionamiento del motor
- Puede incluir un accionamiento manual o automático para iniciar y detener el motor
- Da la posibilidad de cambiar fácilmente el sentido de giro
- Da también la opción de la regulación de la velocidad, así como regulación o limitación del par
- Presenta protección contra sobrecargas y posibles fallas que se puedan presentar

Figura 32: L298N Driver dual para motores - fuente: http://electronilab.co/tienda/driver-dual-para-motores-full-bridge-l298n/

- **PWM (Pulse Width Modulated)**

Para la regulación tanto de la velocidad como del par en el motor de corriente continua, se emplea la regulación por ancho de pulso, más conocida como PWM. Este tipo de regulación se basa en el hecho de recortar la corriente continua de alimentación en forma de onda cuadrada. La energía que recibe el motor disminuye de manera proporcional a la relación entre la parte alta (con energía) y baja (sin energía) del ciclo de la onda cuadrada. Controlando esta relación se logra la regulación deseada de una manera bastante aceptable [41].

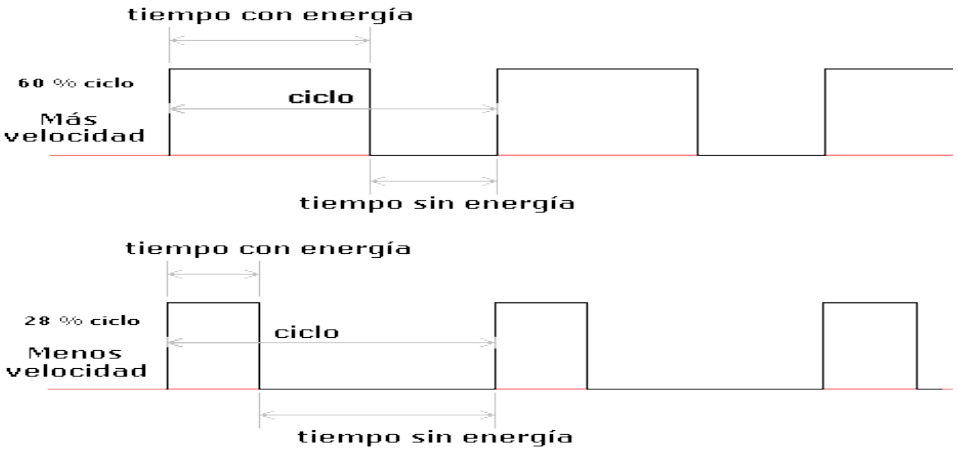


Figura 33: Regulación por ancho de pulsos (PWM) – fuente: http://robots-argentina.com.ar/MotorCC_ControlAncho.htm

- **Puente en H**

Para la función de cambio de sentido del motor DC se usa un circuito electrónico conocido como puente en H. Estos circuitos se construyen con 4 interruptores que pueden ser mecánicos o transistores, y adquieren su nombre de la típica representación gráfica del circuito. Los puentes en H están disponibles como circuitos integrados, pero también pueden construirse a partir de componentes discretos.

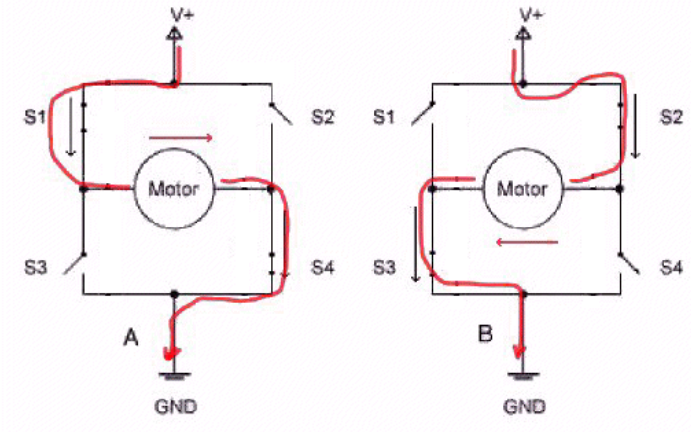


Figura 34: Funcionamiento Puente en H - fuente: http://www.hispavila.com/3ds/atmega/hpuente_files/giros_en_puente_h.gif

Su funcionamiento es sencillo. Al cerrarse los interruptores S1 y S4, y abrirse S2 y S3 se aplica una tensión positiva en los bornes del motor, haciéndolo girar en un sentido. Abriendo los interruptores S1 y S4, y cerrando tanto S2 como S3, el voltaje se invierte en los bornes, permitiendo el giro en sentido inverso del motor.

En la actualidad, existen numerosos drivers controladores de motor DC muy completos y montados en tarjetas electrónicas como el mostrado en la Ilustración 32. Para este trabajo, se ha elegido la opción de diseñar nuestro propio driver que se montará en una PCB, de esta manera se obtendrá un importante ahorro en comparación a haberlo comprado ya montado, y además esto permite una total libertad a la hora de elegir los componentes a utilizar. El driver estará basado en el dispositivo L298N, un encapsulado que presenta un doble puente en H. Existe mucha documentación para su uso con Arduino, es muy robusto, con un precio muy asequible (2.9 €), y se adapta sobradamente a las características que se necesitarán en este trabajo.

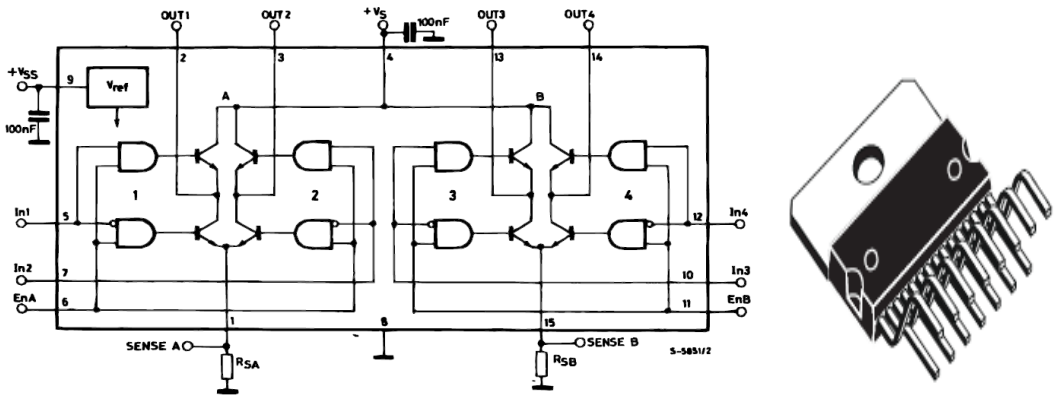


Figura 35: Conexionado interior y encapsulado L298N - fuente: L298 Datasheet

5.5 Módulo De Comunicación Bluetooth

La función de este módulo dentro del robot, es dar la capacidad de transmitir y recibir información con otros dispositivos sin la necesidad de cables, es decir, por un medio inalámbrico. Para ello se ha elegido la tecnología Bluetooth, debido a que es una tecnología muy popular y extendida entre los dispositivos de nueva generación de hoy en día. Entre sus principales características, está su robustez, baja complejidad, bajo consumo y bajo costo.

Figura 36: Logotipo Bluetooth – fuente: http://es.wikipedia.org/wiki/Bluetooth

Más que una tecnología, Bluetooth es un estándar para las comunicaciones inalámbricas de corto alcance o Redes Inalámbricas de Área Personal (WPAN), que posibilita la transmisión de voz y datos entre diferentes dispositivos mediante un enlace por radiofrecuencia en la banda ISM de los 2,4 GHz [42] [43].

Los principales objetivos que se pretenden conseguir con este estándar son:

- Facilitar las comunicaciones entre equipos móviles.
- Eliminar los cables y conectores entre éstos.
- Ofrecer la posibilidad de crear pequeñas redes inalámbricas y facilitar la sincronización de datos entre equipos personales.
- Obtener una tecnología de bajo coste y potencia que posibiliten dispositivos baratos.

La topología de las redes Bluetooth puede ser punto a punto o multipunto. Ésta, se basa en el modo de operación maestro/esclavo, un dispositivo maestro se puede conectar simultáneamente con hasta 7 dispositivos esclavos activos con un alcance radial de hasta 10 m o incluso más, si se utilizan los amplificadores adecuados. A la red que forma un dispositivo y los dispositivos que se encuentran dentro de su rango, se le denomina piconet. Además, se puede extender la red mediante la formación de scatternets. Una scatternet es la red producida cuando dos dispositivos pertenecientes a dos piconets diferentes, se conectan. Pueden coexistir hasta un máximo de 10 piconets dentro de una sola área de cobertura.

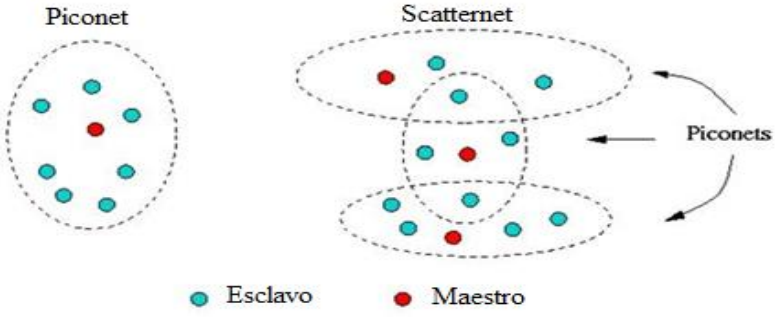


Figura 37: Topología Bluetooth – fuente: http://www.gta.ufrj.br/grad/09_1/versao-final/bluetooth/Page323.htm

El objetivo del módulo Bluetooth dentro del trabajo es poder establecer una conexión con un Smartphone. Una vez establecida esta conexión y con ayuda de aplicaciones software de los Smartphone, se puede desplazar al robot como si de un radiocontrol se tratase. Y también, da la opción de monitorizar en tiempo real variables importantes del sistema de control.

Existen muchos módulos Bluetooth para ser usados en proyectos de electrónica, pero los más utilizados son los módulos de JY-MCU, debido a que son muy económicos y fáciles de encontrar en el mercado. Son módulos pequeños y con un consumo muy bajo que permiten agregar funcionalidades Bluetooth con facilidad a la placa Arduino. Estos módulos contienen el chip con una placa de desarrollo con los pines necesarios para la comunicación serie.

Hay dos modelos de JY-MCU: el HC-05, el cual puede funcionar como maestro/esclavo, y está también el HC-06 que solo puede actuar como esclavo. La diferencia entre maestro y esclavo es que en modo esclavo es el dispositivo quien se conecta al módulo, mientras que en modo maestro es el módulo quien se conecta con un dispositivo.

Ya que se quiere que el Smartphone actúe de maestro y el módulo siempre de esclavo, se ha seleccionado el correspondiente módulo Bluetooth HC-06 con características suficientes para poder cumplir con el objetivo propuesto. Si se quisiera poder establecer conexión con otros robots Arduino se debería emplear un HC-05.

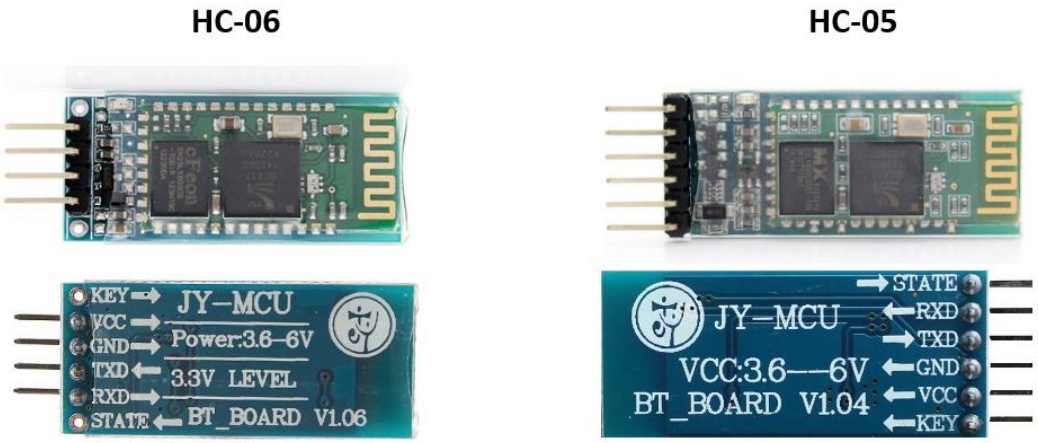


Figura 38: Módulos Bluetooth JY-MCU - fuente:http://diymakers.es/arduino-bluetooth/

6 DISEÑO Y DESARROLLO DEL TRABAJO

Una vez presentada en el apartado anterior, la tecnología que se emplea en este Trabajo Fin de Grado, empieza el diseño y desarrollo del mismo, así como la descripción del modo en el que se emplea esta tecnología. En este apartado se pasa a explicar los pasos que se han ido siguiendo en un orden cronológico, y se muestran tanto los resultados obtenidos como las pertinentes modificaciones que se realizaron hasta llegar a la solución final.

6.1 Estimación Del Ángulo De Inclinación

Como ya se comentó antes, el dispositivo sensor que proporcionará el ángulo de inclinación va a ser un IMU de 6 grados de libertad, concretamente el GY-521 (MPU6050). Este dispositivo como ya se analizó, se sabe que tiene una interfaz de comunicación con la placa Arduino mediante el protocolo I2C.

I2C [44] es un bus de comunicaciones en serie. Su nombre viene de Inter-Integrated Circuit. Su velocidad de transmisión en el modo estándar alcanza los 100 kbit/s, aunque también permite velocidades de 3.4 Mbit/s. Es un bus principalmente usado para comunicar microcontroladores y sus periféricos en sistemas integrados (Embedded Systems), como es el este caso del IMU con Arduino.

La principal característica de I2C es que utiliza dos líneas para transmitir la información, una para los datos y otra para la señal de reloj. También es necesaria una tercera línea, que es la referencia (masa). Como suelen comunicarse circuitos en una misma placa que comparten una misma masa esta tercera línea no suele ser necesaria. Las líneas se llaman:

- SDA: datos
- SCL: reloj
- GND: tierra

Hay que destacar la existencia para Arduino de la librería Wire, así como la de I2Cdev, en las que se incluyen funciones para este tipo de comunicación.

El dispositivo IMU MPU6050 tiene la posibilidad de obtener los ángulos de inclinación de dos formas distintas [45]:

- Mediante la toma de datos de los valores en bruto de acelerómetro y giroscopio, para posteriormente, aplicando fórmulas matemáticas y filtros que fusionen las medidas, obtener los correspondientes ángulos.

- Mediante la tecnología de Procesador Digital de Movimiento (DMP) que contiene este dispositivo, y puede realizar la fusión de datos en el chip IMU por sí mismo independientemente del microcontrolador principal.

6.1.1 Ángulos mediante valores en bruto (raw values)

La gravedad de la Tierra es perpendicular al suelo. Como es lógico, la IMU detecta la aceleración de la gravedad terrestre y gracias a ello se pueden usar las lecturas del acelerómetro para saber cuál es el ángulo de inclinación respecto al eje X o eje Y.

Supongamos la IMU paralela con el suelo. Entonces, el eje Z marcará 9.8, y los otros dos ejes marcarán 0. Ahora girando 90 grados por el eje Y, es el eje X el que queda perpendicular al suelo, por lo tanto marcará la aceleración de la gravedad.

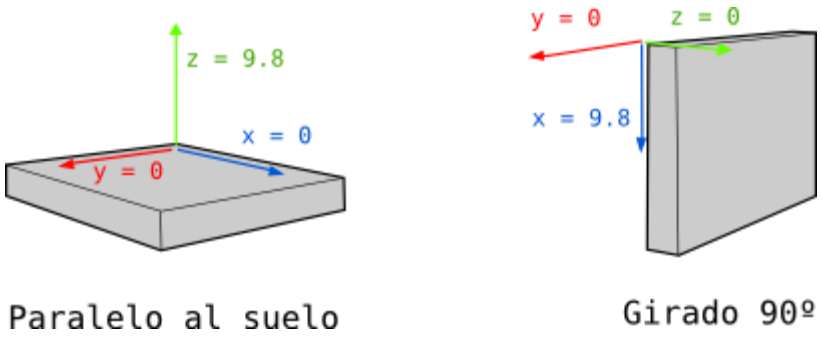


Figura 39: Obtención del ángulo de inclinación con el acelerómetro – fuente: https://robologs.files.wordpress.com/2014/10/imu_ac_ejemplos.png?w=300&h=119

Al saber que la gravedad es 9.8 m/s², y las medidas que dan los tres ejes del acelerómetro, por trigonometría es posible calcular el ángulo de inclinación de la IMU mediante las siguientes fórmulas:

$$AnguloX = atan\left(\frac{y}{\sqrt{x^2 + z^2}}\right)$$

$$AnguloY = atan\left(\frac{x}{\sqrt{y^2 + z^2}}\right)$$

Figura 40: Fórmulas obtención ángulo del acelerómetro – fuente: http://robologs.net/2014/10/15/tutorial-de-arduino-y-mpu-6050/

Por otro lado, si se sabe el ángulo inicial de la IMU, se puede ir sumando el valor que marca el giroscopio para saber el nuevo ángulo a cada momento. Supongamos que iniciamos la IMU a 0º. Si el giroscopio realiza una medida cada segundo, y marca 3 en el eje X, tendremos el ángulo con esta sencilla fórmula:

$$AnguloY = AnguloYanterior + x\Delta t$$

Figura 41: Fórmula obtención ángulo del giroscopio - fuente: http://robologs.net/2014/10/15/tutorial-de-arduino-y-mpu-6050/

Sin embargo, existen dos problemas fundamentales: el acelerómetro es capaz de medir cualquier ángulo, sin embargo sus lecturas son ruidosas y tienen un cierto margen de error. Por otra parte está el giroscopio, que no le afecta el ruido, pero al realizar los cálculos del ángulo es inevitable que se produzca un pequeño error acumulativo, que con el tiempo va agrandándose y se obtienen medidas muy imprecisas.

Pero existe una forma de conseguir eliminar estos problemas y lograr combinar los datos del acelerómetro y el giroscopio para así obtener medidas aceptables. El empleo de un filtro, denominado filtro complementario consigue este propósito. Este filtro se implementa con la siguiente fórmula:

$$Ángulo\ X = (1 - \beta) \cdot (Ángulo\ X + \ ÁnguloGiroscopio \cdot 0{,}01) + \beta \cdot ÁnguloAcelerómetro$$

Figura 42: Fórmula del Filtro Complementario

Donde β, comprende valores de 0 a 1, y queda como variable a definir para obtener mejores o peores resultados de medida según sea el caso.

6.1.2 Ángulos mediante Procesador Digital de Movimiento (DMP)

El MPU6050 contiene un DMP que fusiona los datos del acelerómetro y el giroscopio para minimizar los efectos de los errores presentes en cada sensor. El DMP calcula los resultados en términos de cuaterniones, son una extensión de los números reales similar a la de los números complejos y en la física representan rotaciones en el espacio, y puede convertir los resultados en ángulos que se precisan en este TFG como el cabeceo (pitch), alabeo (roll), y guiñada (yaw). El fabricante del dispositivo, Invensense no revela el algoritmo que utilizan para combinar los datos, pero existe para Arduino un código o sketch ejemplo capaz de obtener estos ángulos. Este código o sketch ejemplo se llama MPU6050_DMP6, y se encuentra en la subcarpeta MPU6050 de la librería i2cdevlib, la cual fue desarrollada mediante ingeniería inversa por Jeff Rowbergs [46], un desarrollador y aficionado a la tecnología. Esta librería es de libre acceso y fácilmente descargable desde la web.

6.1.3 Elección del método a utilizar y pasos a seguir

Se eligió el uso del DMP para la obtención de los ángulos de inclinación por la gran ventaja que presenta al eliminar la necesidad de realizar numerosos y complejos cálculos intensivos con la placa Arduino y poder ejecutarlos en el procesador propio del IMU. Además haciendo un análisis de foros y blogs sobre el tema, se constata que los resultados obtenidos con el uso del DPM son mejores que con el filtro complementario [47] [48].

Para el empleo del DMP, se debe conexionar el MPU6050 de la siguiente manera:

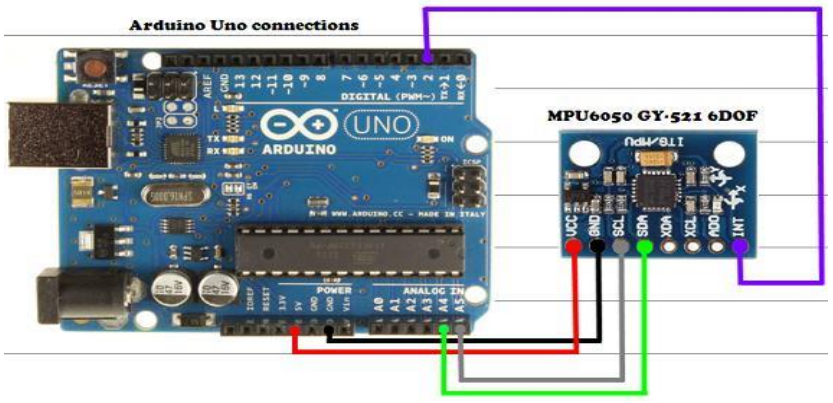
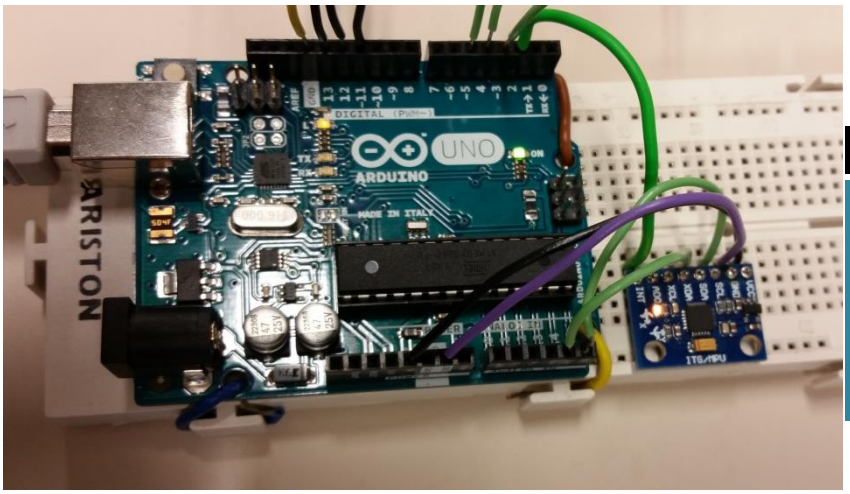


Figura 43: Conexionado Arduino Uno y MPU6050 – fuente: http://42bots.com/tutorials/arduino-uno-and-the-invensense-mpu-6050-6dof-imu/



MPU6050	Arduino UNO Pins
VCC	5V
GND	GND
SDA	A4 (I2C SDA)
SCL	A5 (I2C SLC)
INT	D2 (interrupción # 0)

Figura 44: Conexionado en protoboard Arduino Uno y MPU6050

Después se pasa a la parte de programación. Primero que nada, es obligatorio una rápida introducción al entorno de programación software IDE Arduino:

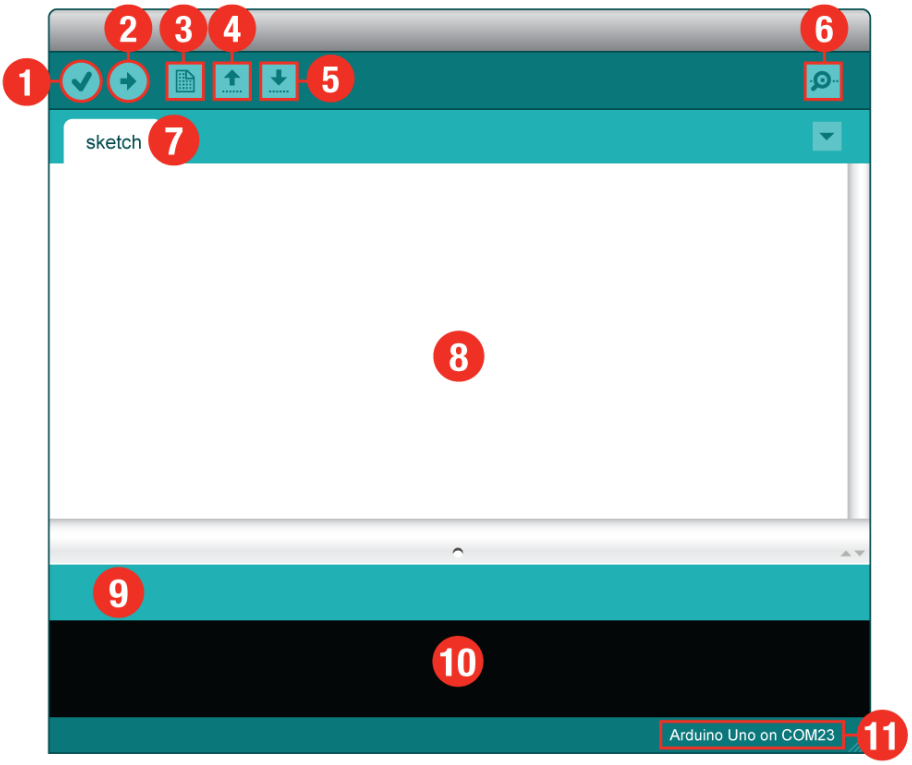


Figura 45: Entorno de programación software IDE Arduino – fuente: https://learn.sparkfun.com/tutorials/sik-experiment-guide-for-arduino---v32

(1) **Verificar**: compila el código escrito y detecta errores en la sintaxis.
(2) **Subir**: envía el código escrito a la placa Arduino.
(3) **Nuevo**: abre una nueva ventana en la que poder escribir código.
(4) **Abrir**: permite abrir código previamente ya escrito y guardado, abre un sketch.
(5) **Salvar**: guarda código escrito en el área de código, guarda un sketch.
(6) **Monitor Serial**: abre una ventana que muestra toda la información Serial que la placa está transmitiendo. Es muy útil para la función debugging.
(7) **Nombre del sketch**: muestra el nombre del sketch con el que se está trabajando.

(8) **Área de código**: área en donde se va a escribir el código del sketch.
(9) **Área de mensaje**: área en la que el IDE dice si existen errores en el código.
(10) **Consola de texto**: muestra los mensajes de error completos.
(11) **Puerto Serial y placa**: muestra la placa y el puerto con que se está trabajando.

Para poder conseguir los ángulos mediante el DMP hay que seguir estos sencillos pasos:

1. Descargar de la web https://github.com/jrowberg/i2cdevlib/archive/master.zip el archivo zip. Descomprimiendo este archivo se encuentra la carpeta llamada "i2cdevlib-master" que contendrá una subcarpeta "Arduino", se abre esta carpeta y se localiza la librería "MPU6050" y la librería "I2Cdev", las dos necesarias para la ejecución del mencionado sketch "MPU6050_DMP6".

2. Instalar las dos librerías en el software IDE Arduino, y para ello, basta con copiar las carpetas "MPU6050" y "I2Cdev" a la misma ubicación que las demás librerías Arduino. C:\Program Files (x86)\Arduino\libraries por defecto.

3. Iniciar el software IDE Arduino y abrir el sketch ejemplo, Archivo → Ejemplos → MPU6050 → Examples → MPU6050_DMP6.

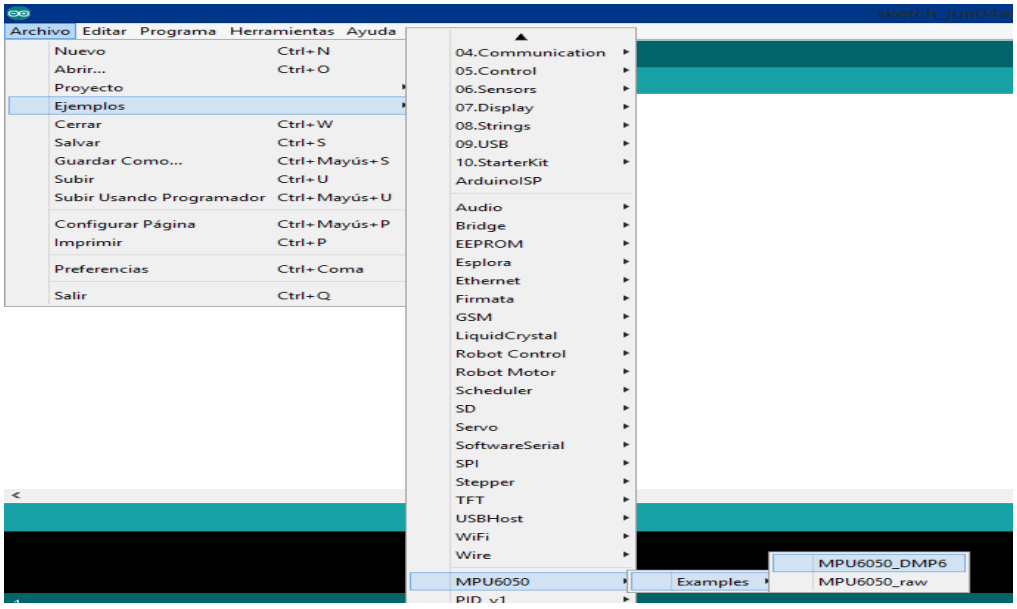


Figura 46: Abrir Sketch ejemplo MPU6050_DMP6

4. Subir el código a la placa Arduino. Después, abrir el monitor serial y establecer la velocidad de transmisión como 115200. A continuación, comprobar que sale "Initializing I2C devices…", en caso contrario se tiene que pulsar el botón reset de la placa Arduino. Si todo va bien, se aprecia una nueva línea con "Send any character to begin DMP programming and demo: ". Para continuar, escribir un carácter cualquier en el monitor y enviarlo. Poco después, se comenzará a visualizar los valores de los ángulos de inclinación buscados; yaw, pitch y roll determinados con las medidas de la IMU.

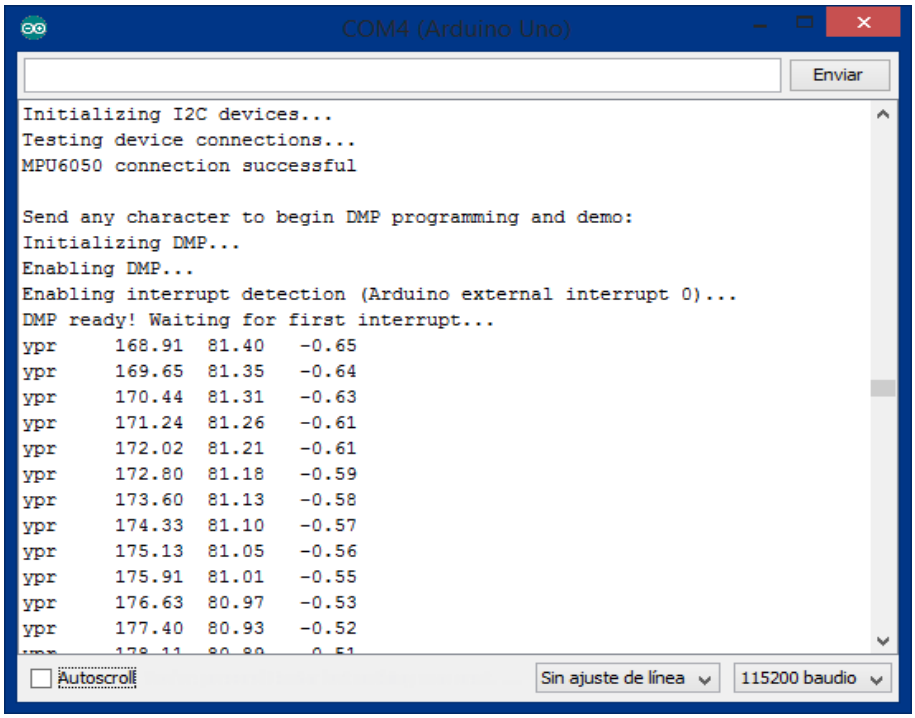


Figura 47: Monitor Serial con ángulos de inclinación del DMP

Para apreciar con mayor exactitud la precisión de los valores obtenidos por el IMU, existe la posibilidad de representarlos en un entorno 3D. Para ello, es necesario descargar e instalar el software IDE Processing desde su página web, https://processing.org/download/?processing. Este software es muy similar al software IDE Arduino, a excepción de un par de funciones. Processing se utiliza principalmente para la visualización de datos, haciéndolo en modelos 2D / 3D.

Una vez instalado Processing, es necesario descargar e instalar una librería necesaria para poder realizar la representación. Esta librería se llama "toxiclibs-cpmplete-0020" y se puede obtener totalmente gratis desde el sitio web, http://hg.postspectacular.com/toxiclibs/downloads/. Para la instalación de la librería, basta con copiarla en el correspondiente directorio, Processing\processing-2.2.1\modes\java\libraries.

Por último, se carga el código ejemplo MPU6050_DMP6 a la correspondiente placa Arduino. Y a continuación, se procede a abrir y ejecutar el código de Processing "MPUTeapot.pde" que viene dentro de la librería Arduino MPU6050 y está localizado en Arduino\libraries\MPU6050\Examples\MPU6050_DMP6\Processing. Entonces, si todo va bien, se logra visualizar un objeto 3D en forma de flecha o avión que se mueve de acuerdo con la orientación que se le va dando al IMU en tiempo real.

El código de programación que se emplea para la obtención del ángulo de inclinación del robot, estará basado en el sketch ejemplo MPU6050_DMP6, y se ha hecho una simplificación significativa, ya que este sketch da la posibilidad de obtener otras muchas medidas totalmente innecesarias para el propósito perseguido. El código empleado puede verse en el Anexo 1.

Partiendo de la base que los ángulos Yaw, Pitch y Roll son cero, en la posición y orientación mostrada en la siguiente figura:

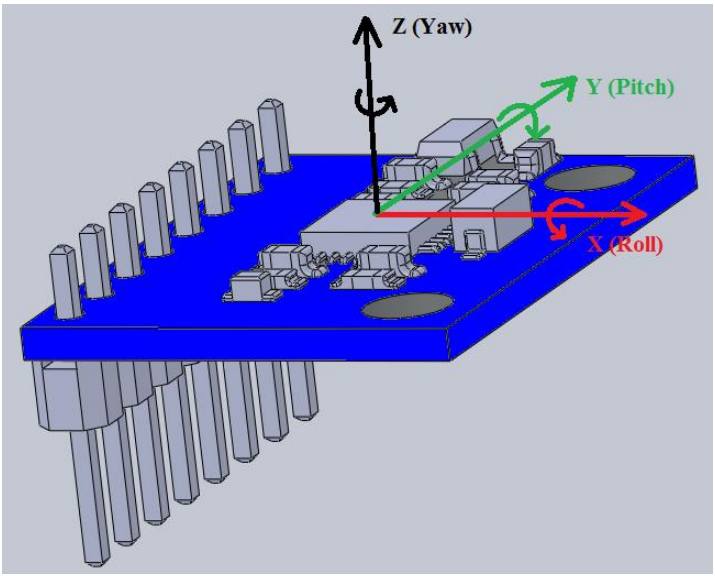
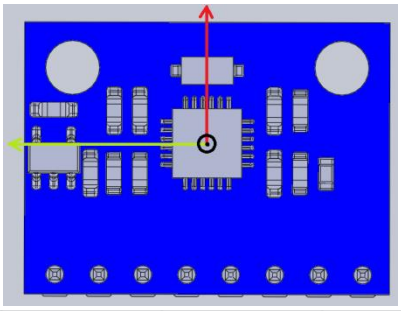


Figura 48: Posición y orientación origen del IMU

se realizaron las medidas de los ángulos en dos orientaciones estáticas elegidas, las cuales son mostradas en las imágenes, y se obtuvieron las gráficas correspondientes a cada posición mediante el software Excel y los datos suministrados por el código:



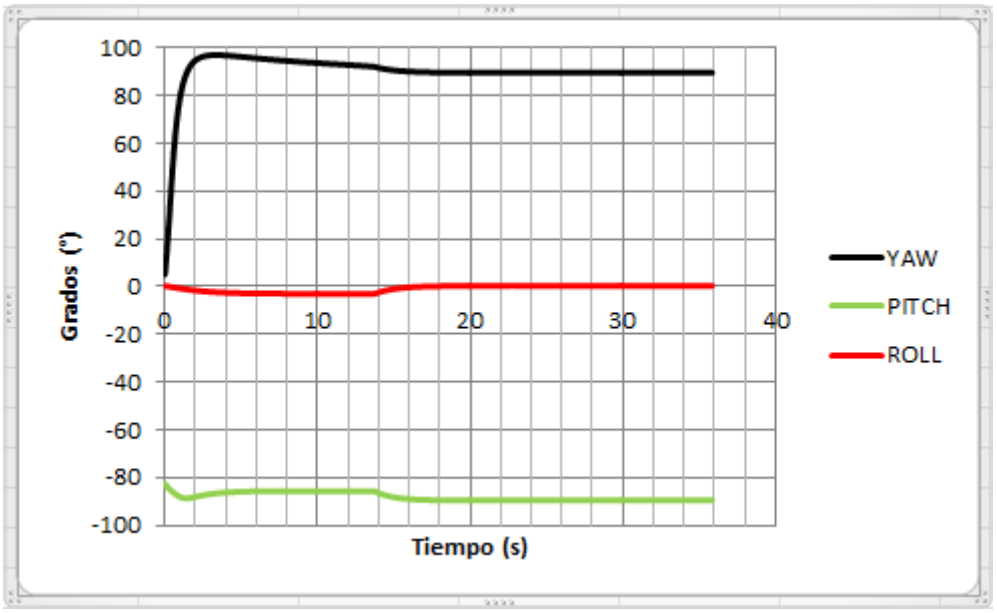


Figura 49: Orientación 1 y su respectiva gráfica

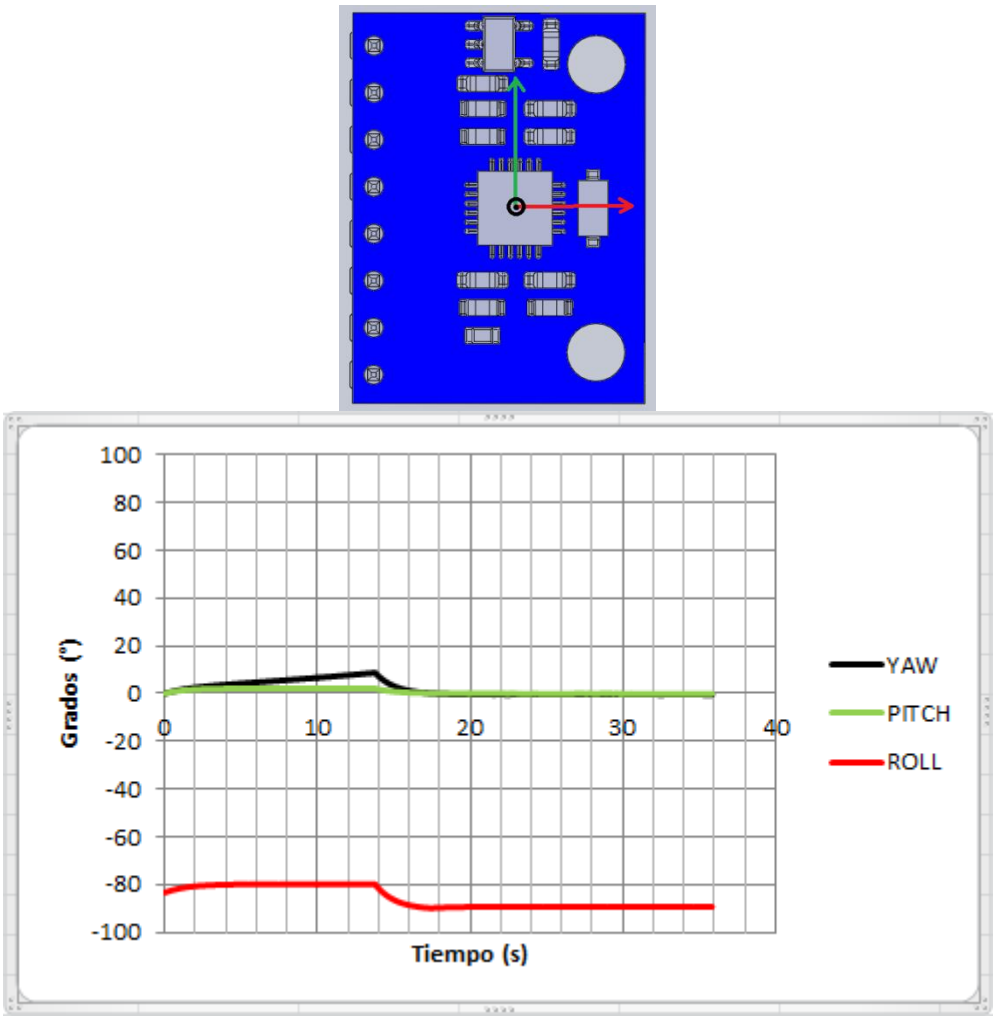


Figura 50: Orientación 2 y su respectiva gráfica

Con esta sencilla prueba se puede apreciar la fiabilidad de los ángulos medidos por el sensor, y además se puede sacar como conclusión que el sensor necesita de un tiempo de estabilización al principio para dar datos precisos. Este tiempo se establece aproximadamente en torno a unos 15 segundos después de empezar a medir.

Por lo tanto cuando se intente el control auto-balanceado del robot, habrá que esperar en torno a 15 segundos para que el sistema empiece a presentar un funcionamiento normal y adecuado. Ya que hasta que no pasen esos segundos iniciales, los ángulos que llegan a la realimentación no son fiables.

A continuación, se presentan imágenes de las visualizaciones obtenidas con el software Processing, las cuales confirman la precisión de los ángulos que se obtienen mediante DMP:

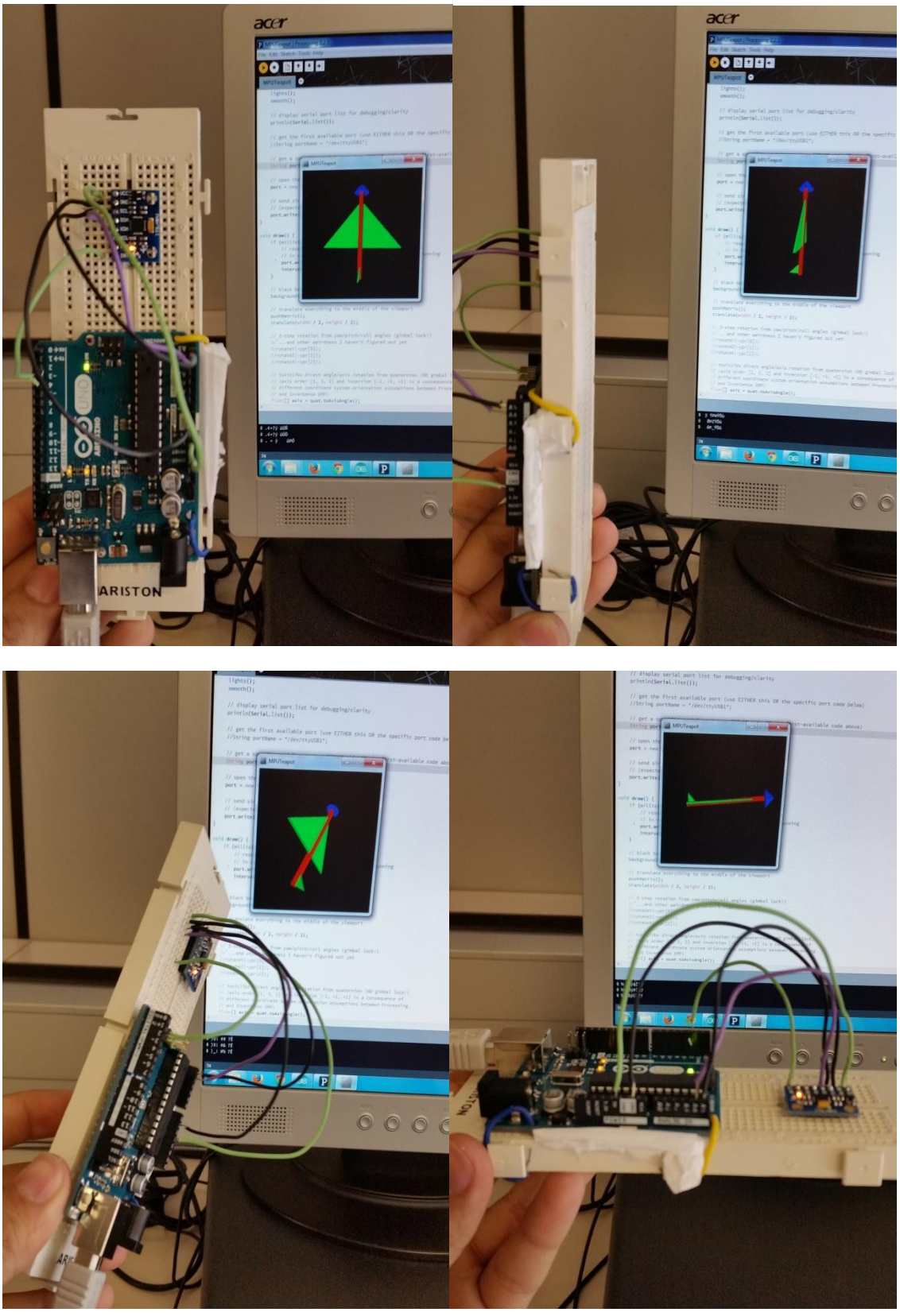


Figura 51: Visualización de IMU con Processing

6.2 Sistema De Potencia

Como ya se comentó, el sistema de potencia lo forman los dos motores DC y su correspondiente driver controlador.

Para el driver controlador, se optó por una placa de circuito impreso de propia fabricación basada en el encapsulado L298N.

La placa a diseñar, será capaz de realizar un control independiente para cada uno de los motores, dispondrá de diodos de protección contra inversión de polaridad, también tendrá un interruptor principal que gobierne todo el driver y un led que indique si está en marcha o no. Además, el regulador de tensión de nuestro sistema que pasa de los 9V a los 5V también se localizará en esta placa.

El primer paso fue realizar un prototipo montado en una protoboard donde se intentó poner todas las especificaciones solicitadas. Y con este prototipo, tener la primera toma de contacto con el driver motor y las primeras pruebas.

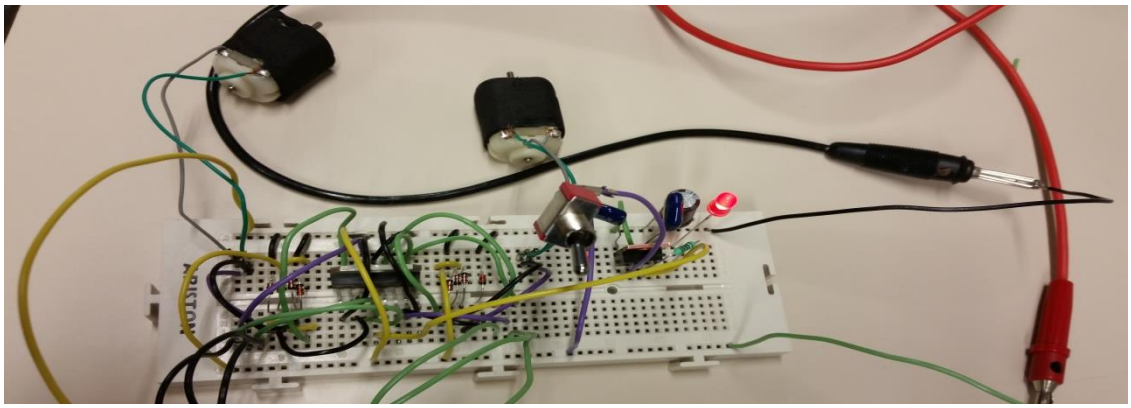


Figura 52: Montaje prototipo pruebas driver motor

Para las pruebas con el driver, se emplearon unos pequeños motores DC obtenidos de un mando de videoconsola estropeado, donde los motores cumplían el papel hacer vibrar el mando. Se recurren a estos motores debido a que no se disponía de los motores finales a emplear en el trabajo y para las pruebas a realizar fueron más que suficiente.

El código del sketch Arduino sobre las pruebas realizadas del correcto funcionamiento del circuito prototipo puede verse en el Anexo 2. Esta prueba consiste básicamente en enviar un carácter numérico en el monitor serial del software IDE Arduino. Este carácter puede ir del 1-6 y, dependiendo de cuál sea el carácter, se ejecuta una orden hacia los motores. Por lo tanto se tienen 6 acciones diferentes:

1. Motor A gira en sentido anti-horario a máxima velocidad
2. Motor A se para - stop
3. Motor A gira en sentido horario a máxima velocidad
4. Motor B gira en sentido anti-horario a máxima velocidad
5. Motor B se para - stop
6. Motor B gira en sentido horario a máxima velocidad

Una vez comprobado el buen funcionamiento del circuito prototipo, se pasó a diseñar la PCB en donde van montados y soldados todos elementos que forman el driver. El diseño y desarrollo de la PCB se explica en el apartado correspondiente de la memoria.

Para la construcción de esta placa, se ha tomado como referencia las placas driver de motores existentes en el mercado, y se intentó asemejarla lo máximo posible a las citadas placas, dando como resultado lo siguiente:

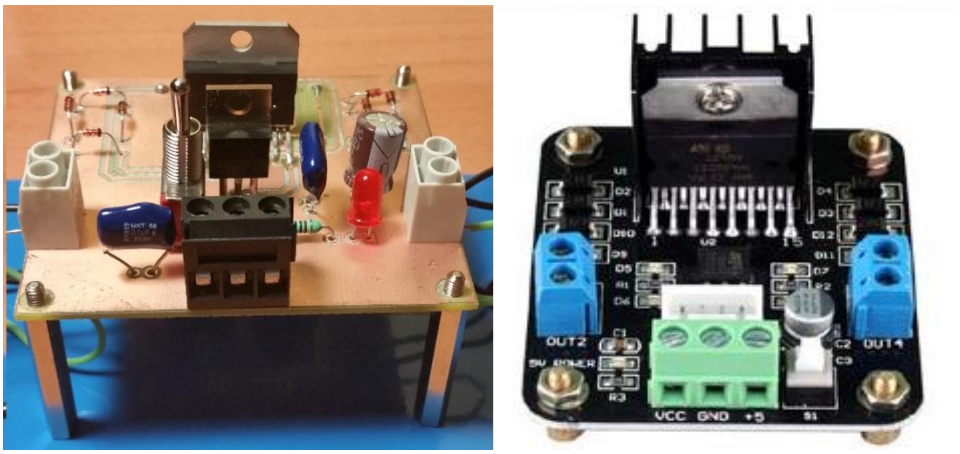


Figura 53: Placa Driver Final de propia construcción en comparativa a una del mercado

6.3 Diseño Y Elaboración De La Estructura Robótica

Como ya se mencionó en el apartado de descripción del trabajo, se diseñó y construyó la estructura robótica que va a albergar los distintos componentes del citado robot. Para tal fin, se usó el software de diseño SolidWorks.

La forma de la estructura elegida, viene dada del análisis del estado del arte de varios robots auto-balanceados existentes. Esta forma consiste en la disposición de tres bases horizontales una sobre otra, y estas bases son soportadas por cuatro varillas situadas en las cuatro esquinas de cada base.

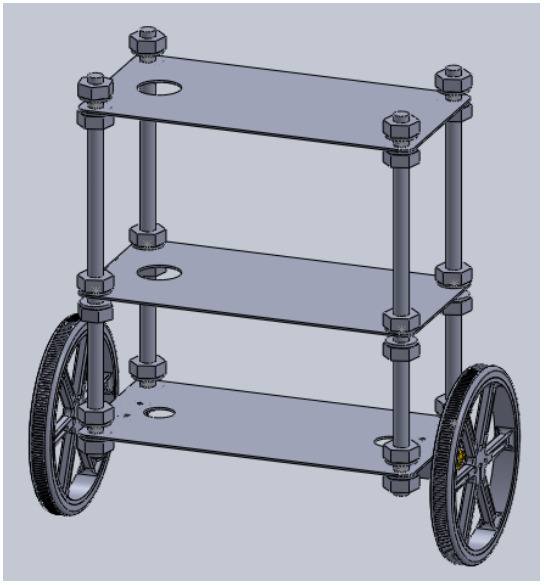


Figura 54: Forma de la estructura robótica elegida

Se eligió esta forma de estructura por su sencillez de diseño, por aparentar ser bastante ligera, por la facilidad de poner y quitar elementos del sistema en las bases; además presenta la posibilidad de mover el centro de masas según convenga y de esta manera tener una estructura más equilibrada.

6.3.1 Software SolidWorks

SolidWorks [49] es un software CAD (Diseño Asistido por Computadora) para modelado mecánico en 3D, desarrollado en la actualidad por SolidWorks Corp. Este tipo de softwares facilitan y agilizan los trabajos de diseño y cálculo.

Figura 55: Logo SolidWorks - fuente: http://imgsoup.com/1/solidworks-logo/

SolidWorks permite modelar piezas y conjuntos; y además extraer de ellos tanto planos técnicos como otro tipo de información necesaria para la producción, todo esto de una manera bastante automatizada y rápida.

El software presenta una gran ventaja, y es que hay disponibilidad de numerosos manuales de utilización, tanto del propio desarrollador del programa como de personas ajenas a él.

A continuación, se explicará los pasos seguidos para el diseño de la estructura deseada:

1. Primero se ejecuta el programa SolidWorks y nos aparece el entorno gráfico, donde para empezar a trabajar se abre un nuevo documento. Al abrir un nuevo documento, el programa nos da tres opciones a elegir: crear una pieza, crear un ensamblaje (a partir de piezas creadas) o crear un dibujo (plano a partir de piezas creadas). Se empezará por crear una pieza, en concreto las bases de la estructura robótica.

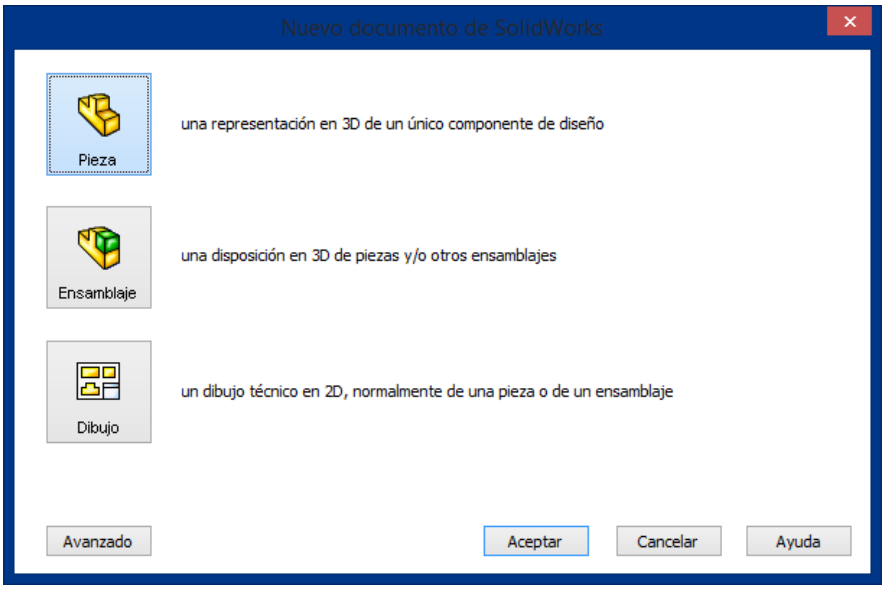


Figura 56: Ventana SolidWorks para crear un nuevo documento

2. Aparece una ventana tal y como se muestra en la Ilustración 54, donde se pasa a dibujar el croquis en 2D de la pieza que se quiere desarrollar en el plano que más convenga, ayudándose de las múltiples herramientas de dibujo del software.

   En este caso, se empieza por dibujar un rectángulo y posteriormente redondear sus esquinas. Luego se acota el dibujo con las medidas que se desean que tenga la pieza.

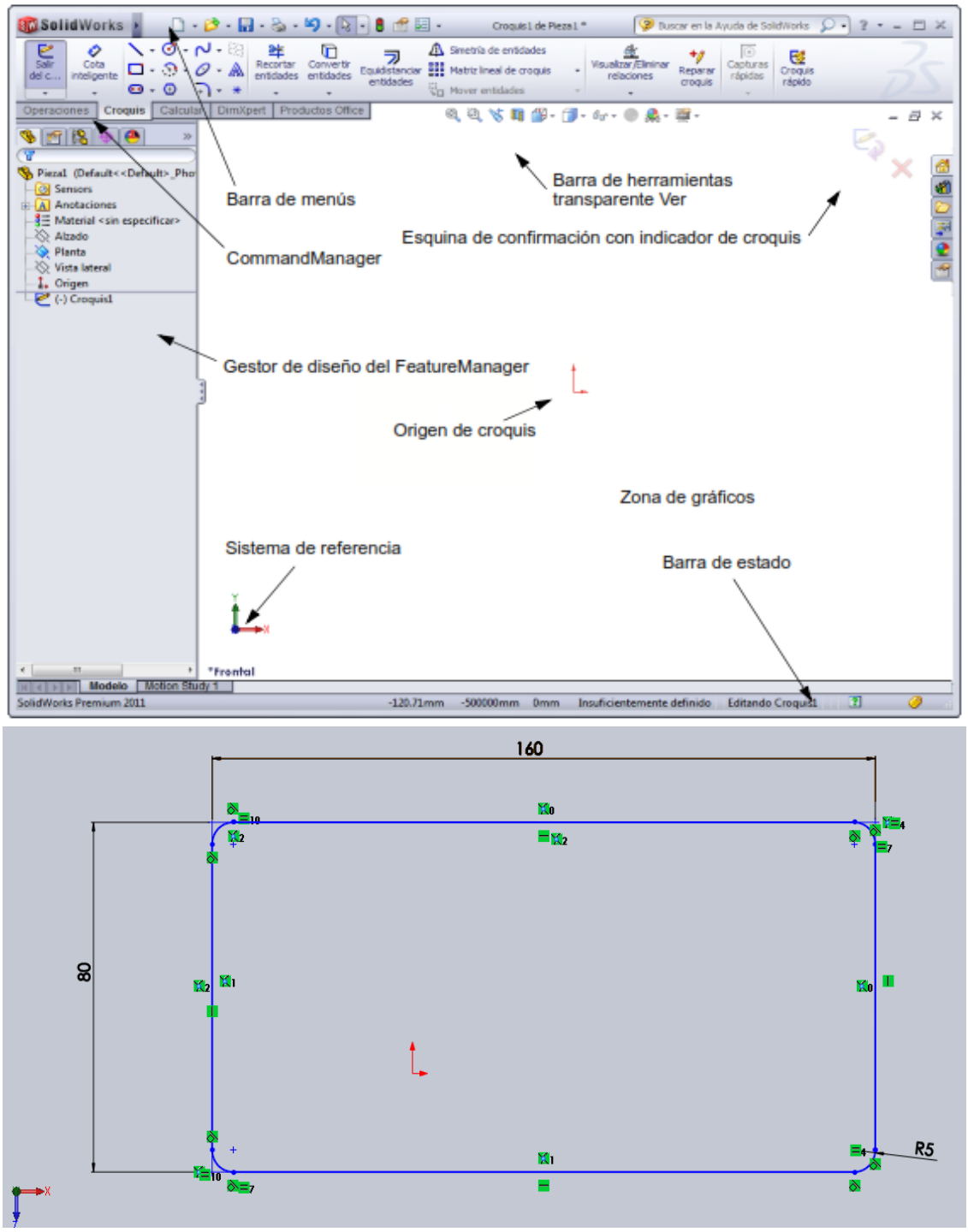


Figura 57: Perspectiva general de la ventana de SolidWorks y croquis de la base

3. Una vez definido el croquis final, para la obtención de la pieza en 3D se somete al croquis a la operación de "Extruir saliente/base" , tal y como se muestra en la Ilustración 55. En esta operación hay que definir la profundidad que se le quiere dar a la pieza 1.00mm, en este caso 1mm.

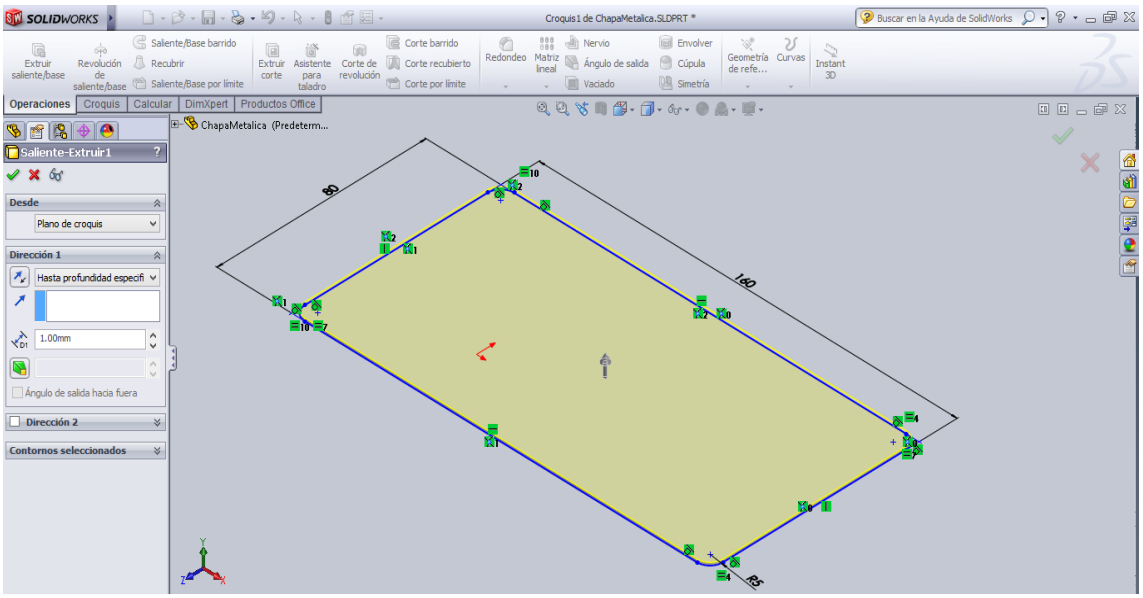
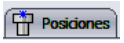
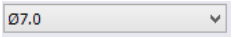
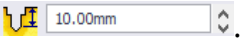


Figura 58: Operación de extruir croquis

4. Obtenida la pieza que va a servir como de bases de la estructura, hay que tener en cuenta que en estas bases van a ir colocados los distintos elementos que forman el conjunto robótico: PCBs, batería, motores junto con ruedas, etc.
   Por ello hace falta la realización de agujeros para el futuro acoplamiento tanto de motores como de las PCBs, además también se precisan agujeros en cada esquina para el paso de las varillas que soportarán las bases y agujeros para el paso de cables que permitan conexionar las etapas del sistema.

   Para agujerear la pieza se recurre a la operación "Asistente para taladro"  donde hay que precisar los puntos donde se va a agujerear 

y el tamaño de estos agujeros 

con su profundidad 

.

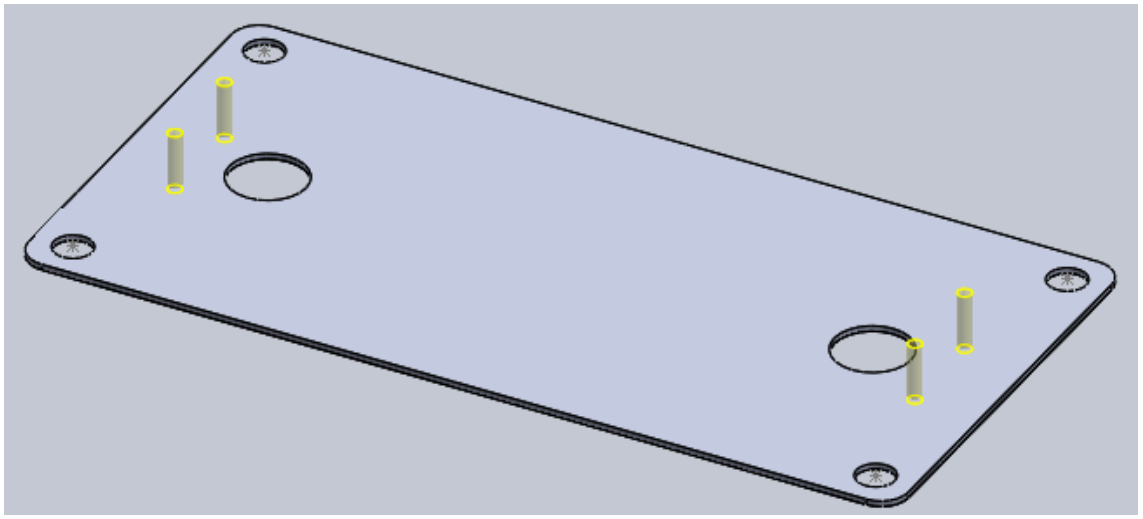


Figura 59: Pieza base agujereada SolidWorks

5. Se pasa a crear la varilla que cumplirá la función de soporte para las tres bases. Se empieza de la misma forma, a través de un croquis previo con sus respectivas cotas.

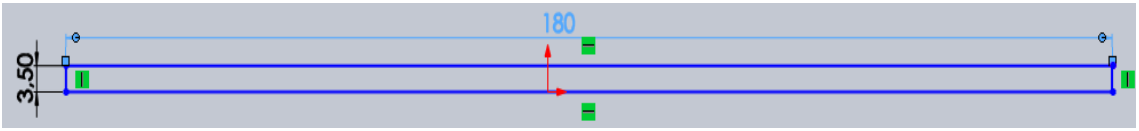
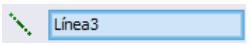


Figura 60: Croquis de la varilla

6. Como en este caso la pieza es una pieza de revolución, se opta por la operación de "Revolución de saliente/base", en donde hay que definir el eje de revolución 

que para este caso es una de las líneas de 180mm, y además hay que especificar el ángulo de revolución 

con 360º para una revolución completa.

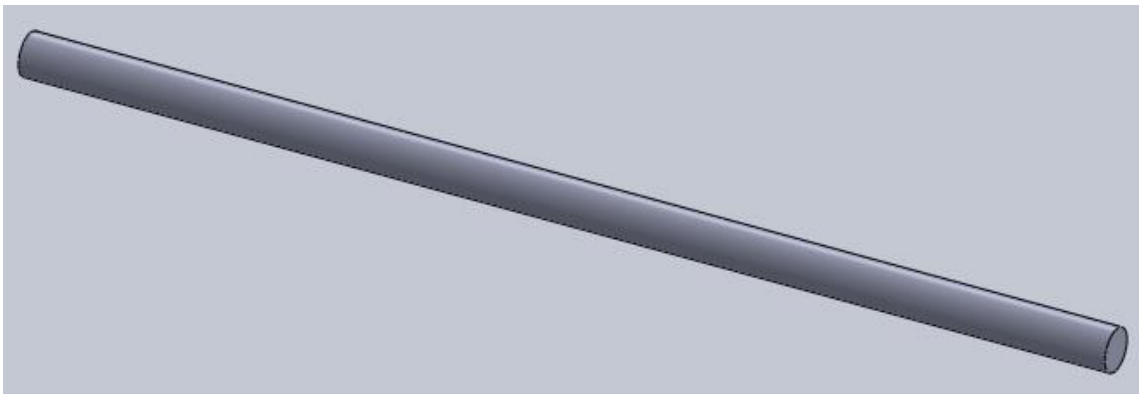


Figura 61: Varilla SolidWorks

Para la obtención de las tuercas y arandelas se realizan operaciones muy parecidas a las ejecutadas para la base y varilla.

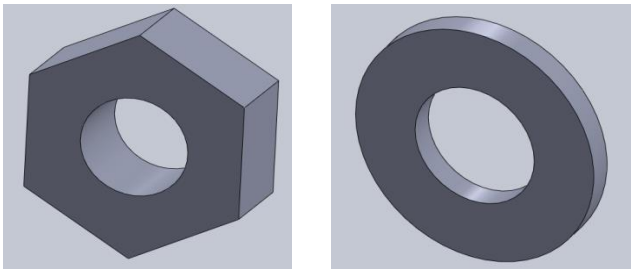
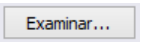


Figura 62: Tuerca y Arandela SolidWorks

7. Una vez que se han diseñado todas las piezas principales de la estructura, se abre un nuevo documento para crear un ensamblaje y así, poder hacerse una idea de cómo quedará la estructura final diseñada. Al abrir el nuevo ensamblaje, aparece una ventana como la mostrada en la Ilustración 60.
   En esta ventana hay que insertar las piezas antes diseñadas, localizándolas en el directorio donde hayan sido guardadas 

.

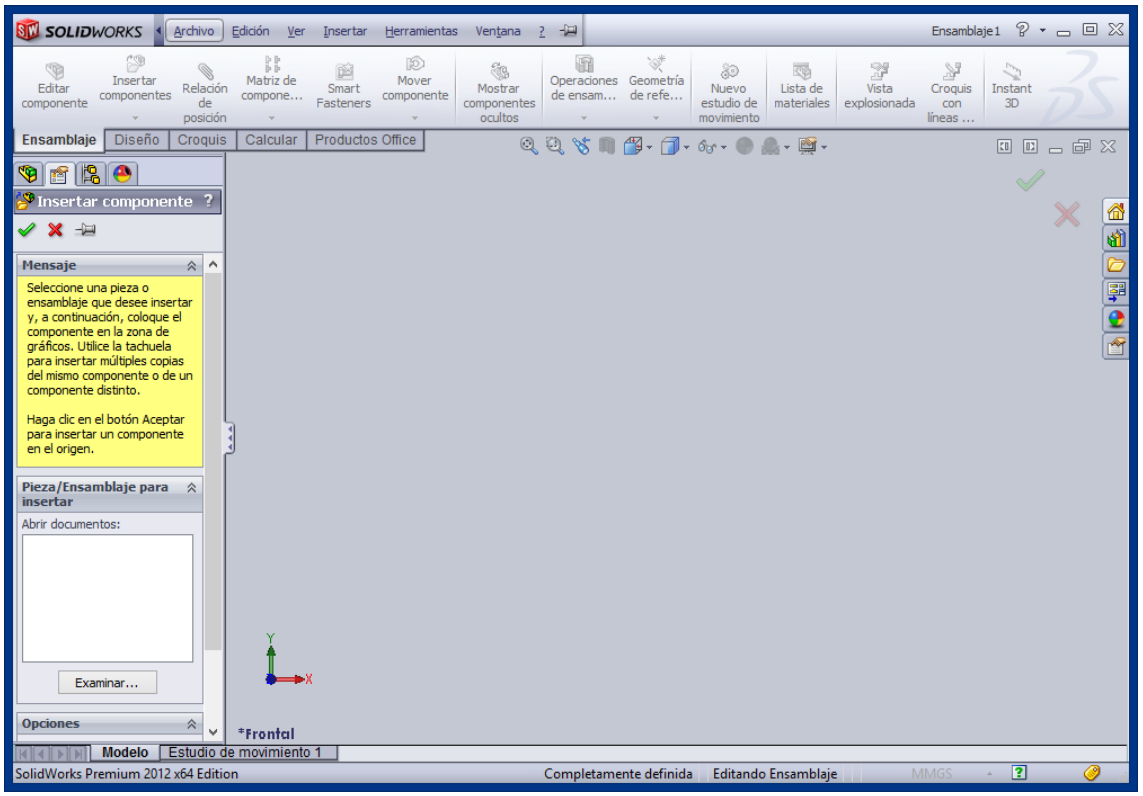


Figura 63: Perspectiva de la ventana de nuevo ensamblaje

8. Después de insertar las piezas, todas ellas quedan desperdigadas y sin ningún orden en la zona de gráficos como se aprecia en la siguiente imagen:

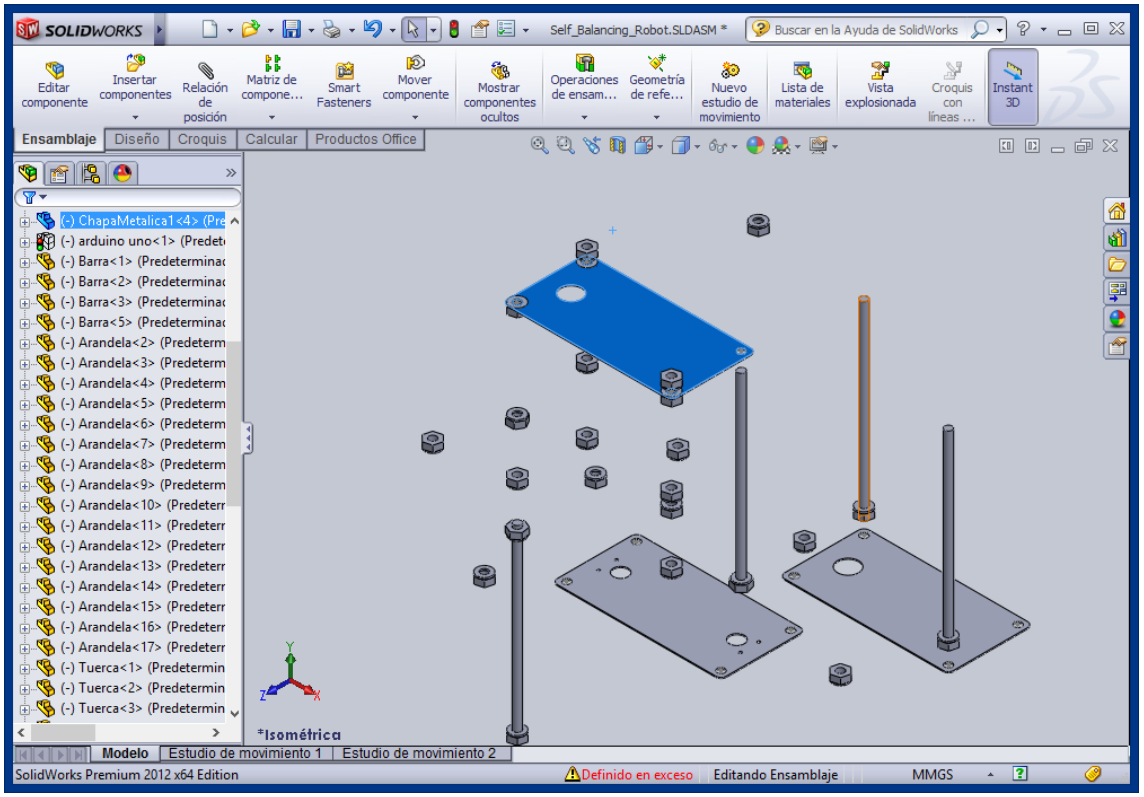


Figura 64: Ensamblaje de las piezas sin relación de posición

Para poder ordenarlas y posicionarlas según se requiere, es necesario usar la operación de ensamblaje "Relación de posición" . Esta operación dispone de múltiples opciones de posicionamiento de las piezas, que permitirán ordenar y posicionar correctamente las piezas de la forma en la que deben ir. Así que las piezas quedaron perfectamente posicionadas de la siguiente manera:

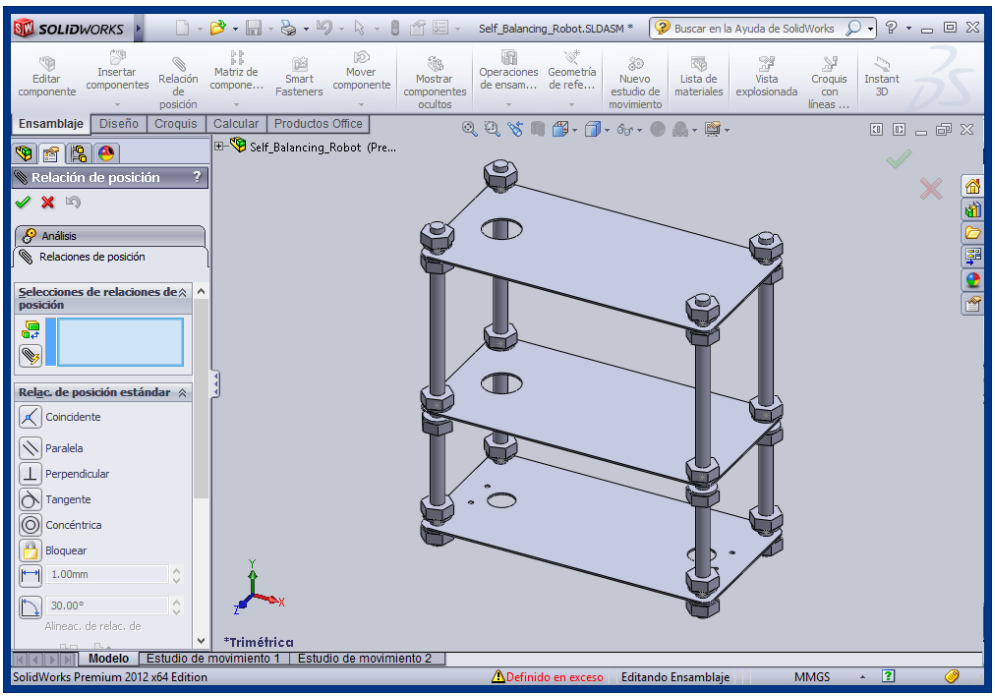


Figura 65: Ensamblaje de las piezas con relación de posición

9. Por último, al estar tan extendidos los softwares de CAD, es posible conseguir componentes muy utilizados que han sido diseñados por otras personas. Por lo tanto, se decidió utilizar diseños de otras personas para los componentes más habituales como son la batería, ruedas, motores, placa Arduino, IMU y driver motor. Los componente se distribuyeron por toda la estructura en el lugar que más o menos se deseaba que estuviesen, quedando como resultado final:

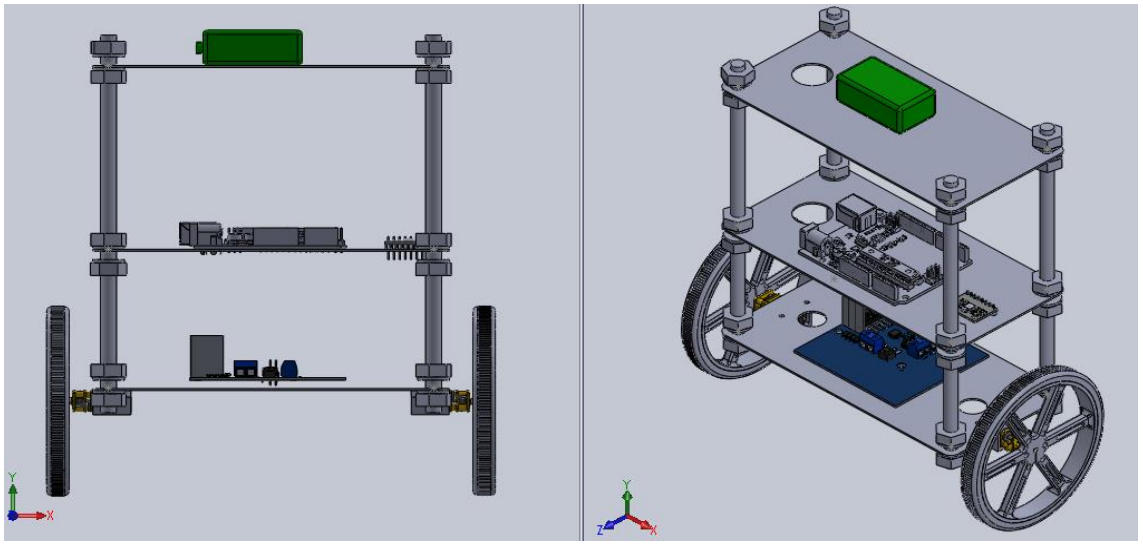


Figura 66: Robot completo SolidWorks

6.3.2 Fabricación de la estructura

Tras el proceso de diseño de la estructura completa mediante SolidWorks, se obtienen los planos de las tres bases que forman la estructura para su posterior fabricación, estos planos viene dados en el apartado correspondiente. Para la fabricación de estas piezas se recurrió al corte con láser, que es una técnica empleada para cortar piezas de chapa caracterizada en que su fuente de energía es un láser que concentra luz en la superficie de trabajo.

Por lo tanto, el material de fabricación elegido fue acero, concretamente chapas finas de 1mm de espesor. Se recurrió a esta técnica ya que se tenía fácil acceso a ella. Además, al ser piezas de muy reducido tamaño no resulta nada caro su encargo y de esta forma las piezas obtenidas tendrían un muy buen acabo y una alta precisión.

Figura 67: Pieza base de chapa de acero, obtenida por corte laser

Para las varillas que actúan de soporte de las bases, se recurrió a una varilla roscada lo suficientemente larga para posteriormente cortarla en cuatro partes iguales. Esta varilla será M6, de métrica 6 para que pase perfectamente por los agujeros de la chapa. También hay que emplear arandelas y tuercas que cumplen la función de mantener fija la estructura y no permitir que se desmonte.

Figura 68: Varilla utilizada junto con tuercas

Una vez que se tuvo las piezas fundamentales de la estructura, se procedió a su respectivo montaje con el fin de poder empezar a programar el algoritmo de control y realizar las pruebas pertinentes que permitieran analizar los problemas y fallos que se nos presentarían con la estructura.

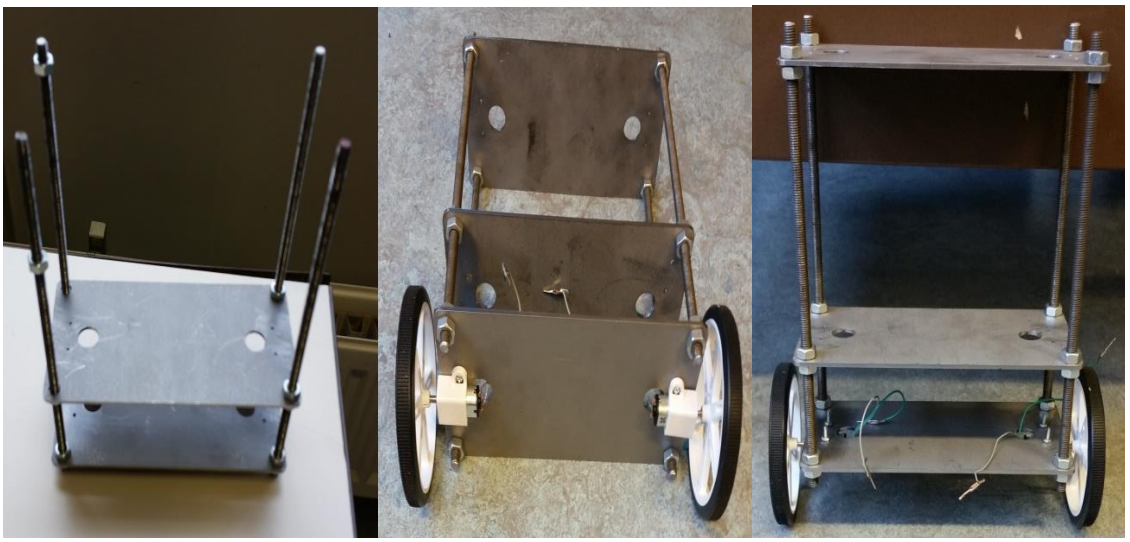


Figura 69: Estructura previa del robot

6.4 Diseño Y Elaboración De Las PCBs

Una de las metas a seguir en el trabajo era el diseño y fabricación de PCBs (Printed Circuit Board), placas de circuito impreso en las que se situarían los componentes electrónicos que forman el robot.

Las PCBs [50] son superficies constituidas por caminos, pistas o buses de material conductor sobre una base no conductora, y sobre estas pistas conductoras se sueldan los distintos componentes electrónicos para formar la placa completa que se quiere fabricar. De esta forma las conexiones entre estos componentes electrónicos no necesitan cables para conectarse unas a otras. Estos circuitos impreso, además de utilizase para conectar eléctricamente a los componentes a través de los caminos conductores, también los sostiene mecánicamente mediante agujeros y soldaduras entre la placa y el dispositivo electrónico. Las pistas son generalmente de cobre mientras que la base se fabrica de resinas de fibra de vidrio reforzada, cerámica, plástico, teflón o polímeros como la baquelita.

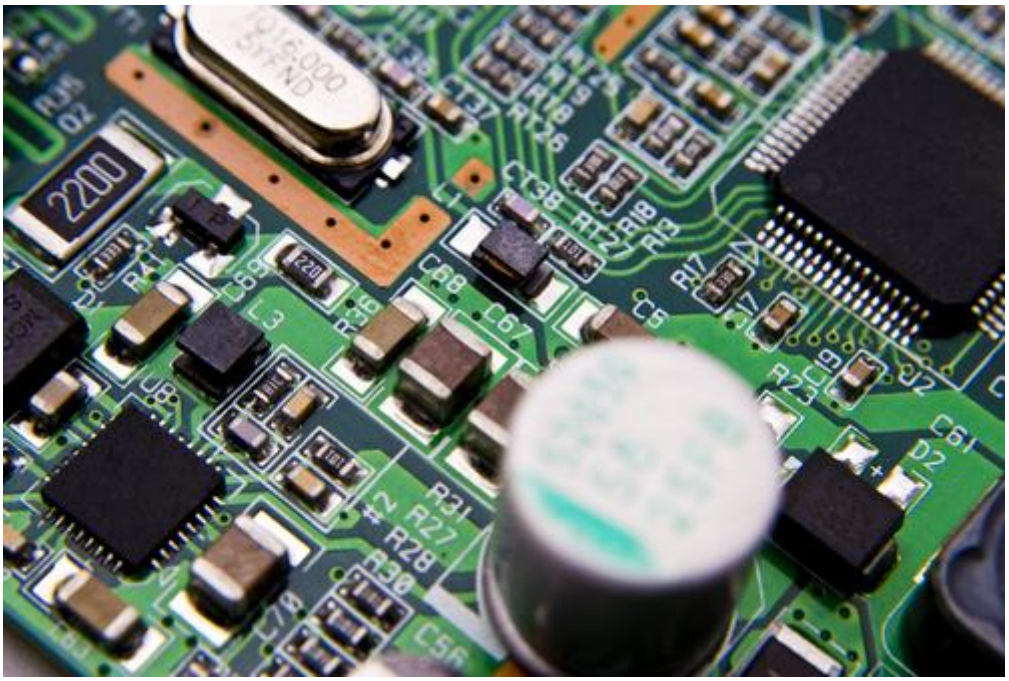


Figura 70: Ejemplo de PCB - fuente:http://cfnewsads.thomasnet.com/images/cmsimage/image/PCB2.jpg

Las PCBs más sencillas corresponden a las que contienen caminos de cobre (tracks) solamente por una de las superficies de la placa. A estas placas se les conoce como PCBs de una capa, o en inglés, 1 Layer PCB. En este trabajo, la placa que contiene al Arduino, los potenciómetros, la IMU y el módulo bluetooth será de este tipo.

Por otro lado, las PCBs más comunes de hoy en día son las de 2 capas o 2 Layer. Sin embargo, dependiendo de la complejidad del diseño del físico del circuito, pueden llegar a fabricarse hasta de 8 o más capas. En este trabajo, la placa que contiene el driver motor será de 2 capas debido a que requiere de conexiones un poco más complejas.

Como ya se ha dicho, se diseñaron y fabricaron dos PCBs distintas, una perteneciente al driver motor y la otra perteneciente a las demás etapas del robot.

Para el diseño se utilizó el software llamado DesignSpark PCB, que fue conocido y utilizado en el transcurso del Grado.

6.4.1 Software DesignSpark PCB

DesignSpark PCB [51] es un software gratis de libre distribución desarrollado por el distribuidor de electrónica RS Components. Este es un tipo de software pertenece a la categoría de los EDA (Electronic Design Automation) que sirven para la automatización de diseño electrónico como puede ser el diseño de placas de circuitos impresos o PCBs.

Figura 71: Logotipo del software DesignSpark PCB – fuente: http://en.wikipedia.org/wiki/DesignSpark_PCB

El software básicamente dispone de tres tipos de archivos:

- Diseño de esquemáticos: se utilizan para elaborar diagramas de circuitos y conexiones. En un proyecto, se combinan todos juntos para formar el diseño completo.

- Diseño de PCB: Los esquemas se traducen en un archivo de diseño de PCB con un Asistente de PCB. Un diseño de PCB puede tener varias iteraciones antes de que se pase a una placa de circuito impreso finalizada para la producción.

- Proyectos: se utilizan para organizar los archivos de diseño. Un proyecto puede tener un número ilimitado de diseños esquemáticos y un archivo de diseño de PCB.

DesignSpark PCB es capaz de producir archivos de salida como Gerber y Excellon. Estos archivos estándar son aceptados por compañías fabricantes de PCB y se utilizan para construir una placa de circuito impreso.

6.4.2 PCB de Arduino y sensado

A continuación se explican los pasos seguidos durante la realización de la placa y los resultados finales obtenidos.

1. Se monta el prototipo en una placa protoboard, con el fin de tener una primera toma de contacto con los componentes, y localizarlos en el espacio. Esto resulta de gran utilidad a la hora de conocer las necesidades que podrían surgir en cuanto a espacio, conexión con otros elementos, resultados inesperados, etc.

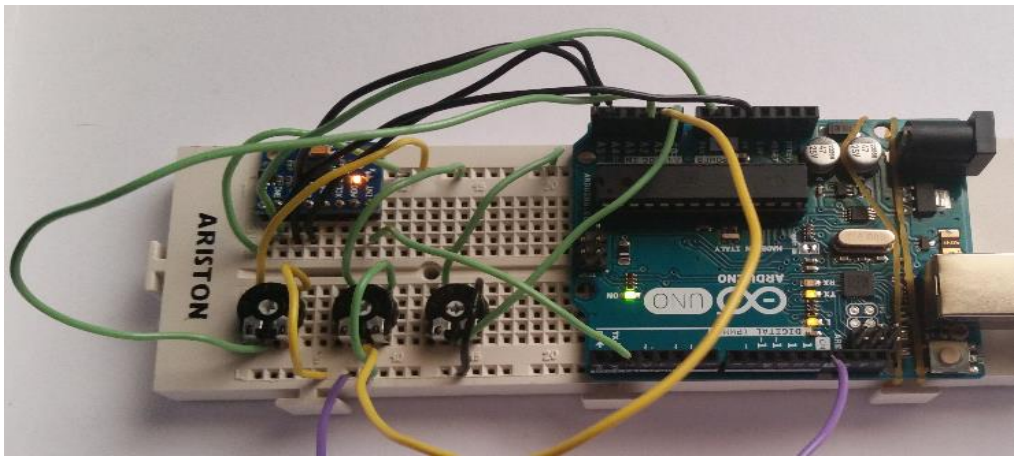


Figura 72: Montaje en protoboard Arduino y sensado

2. Una vez se sabe todos los elementos electrónicos que se emplearán, se pasa a crear el correspondiente archivo esquemático dentro de un archivo proyecto. Entonces, se dibuja el esquema de los circuitos que va a contener la PCB con ayuda del panel de herramientas a la izquierda de la pantalla.

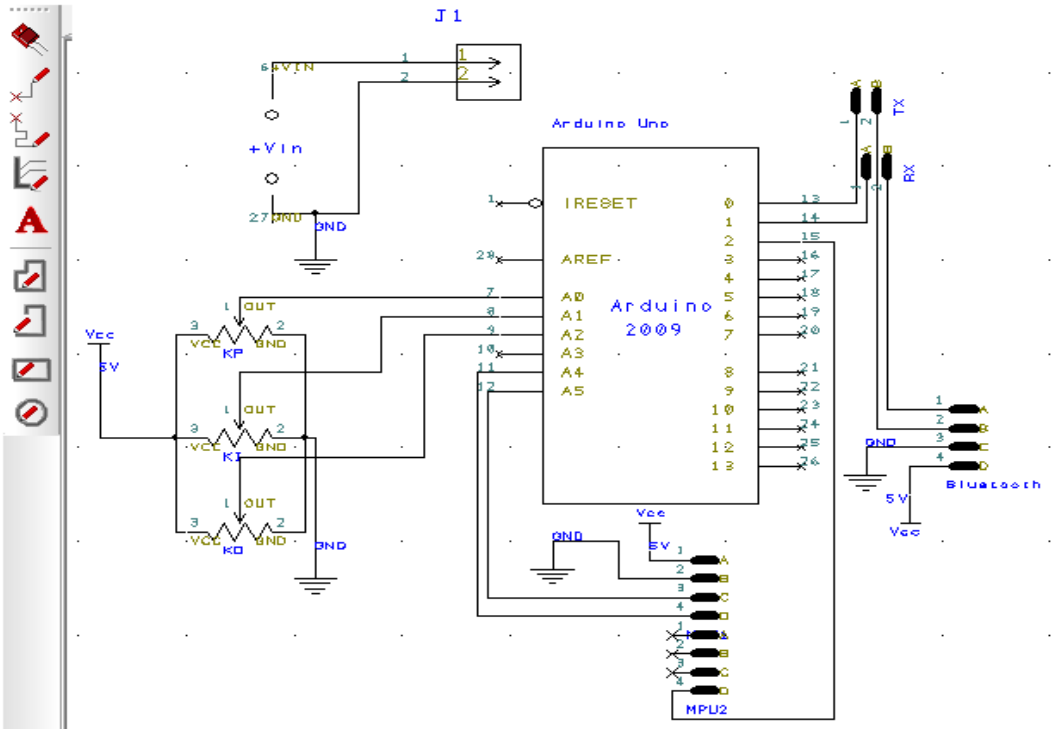


Figura 73: Esquemático de la PCB Arduino y sensado

3. Al terminar el archivo esquemático, el software da la opción de traducirlo a una PCB mediante la pestaña del menú principal → Tools → Translate To PCB.

Para el diseño de la PCB se tienen en consideración los siguientes parámetros:

- Minimizar todo lo posible la longitud de las pistas y evitar que formen 90º: de esta forma se evita caída de tensión en la pista, lo que conllevaría una peor transmisión de señal, estas deben ser lo más cortas posibles y en ningún caso formar 90º.
- Las pistas deben tener una anchura apropiada: 0.8mm para las pistas de señal y para las pistas de potencia 0.9mm. Esto ayuda en el proceso de fabricación de la placa y a la hora del soldado de los componentes.
- Empleo de planos de cobre: al ser la PCB de una cara, solo se puede utilizar un único plano de cobre, que en este caso será un plano de masa.
- Distancias adecuadas: la distancia entre pista y pista o pista y plano de masa, es 0.9mm para facilitar posteriormente la soldadura.
- Posicionamiento adecuado: los componentes deben estar posicionados de una manera lógica, es decir, cercanos a sus conexiones, y ocupando el mínimo espacio posible.

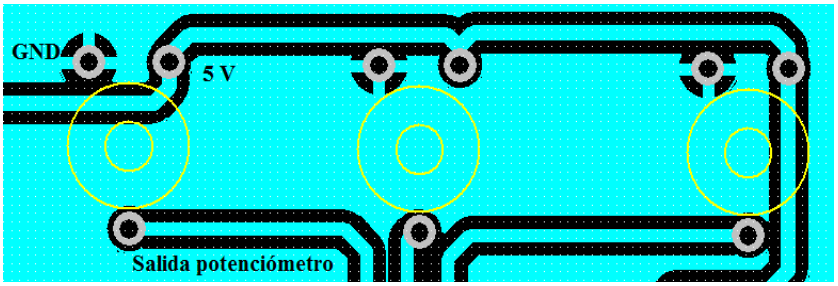


Figura 74: Potenciómetros en PCB

Se emplean tres potenciómetros para el calibrado y cambio manual de las constantes de control (Kp, Ki, Kd). Cada potenciómetro tendrá conexión con masa (GND) y alimentación (5 V) y sus salidas irán conectadas a las entradas analógicas del Arduino para poder emplear las lecturas de los potenciómetros escalándolas en el algoritmo de control.

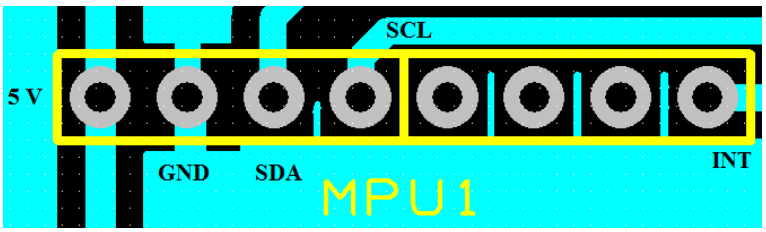


Figura 75: Zocalos MPU en PCB

El MPU irá conectado en un zócalo apropiado con las conexiones necesarias para su uso en DMP. El dispositivo necesita una conexión de alimentación (5V), a masa (GND), conexiones I2C (SDA y SCL) y por último la conexión para la interrupción INT.

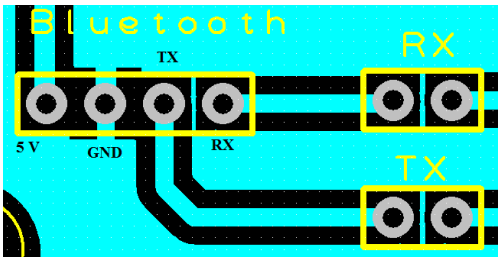


Figura 76: Pads del módulo Bluetooth y jumpers

Para el módulo Bluetooth HC-06 se precisa las conexiones de alimentación (5V y GND) y además las conexiones de la comunicación serie (TX y RX). También es necesario el uso de jumpers que corten la comunicación serie con el Arduino cuando se está programándolo, ya que de lo contrario dará siempre un error.

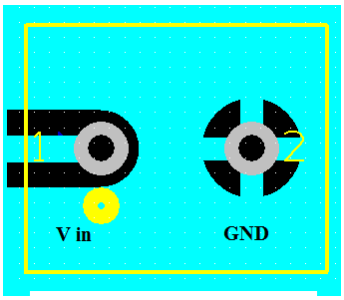


Figura 77: Conector de alimentación de Arduino

Hay también un conector PCB que tiene conexión con el plano de masa (GND) y el pin de entrada de tensión (V in) de Arduino. Este conector se utiliza para conexionar la salida de 5 V del regulador de tensión del driver motor con el Arduino, y de esta forma darle la alimentación necesaria para su funcionamiento.

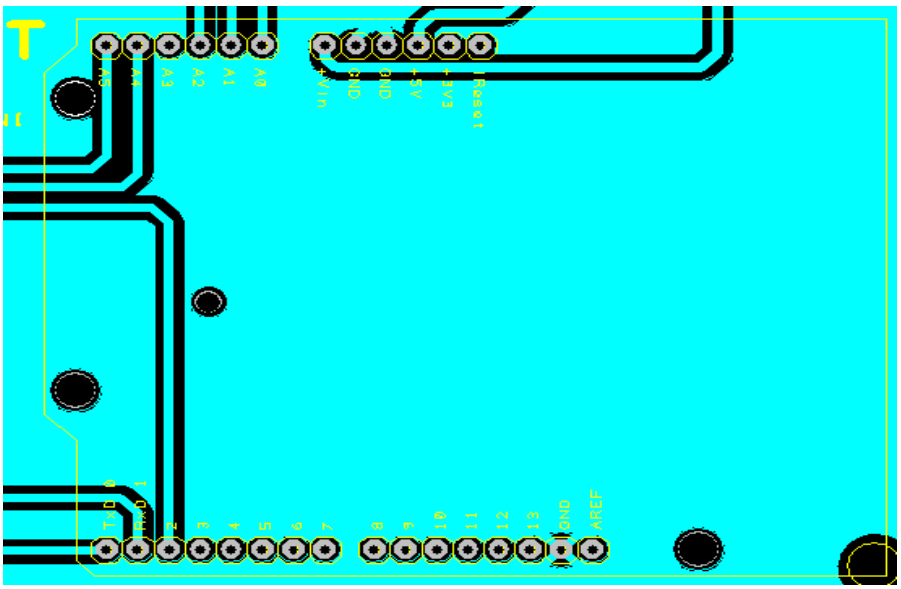


Figura 78: Placa Arduino en el PCB

Por último, está la placa Arduino Uno que va acoplada a unos zócalos y atornillada a la PCB con el fin de mantenerla firme y con buen contacto.

Pines Arduino Uno	Conexiona con…
A0	Salida del potenciómetro KP
A1	Salida del potenciómetro KI
A2	Salida del potenciómetro KD
A4	Pin SDA MPU6050
A5	Pin SCL MPU6050
D0	Jumper TX
D1	Jumper RX
D2	Pin INT MPU6050
Vin	Pin 1 Conector PCB J1
GND	Pin 2 Conector PCB J1
D0	Jumper TX
D2	Pin INT MPU6050

Tabla 7: Conexiones pines Arduino

Como resultado general del diseño se obtuvo la siguiente placa:

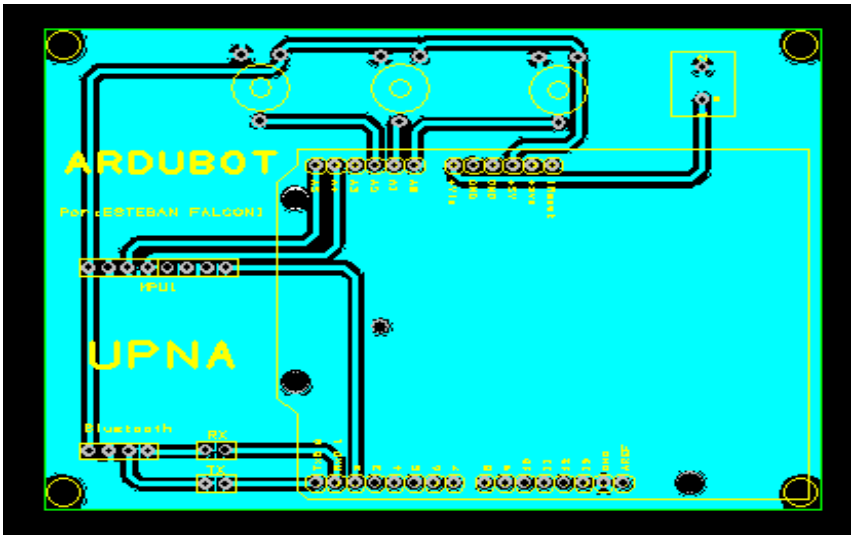


Figura 79: PCB Arduino y sensado en DesignSpark

Hay que destacar que el software de diseño tiene la posibilidad de obtener vistas de 3D de las PCBs diseñadas, siendo de gran ayuda para poder visualizar el posible acabado final y de cómo puede quedar.

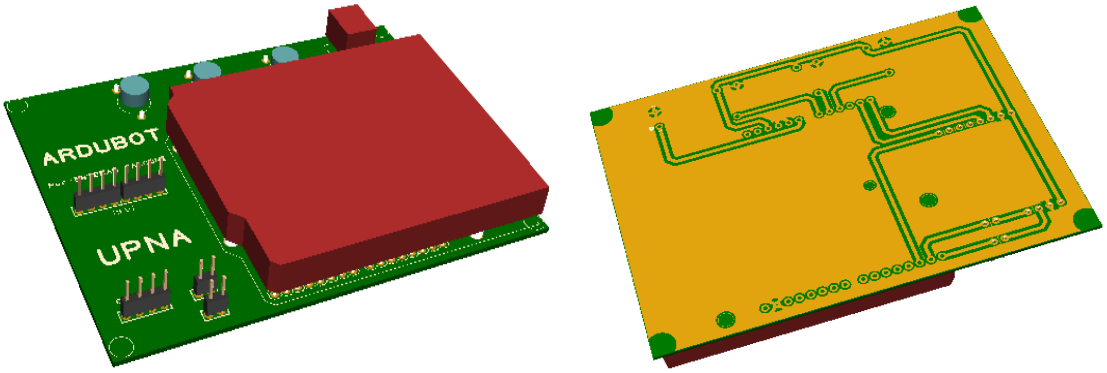


Figura 80: Vistas 3D PCB Arduino y sensado

4. El paso final es la fabricación de la PCB diseñada con ayuda de los archivos de salida que son proporcionados por el propio software, y posteriormente la soldadura de todos los componentes que la forman.

   Los resultados obtenidos fueron los siguientes:

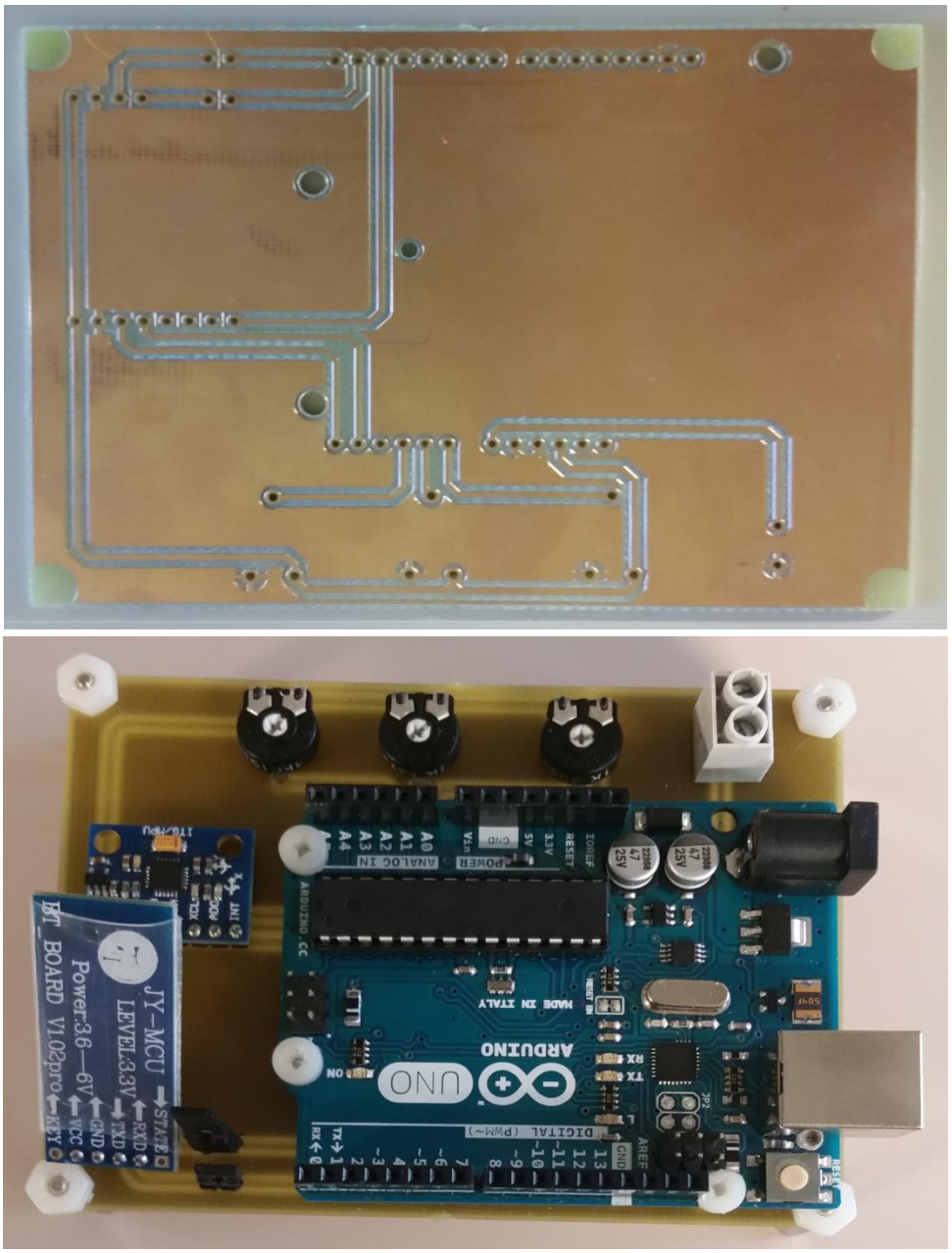


Figura 81: PCB Arduino y sensado final

6.4.3 PCB de Driver motor

A continuación se explican los pasos seguidos durante la realización de la otra placa y los resultados finales obtenidos.

1. Al igual que la anterior PCB se monta el prototipo en una placa protoboard, con el fin de tener una primera toma de contacto con los componentes, y localizarlos en el espacio.

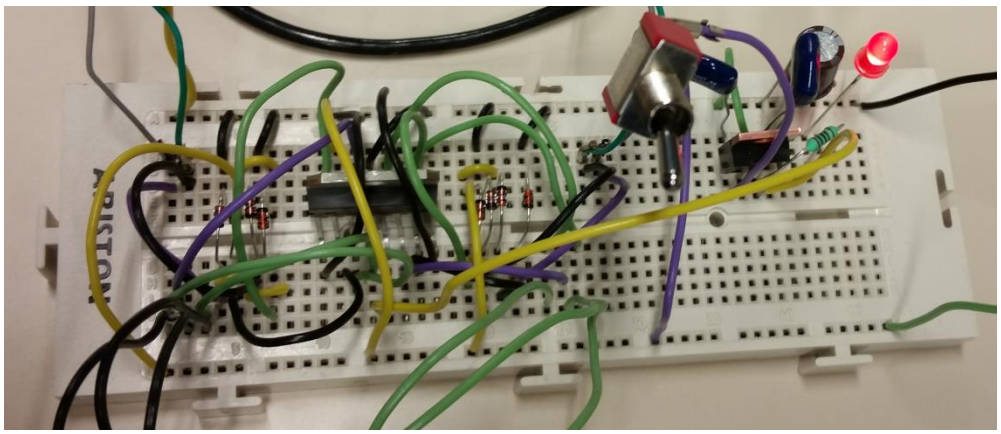


Figura 82: Montaje en protoboard Driver motor

2. A continuación, se crea el correspondiente archivo esquemático dentro de un archivo proyecto y se dibuja el esquema de los circuitos que va a contener la PCB del driver motor.

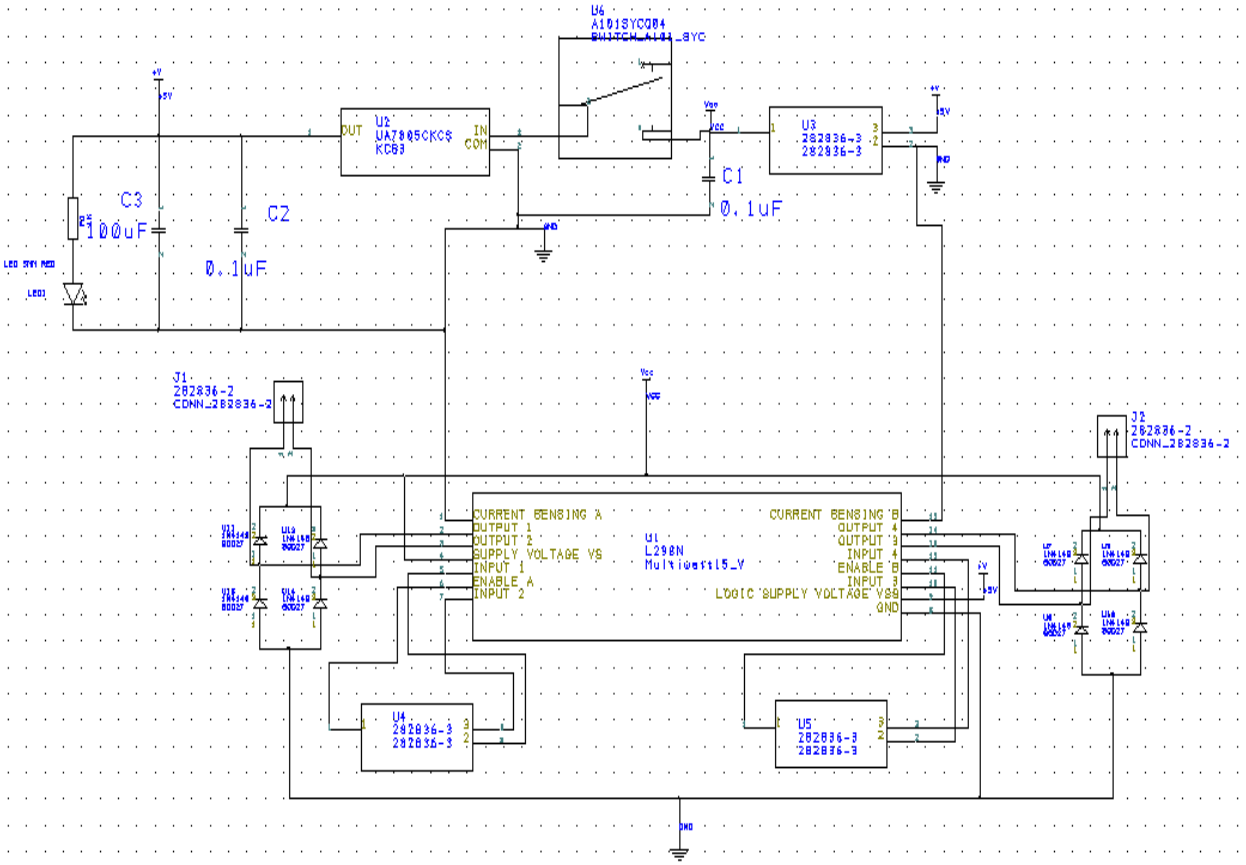


Figura 83: Esquemático de la PCB Driver motor

3. Con el esquemático finalizado, se procede a la traducción de ese esquema a una PCB con la correspondiente herramienta del software DesignSpark.

   Al igual que en la anterior PCB, se utilizan los mismos parámetros añadiendo los siguientes de forma específica:

   - Empleo de planos de cobre: en este caso ya no es de una sola cara, por tanto se puede utilizar dos planos de cobre, que en este caso será un plano de masa (GND) en la cara posterior y otro plano de alimentación (5V) en la cara frontal de la placa.
   - Posicionamiento adecuado: los componentes deben estar posicionados de una manera lógica, es decir, cercanos a sus conexiones, y ocupando el mínimo espacio posible; además en este caso existe la posibilidad de posicionar componentes en ambas caras.
   - Emplear el mínimo número de pista en la cara frontal: sólo se utilizarán pistas en esta cara cuando no se tenga otro remedio de conexionado, para que no se terminen cruzando pistas.

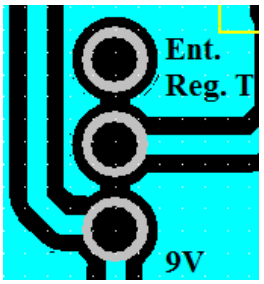


Figura 84: Interruptor general en PCB

Se emplea un interruptor general, que apaga o enciende el driver motor aunque esté conectada la batería. Este interruptor corta el paso de la entrada de alimentación al regulador de tensión, consiguiendo de esta forma que ni el driver, ni la placa Arduino se pongan en funcionamiento.

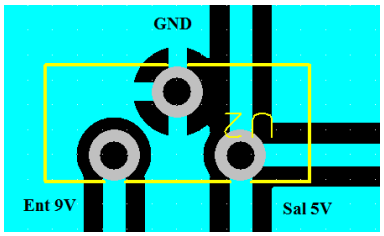


Figura 85: Regulador de tensión en PCB

También se hace uso de un regulador de tensión, que va a ser el encargado de suministrar los 5V de alimentación necesarios para todos los circuitos digitales. Esto es debido a que la batería nos suministra 9V, que sólo son utilizados directamente para los motores.

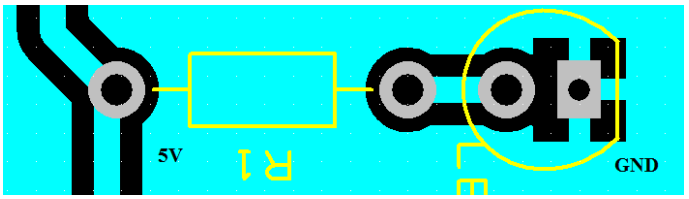


Figura 86: Led indicador en PCB

Para saber si hay o no tensión de alimentación de 5V, se añade a la placa un led de color rojo que al iluminarse indica que efectivamente el regulador de tensión está cumpliendo su trabajo.

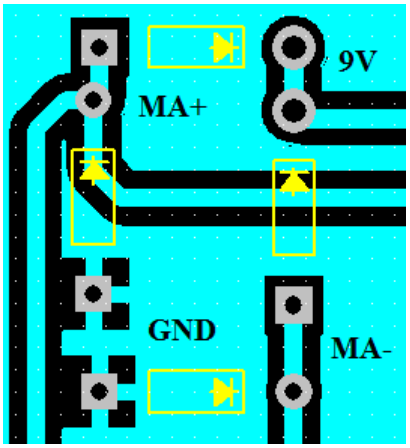


Figura 87: Puente de diodos de protección en PCB

Han sido colocados dos puentes de diodos, uno por cada motor. Estos puentes cumplen la función de protección contra inversión de polaridad y tienen conexión con las salidas de los motores.

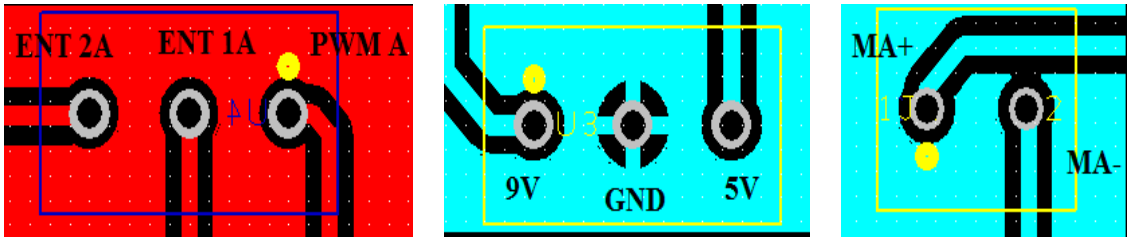


Figura 88: Conectores de 2 y 3 entradas en PCB

Para el conexionado de las salidas a los motores se utilizan conectores de PCB de dos entradas y para el conexionado tanto de la alimentación, como de las entradas de control de los motores se utilizan conectores de 3 entradas.

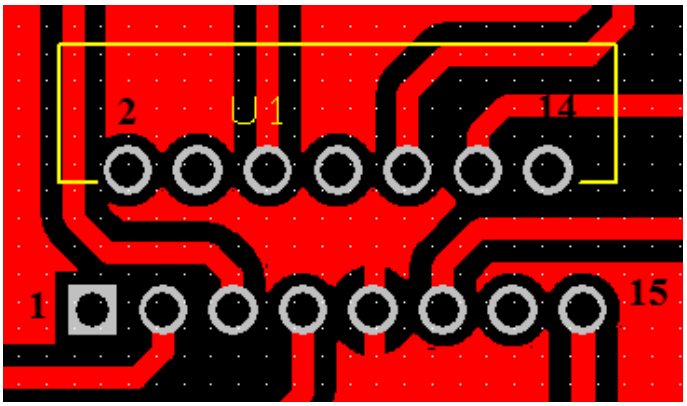


Figura 89: L298N en PCB

Por último, como núcleo principal de la PCB se presenta el encapsulado L298N que es el encargado de realizar las funciones de control de los motores. Al disponer de muchas patillas y ser su conexionado un poco más complejo, se tuvo que emplear pistas por la cara frontal para este dispositivo y así conseguir que no se crucen pistas.

Pines L298N	Conexiona con…	Pines L298N	Conexiona con…
1 y 15	GND	8	GND
2	Salida Motor A+	9	5V
3	Salida Motor A-	10	Entrada 1 Motor B
4	9V	11	PWM Motor B
5	Entrada 1 Motor A	12	Entrada 1 Motor B
6	PWM Motor A	13	Salida Motor B+
7	Entrada 2 Motor A	14	Salida Motor B-

Tabla 8: Conexionado pines L298N

Como resultado general del diseño se obtuvo la siguiente placa de doble cara:

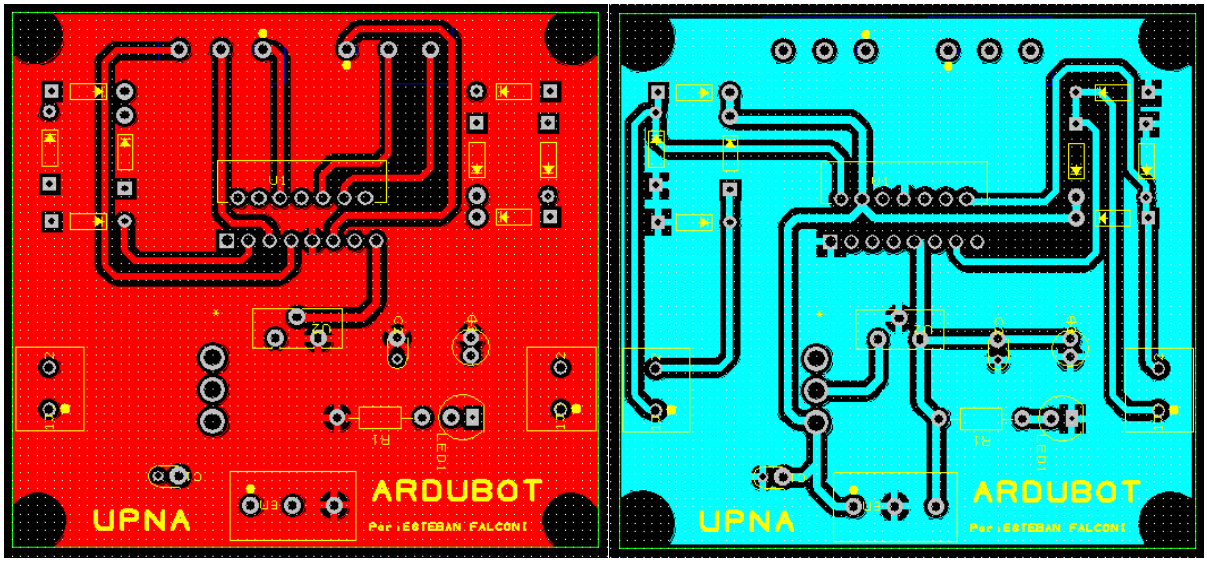


Figura 90: PCB Driver motor, cara frontal y cara posterior

También en este diseño se recurrió a la vista en 3D para hacerse una idea del posible acabado final y de cómo puede quedar.

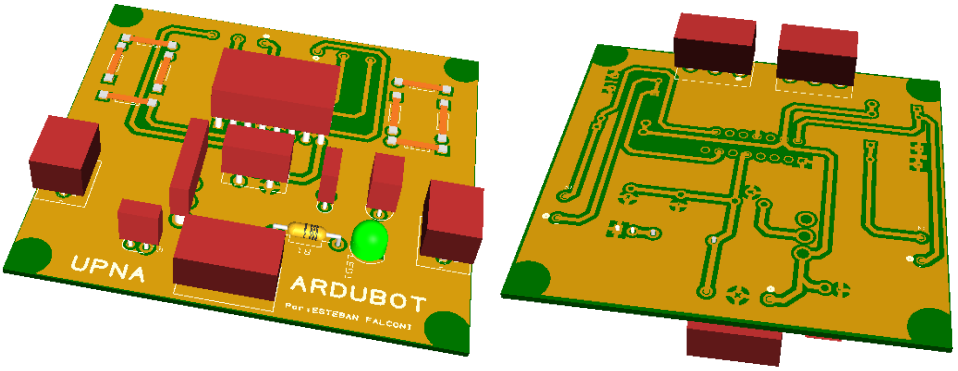


Figura 91: Vistas 3D PCB Driver motor

4. Como paso final se procede a la fabricación de la PCB diseñada, y posteriormente la soldadura de todos los componentes que la forman.

Los resultados obtenidos fueron los siguientes:

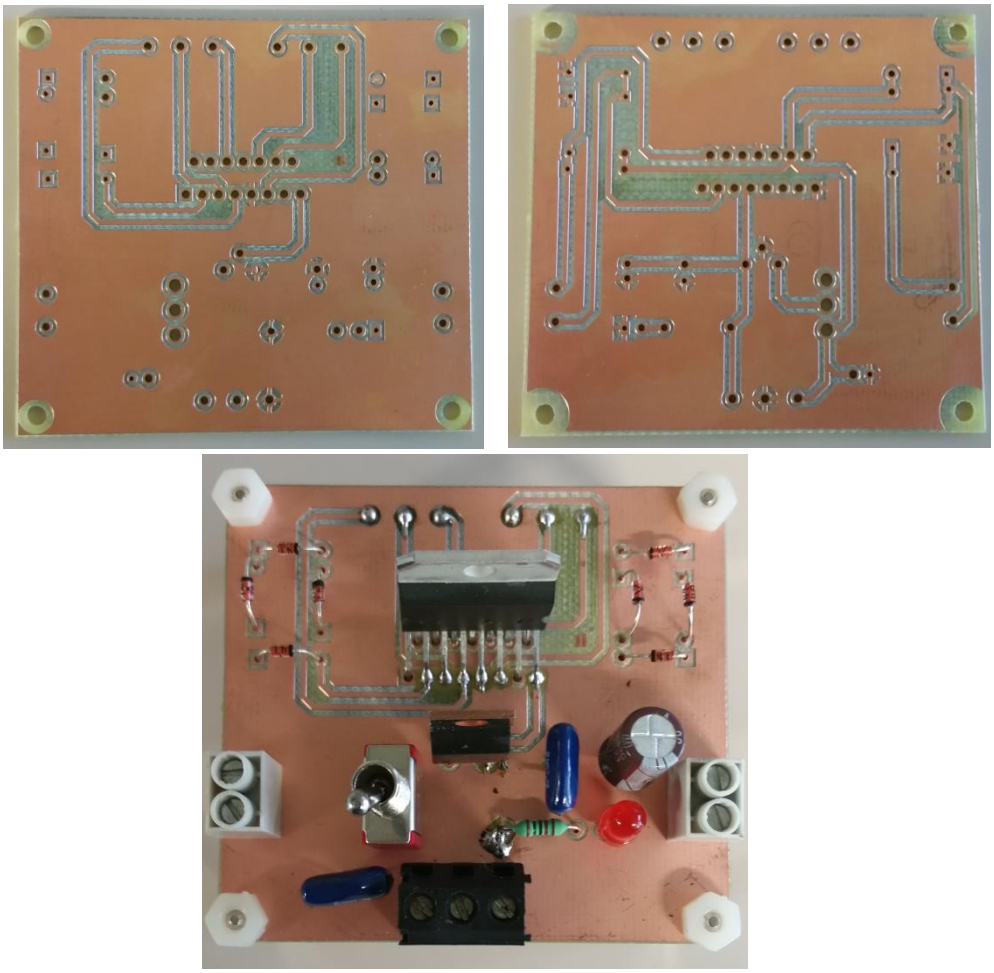


Figura 92: PCB Driver motor final

6.5 Elaboración Del Algoritmo De Control

No basta con implementar un algoritmo de control de un PID como el mostrado en el apartado de la etapa de control, ya que se quedaría con ciertas limitaciones en cuanto a su aplicación a un sistema real como es el caso de este TFG.

Por lo tanto, para conseguir un comportamiento de un PID de nivel más complejo [52], hay que tener en cuenta otros parámetros.

Uno de ellos es que el algoritmo del PID funciona mejor si se ejecuta a intervalos regulares, es decir, incorporando el concepto del tiempo dentro del PID se pueden llegar a simplificar los cálculos que tiene que efectuar el Arduino en cada iteración de su bucle principal.

Los PID básicos, es decir los que son muy simples y no tienen en cuenta las irregularidades del sistema, presentan el inconveniente de estar diseñados para ejecutarse a periodos irregulares. Esto acarrea un comportamiento irregular del PID, debido a que en ocasiones se ejecuta regularmente y otras veces no. Además, hay que realizar operaciones matemáticas extras para calcular los términos de la acción integral y derivativa del PID, ya que ambos son dependientes del tiempo.

Entonces, hay que asegurarse que la función que ejecuta el PID lo hace de una forma regular. Por ello el código del algoritmo, en este caso con el nombre de "miPID", decide si debe hacer cálculos o retornar de la función. Consiguiendo que el PID se ejecute a intervalos regulares, los cálculos correspondientes a la parte integral y derivativa se simplifican.

Para implementar el concepto de tiempo al algoritmo, se añadió al código de programación lo que se muestra en la siguiente figura:

```
unsigned long Ahora = millis();  // Indicará el tiempo que ha pasado desde que la placa empezó a ejecutar el programa
TiempoCambio = (Ahora - TiempoAnterior); //Indicará el tiempo transcurrido desde la última vez que se ejecutó el PID

//Cumple la función de saber cuando hay que ejecutar el PID o retornar de la función
if(TiempoCambio >= TiempoMuestreo)
{
   .....
  //Se guardan tanto tiempo como error para el proximo ciclo de cálculo
  erroranterior = error; //
  TiempoAnterior = Ahora;
}
```

Figura 93: Concepto de tiempo al algoritmo de control

Otro parámetro a tener en cuenta en un PID es el "Derivative Kick", este fenómeno se produce cuando hay cambios bruscos en la señal de referencia.

Este problema queda perfectamente reflejado junto con su solución en la siguiente figura:

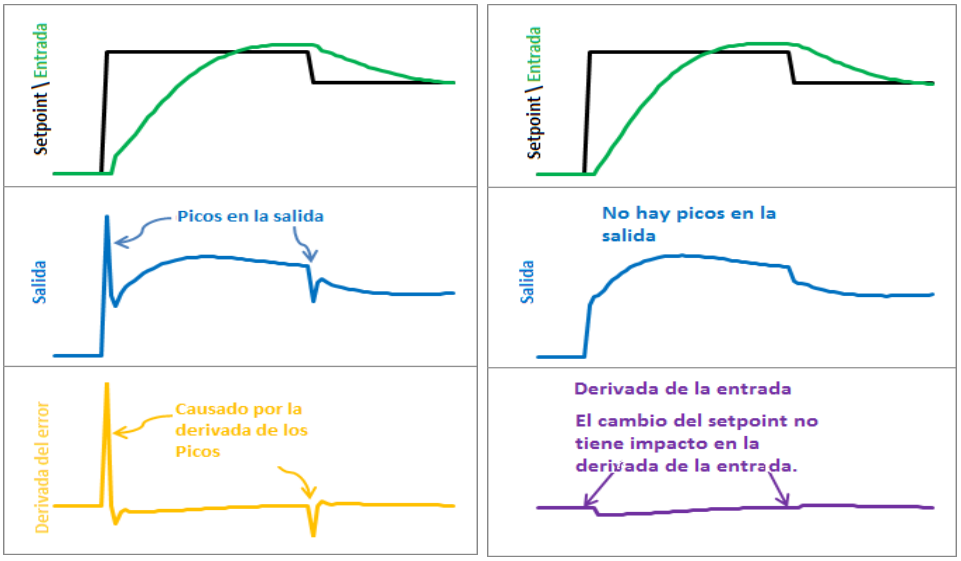


Figura 94: Fenómeno del Derivative Kick

Siendo el error = setpoint - entrada, cualquier cambio en la referencia causa un cambio instantáneo en el error y la derivada de este cambio tiende a infinito. Esto produce un sobrepasamiento muy alto en la salida o acción de control, cosa que no se quiere de ninguna manera para el sistema.

En el caso del robot estos cambios de referencia se darán al intentar desplazar el robot hacia delante o atrás.

El fenómeno se puede solucionar de una manera muy sencilla empleando la fórmula de la derivada del error:

$$\frac{d\text{Error}}{dt} = \frac{d\text{Setpoint}}{dt} - \frac{d\text{Input}}{dt}$$ → cuando Setpoint es constante, su derivada se hace cero, quedando → $$\frac{d\text{Error}}{dt} = -\frac{d\text{Input}}{dt}$$

Por ello a la hora de calcular la acción de control, en lugar de sumar la acción derivativa "Kd * derivada error", se le resta "Kd * deriva entrada".

Las modificaciones pertinentes que se necesitan hacer en el algoritmo serán presentas en la figura 95, donde se cambiará levemente la acción derivativa con el objetivo de eliminar este fenómeno.

Se cambia:

```
deriv_error = (error - errAnterior) / TiempoCambio; //Cálculo de la derivada de error
Salida = kp * error + ki * errSum + kd * deriv_error; //Cálculo de la acción de control
```

Por:

```
deriv_error = (angulomedido - anguloanterior) / TiempoCambio;  //Cálculo de la derivada de error
Salida = kp * error + ki * integ_error - kd * deriv_error;  //Cálculo de la acción de control
```

Figura 95: Eliminación del Derivative Kick

Por último, existe también un parámetro más a tener en cuenta, y este es el fenómeno Windup que se muestra al arrancar el sistema o en cualquier otra situación, donde aparece un error muy grande durante un tiempo prolongado. Como es lógico, la acción integral aumentará para reducir el error.

Pero si el actuador está limitado, como en este caso que se encuentra entre 0 y 5V, se saturará, pero la acción integral seguirá creciendo. Cuando el error se reduce, la acción integral también comenzará a reducirse, pero desde un valor muy alto, llevando mucho tiempo hasta que logre la estabilidad. Por lo tanto, el problema se da en forma de retrasos y con valores de acción de control muy por encima del límite.

Este problema queda perfectamente reflejado junto con su solución en la siguiente figura:

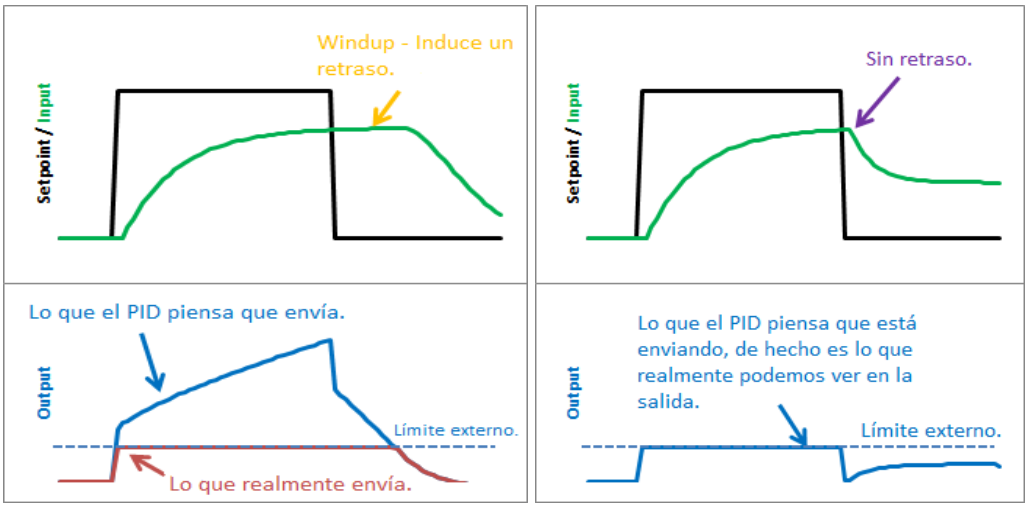


Figura 96: Fenómeno WindUp

Una forma de corregir el WindUp, es aplicar al PID los límites pertinentes tanto a la acción integral como a la acción de control.

Hay que acotar ambas acciones, porque aunque se pongan límites al valor que puede tomar la salida, el término integral seguirá creciendo, introduciendo errores en la salida.

Las modificaciones que se han tenido que realizar en el algoritmo serán presentas en la figura 97:

```
Accion_integ += (ki*(error * TiempoCambio)); //Cálculo separado de la acción integral
//Limites asignados a la acción integral [255,-255]
if(Accion_integ > 255)
    Accion_integ = 255;
else if(Accion_integ < -255)
    Accion_integ = -255;

//Cálculo de la acción de control
Salida = (kp * error) + Accion_integ - (kd * deriv_error);
//Limites asignados a la acción de control [255,-255]
if(Salida > 255)
    Salida = 255;
else if(Salida < -255)
    Salida = -255;
```

Figura 97: Solución al fenómeno WindUp

Con todos estos parámetros que se han tenido en cuenta, se consigue un algoritmo de control PID bastante robusto y que es capaz de responder muy bien ante los problemas que pueda presentar el sistema.

El código final del algoritmo se puede visualizar en el correspondiente Anexo 3.

6.6 Montaje Final Del Robot Y Pruebas Pertinentes

Para la estructura robótica fabricada, se optó por un acabado en pintura, consiguiendo de esta forma dar una buena estética al robot y además conseguir aislar eléctricamente la estructura de los componentes que se van a colocar en ella.

Con el objetivo de añadir las PCBs fabricadas a la estructura del robot, se ha elegido separadores hexagonales de forma que la PCB Arduino y sensado quedará adherida a la base de en medio y la PCB Driver motor se situará en la base de abajo, donde también van acoplados los motores DC. En la base de arriba por tanto se colocará la batería de 9V recargable.

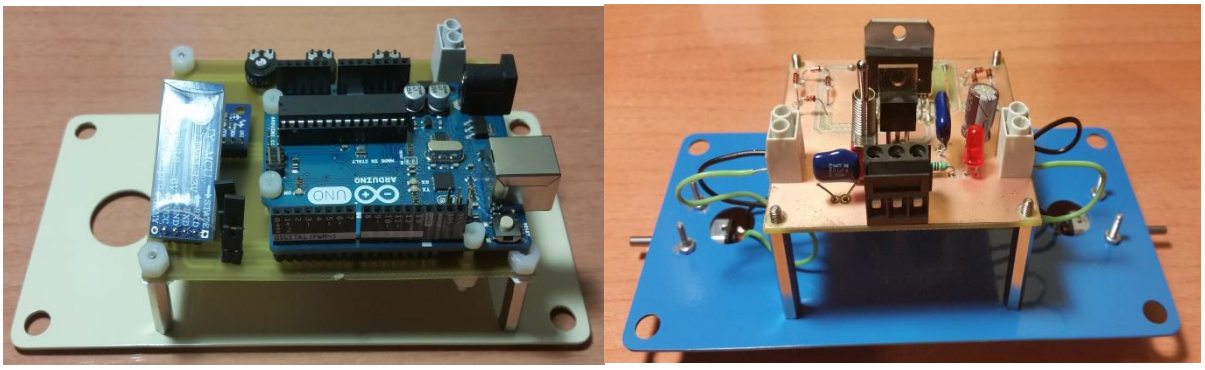


Figura 98: PCBs adheridas a la bases de la estructura

Con todo esto, el aspecto final del robot queda de la siguiente manera:

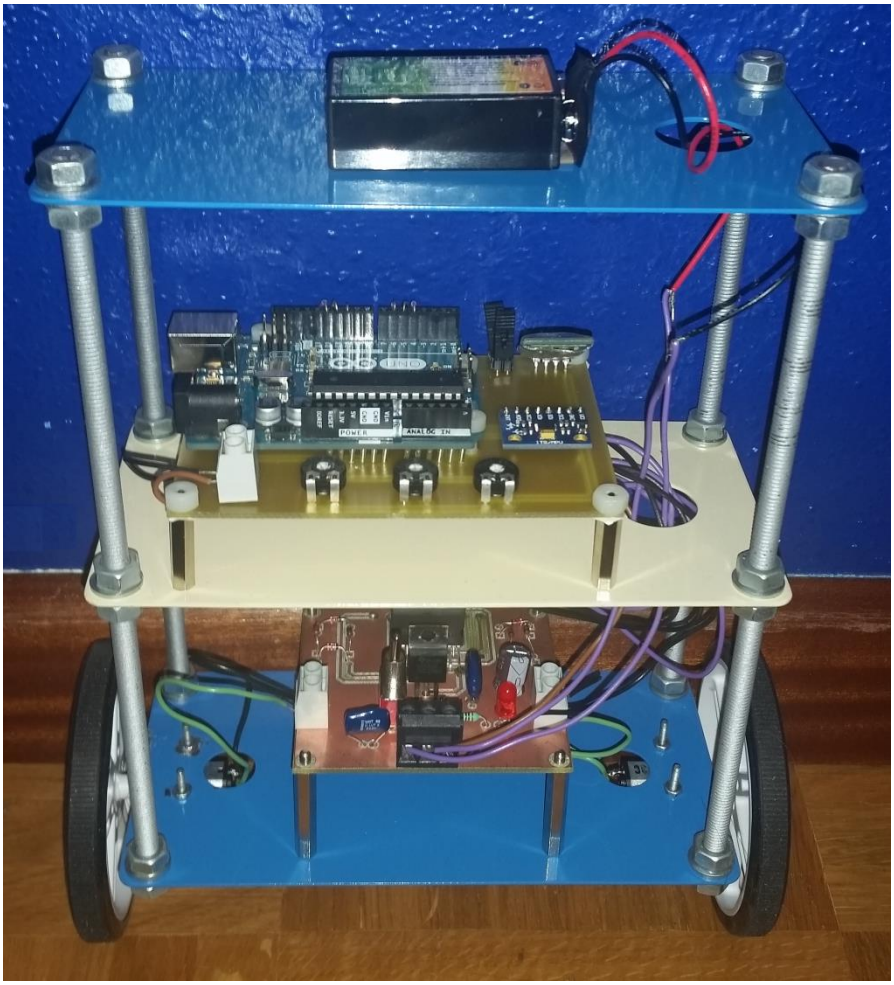


Figura 99: Montaje final robot

Para la sintonización de las constantes de control (Kp, Ki, Kd) se eligió el método explicado a continuación. Este método es utilizado cuando se desconoce el modelo del sistema y consiste en realizar modificaciones sucesivas de las constantes de control hasta conseguir que el sistema se estabilice en torno a la referencia deseada. En este caso el valor de estabilización es un ángulo 0 de inclinación con respecto de la posición vertical.

Se inicia la sintonización con las constantes integral (Ki) y derivativa (Kd) a cero, luego se va dando valores a la constante de proporcionalidad (Kp) hasta lograr una oscilación con la menor amplitud posible en torno al punto de ajuste, y a partir de ahí, se regulan los valores de Ki y Kd en forma iterativa hasta lograr la exactitud deseada.

Si es necesario, se varía también Kp, en función de los aportes de la acción integral y derivativa. La mayor dificultad de este método reside en establecer parámetros iniciales, a partir de los cuales se pueda ajustar el controlador.

Ahora se detalla cada uno de los pasos:

1. Se va a dar valores a Kp para conseguir el óptimo, con unas oscilaciones de menor amplitud en torno al ángulo 0 deseado. A continuación se presentan unas gráficas a modo de comparativa:

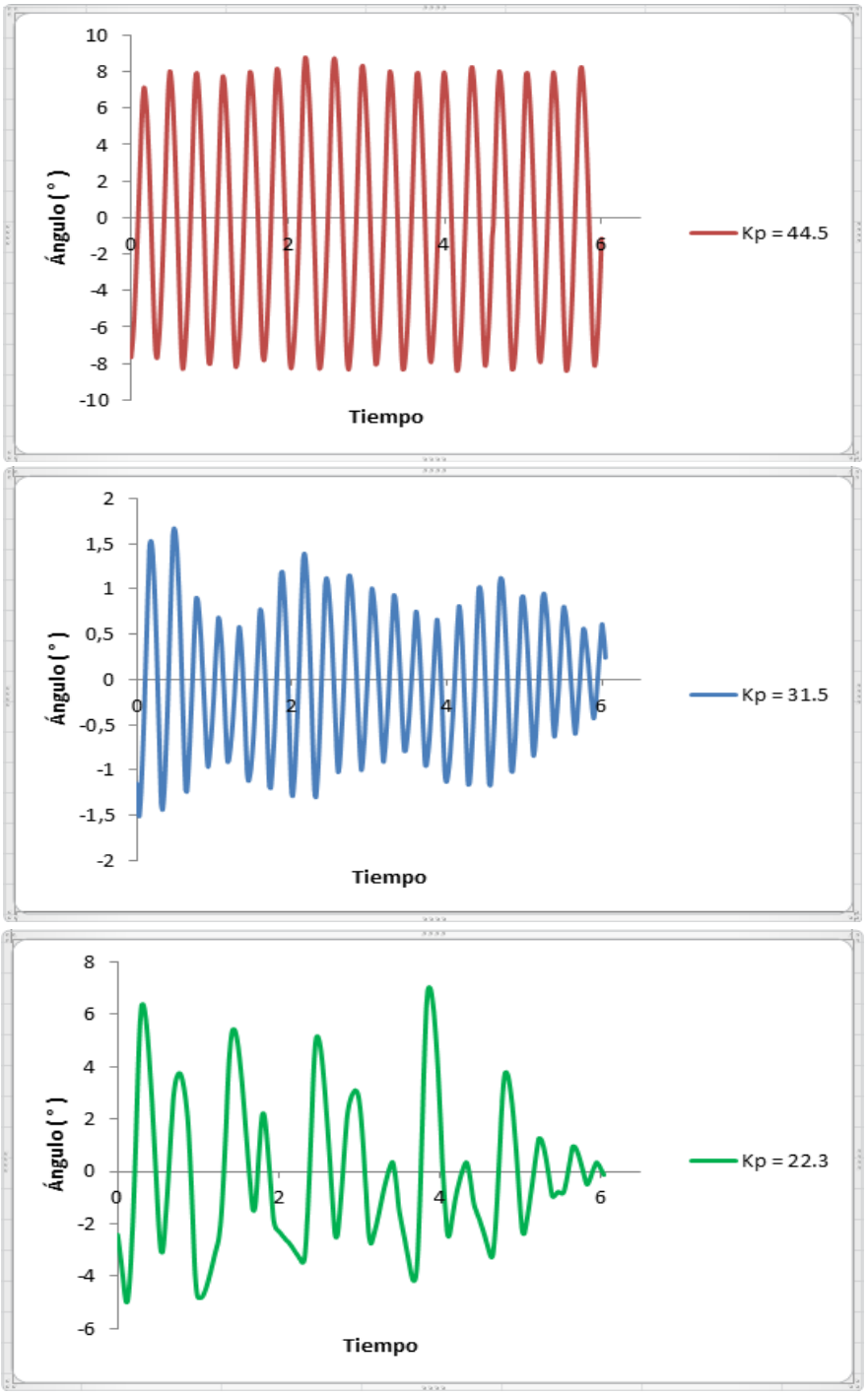


Figura 100: Comparativa de valores Kp

Como conclusión se saca que el valor de Kp tiene que estar en torno a los 31,5 ya que para valores más altos como puede ser el caso de 44,5 las oscilaciones son más fuertes, y para valores bajos como 22,3 el robot no consigue la fuerza necesaria para mantener el equilibrio empezando a tambalearse, es decir, se queda corta la acción proporcional.

2. Una vez obtenido el valor Kp se procede a probar con los valores de Ki y de Kd obteniendo los siguientes resultados:

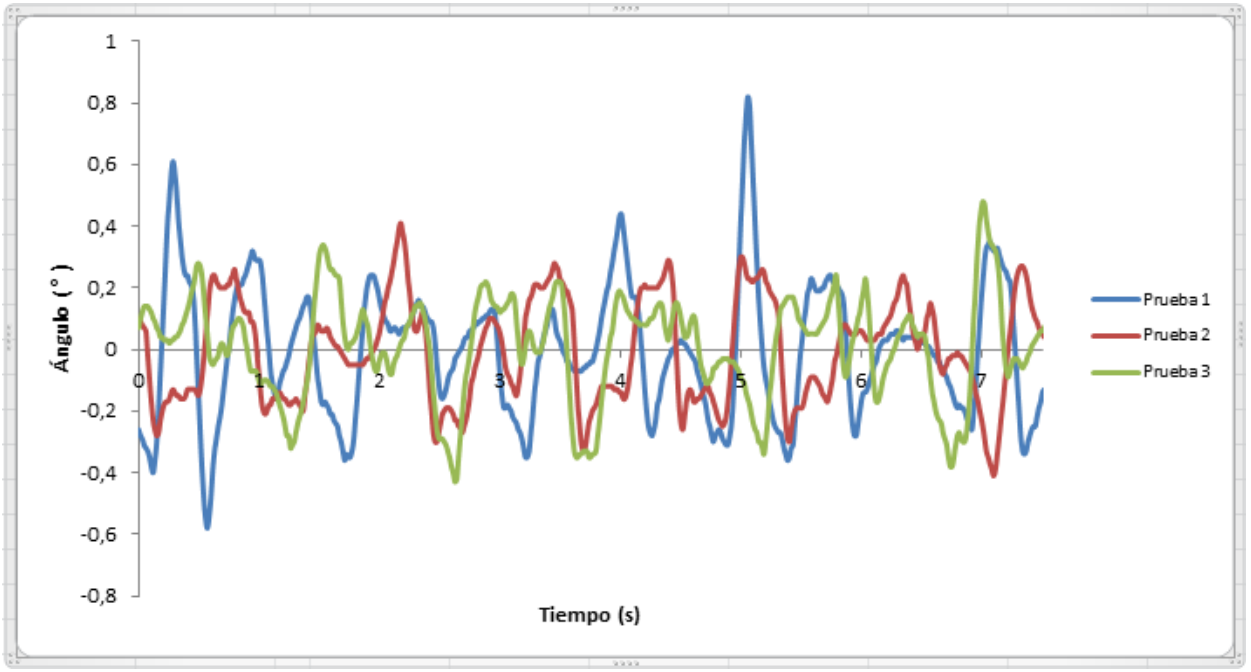


Figura 101: Gráfica comparativa de sintonización

Se realizaron tres pruebas con distintas combinaciones de Ki y Kd:

Prueba	Kp	Ki	Kd
1	31,5	81	11
2	31,5	86	15
3	31,5	100,5	18

Tabla 9: Valores Kp, Ki, Kd

Como se aprecia en la figura 101, los valores de la prueba 1 son los que peor resultados presenta, ya que es la que más se distancia de la referencia con picos de hasta 0,8 º. Entre la prueba 2 y 3, se puede decir que se asemejan mucho y distan mucho menos de la referencia de ángulo 0, con picos positivos y negativos de hasta 0,4. Al ser ambas configuraciones perfectamente válidas para el robot, se procede a comparar su reacción ante posibles perturbaciones que se le puedan aplicar al robot. Estas perturbaciones serán pequeños empujones aplicados al robot y así visualizar si consigue estabilizarse nuevamente y en cuento tiempo lo hace.

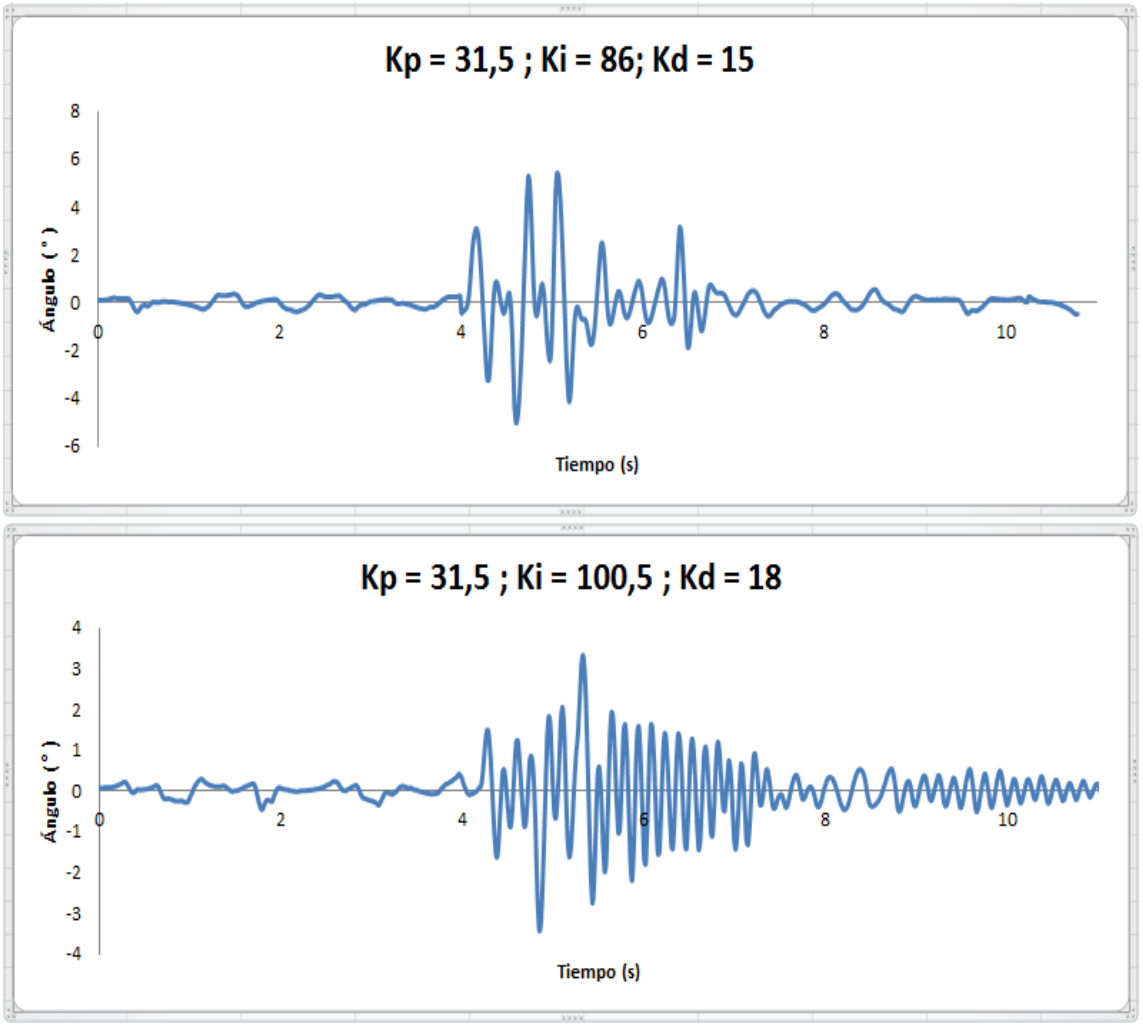


Figura 102: Prueba ante perturbaciones en el sistema

Gracias a la prueba sobre cómo reaccionan ambas configuraciones ante posibles perturbaciones se pudo decidirse por una de ellas.

A las dos configuraciones se les aplicó más o menos a los 4 segundos un empujón, como se observa, en los dos casos el sistema reacciona e intenta volver a seguir la referencia. En el caso de la prueba 2, lo consigue mucho más rápido y de mejor manera que en el caso de la prueba 3. Por tanto, se toma como conclusión que la configuración de la prueba 2 es mucho más fiable ante perturbaciones en nuestro sistema. Y por ello serán elegidos estos valores de Kp, Ki, Kd para la configuración final del robot.

6.7 Uso Del Módulo Bluetooth

Para empezar a utilizar el módulo bluetooth con Arduino, se realiza el conexionado mostrado a continuación:

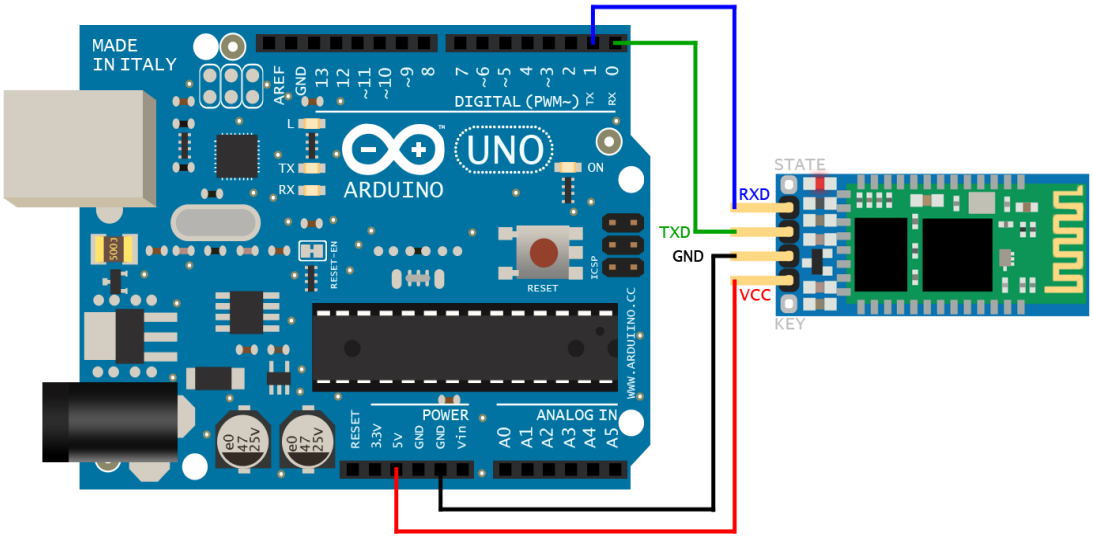


Figura 103: Conexión Arduino Uno con módulo bluetooth HC-06

Este módulo presenta un conexionado bastante sencillo con Arduino. Solamente se precisa darle alimentación, que puede ser de 3,3 V o 5V, y conectar los pines de transmisión y recepción serial (TX y RX). Hay que destacar que en este caso los pines se deben conectar cruzados TX HC-06 → RX de Arduino y RX HC-06 → TX de Arduino.

Sin embargo, los pines 0 y 1 se utilizan en la comunicación serie de Arduino con el PC a través del USB y por tanto, si se usan para comunicar con el módulo HC-06, se pierde la conexión con el PC. Como solución a esto, se incorporan en el diseño de la PCB donde va a ir el módulo dos jumpers, uno por cada pin de transmisión y recepción. De esta sencilla manera se puede cortar o volver a realizar la conexión con el módulo cuando así se precise.

Para poder configurar el HC-06 es necesario que no esté emparejado ni siendo usado por ningún dispositivo, es decir, cuando hay una luz roja parpadeando en el módulo. Para empezar a configurarlo es necesario conectarlo a la PC y usar un código sketch de Arduino, donde la configuración viene dada por comandos AT.

El módulo HC-06 acepta un set muy básico de comandos AT que permite pocas configuraciones, los comandos que soporta son:

- Prueba de funcionamiento:

  Enviar: AT

  Recibe: OK

- Configurar de los baudios:

  Enviar: AT+BAUD<Numero>

  El parámetro número es un caracter hexadecimal de '1' a 'c' que corresponden a los siguientes ratios: 1=1200, 2=2400, 3=4800, 4=9600, 5=19200, 6=38400, 7=57600, 8=115200, 9=230400, A=460800, B=921600, C=1382400

  Recibe: OK<baudrate>

- Configurar el Nombre de dispositivo Bluetooth:

  Enviar: AT+NAME<Nombre>

  Recibe: OKsetname

- Configurar el código PIN de emparejamiento:

  Enviar: AT+PIN<pin de 4 dígitos>

  Recibe: OK<pin de 4 dígitos>

- Obtener la versión del firmware:

  Enviar: AT+VERSION

  Recibe: Linvor1.8

El código empleado para la configuración inicial del módulo que se ha empleado en el robot se presenta en el Anexo 4.

Una vez configurado el módulo, el paso siguiente es lograr comunicar Arduino con un App-Android. Con esta comunicación se puede conseguir desde encender y apagar leds hasta controlar y monitorizar un robot de manera remota, como es el caso.

Hay multitud de apps-android con acceso totalmente gratuito, ya que Andriod es una plataforma abierta. Estas apps se pueden descargar fácilmente desde el propio Smartphone accediendo a "Google Play Store" o como alternativa desde la página web oficial, https://play.google.com/store.

Entre todas las opciones que cumplen con los objetivos de la comunicación, se destacan las siguientes:

- **BlueTerm**: es una app muy básica desde la que no se puede mandar ninguna orden al robot, pero con ella se consigue monitorizar las variables de interés como si del propio monitor serial del software Arduino se tratase.
- **ArduinoRC**: es una app propiamente diseñada para ejercer un radiocontrol en un dispositivo basado en Arduino, aunque presenta el inconveniente de que solo se podría mandar órdenes al robot.

- **BluetoothRC**: otra app con la que se puede, tanto mandar ordenes al robot, como poder visualizar sus variables. Tiene el inconveniente de ser muy poco configurable y no deja hacer ninguna variación si así se precisara.
- **Arduino BT Mando**: con esta app solo se puede mandar órdenes al robot, aunque es bastante limitado y muy poco configurable.
- **Bluetooth Serial Controller**: es una de las apps más completas, con las prestaciones de dejar controlar el robot y de monitorizar sus variables. Esta app, a diferencia de las demás, permite una alta configuración.

Tras analizar las opciones detenidamente, se optó por Bluetooth Serial Controller, ya que era la que más prestaciones presenta y desde ella se puede tanto controlar al robot como monitorizar sus variables. Además, esta app es altamente configurable, permitiendo añadir o quitar acciones que se precisen e incluso tiene un modo de envío automático de comandos.

El modo en el que se va intentar controlar al robot consiste básicamente en enviar comandos desde el dispositivo maestro. En este caso será el Smartphone el que logrará enviar comandos a la placa Arduino. Una vez la placa reciba el comando, mediante programación se decidirá qué acción ejecutar.

Por ejemplo, para el que el robot vaya hacia delante, en la app se presionaría un botón configurado con el nombre "avance". Al presionar dicho botón, se envía el comando previamente configurado como "A". En la programación del Arduino estará definido que si la placa recibe el comando "A", procederá a incrementar el ángulo de referencia. Este incremento provocaría que el robot se inclinase hacia adelante consiguiendo de esta manera su desplazamiento en la dirección deseada.

Por tanto se configuró la citada app de forma que disponga de 4 botones o teclas, Avance, Retroceso, Giro Derecha (Dcha), Giro Izquierda (Izda). Cada uno de estos botones enviarán los comandos "A", "R", "D", "I" respectivamente los cuales ejecutarán las acciones programadas. Además, se tendrá opción de enviar mensajes a la placa y recibir también mensajes de la misma.

El aspecto de cómo queda configurada la app se puede visualizar en la figura 105.

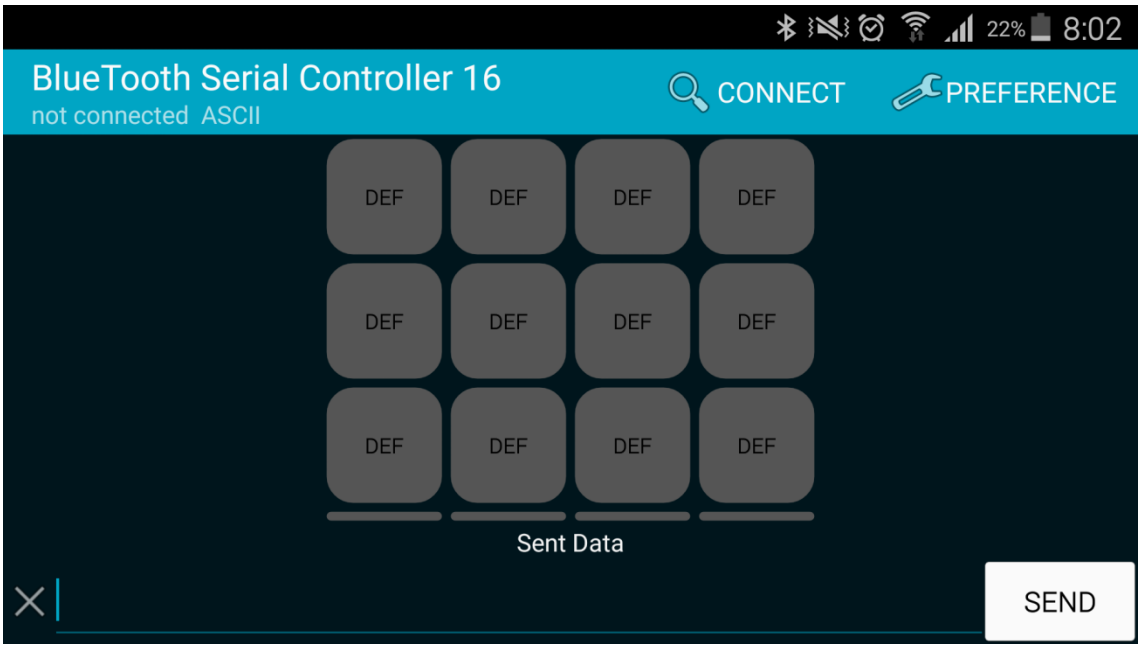


Figura 104: Aspecto de la app sin ser modificada

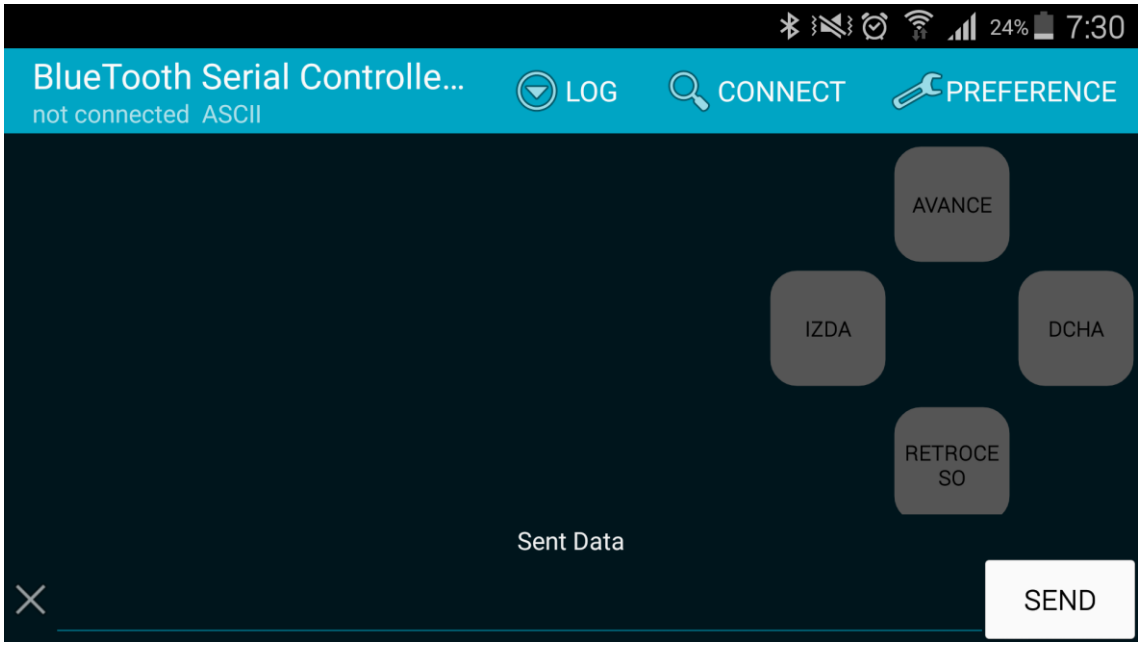


Figura 105: Pantallazo del aspecto final de la app

7 PRESUPUESTO

Coste de fabricación de un sólo robot prototipo final (este primer presupuesto incluye únicamente el coste de los materiales empleados):

Apartado	Componente	Precio Unidad	Cantidad	Precio Total
Sensor	IMU-MPU 6050	7,16 €	1	7,16 €
Control	Arduino Uno	17,00 €	1	17,00 €
	Potenciómetro	0,26 €	3	0,78 €
Potencia	L298N	2,90 €	1	2,90 €
	Diodos 1N4148	0,03 €	8	0,24 €
	Motor DC 50:1	13,90 €	2	27,80 €
	Pareja de ruedas	7,90 €	1	7,90 €
	Soporte motor	4,20 €	1	4,20 €
Alimentación	Pila recargable 9V	6,05 €	1	6,05 €
	Regulador de tensión	1,60 €	1	1,60 €
	Conector Pila 9V	0,46 €	1	0,46 €
PCB	PCB una capa	20,00 €	1	20,00 €
	PCB doble capa	15,00 €	1	15,00 €
Bluetooth	Módulo HC-06	6,50 €	1	6,50 €
Estructura	Base	3,77 €	3	11,31 €
	Varilla	0,60 €	1	0,60 €
	Tuercas(25 Uds)	1,19 €	1	1,19 €
	Arandelas (50 Uds)	1,00 €	1	1,00 €
	Pintura	5,79 €	1	5,79 €
Otros	Led rojo	0,04 €	1	0,04 €
	Resistencia 1K	0,03 €	1	0,03 €
	Condensador cerámico de 0.1uF	0,08 €	2	0,16 €
	Condensador electrolítico de 100uF	0,03 €	1	0,03 €
	Conector PCB 2 entradas	0,25 €	3	0,75 €
	Conector PCB 3 entradas	0,44 €	3	1,32 €
	Interruptor PCB	1,00 €	1	1,00 €
	Jumper	0,03 €	2	0,07 €
	Soporte hexagonal	0,28 €	8	2,23 €
Total				**143,10 €**

Tabla 10: Presupuesto prototipo robot final

El presupuesto de un sólo robot prototipo final, asciende al precio de CIENTO CUARENTA Y TRES EUROS CON DIEZ CÉNTIMOS.

Probablemente el coste total se reduciría considerablemente con una fabricación en serie de varias decenas o cientos de robots ya que los precios de los componentes al por mayor son bastante menores.

Para el presupuesto total final, hay que tener en cuenta la mano de obra necesaria para montaje, pruebas, diseño, puesta a punto, etc.

Para determinar el tiempo de mano de obra necesario, se consideró que el Trabajo Fin de Grado tiene estipulado el valor de 12 ECTS dentro del Grado, y esto más o menos equivale a dos meses de trabajo con sus respectivas ocho horas por día.

Mano De Obra	
Sueldo ingeniero/mes	1.750,00 €
Meses	2
Total	3.500,00 €

Tabla 11: Mano de obra

Así pues el presupuesto total final queda:

Presupuesto Total	
Concepto	**Valor**
Mano de Obra	3.500,00 €
Precio Fabricación Robot	143,10 €
Subtotal	3.643,10 €
IVA	21%
TOTAL	4.408,15 €

Tabla 12: Presupuesto Total Final

El presupuesto total final asciende al valor de CUATRO MIL CUATROCIENTOS OCHO EUROS CON QUINCE CÉNTIMOS.

8 CONCLUSIONES

Se empezó este trabajo con el reto de poder emplear y demostrar parte de los conocimientos adquiridos a lo largo del Grado y utilizarlos en un caso práctico.

Como ya se comentó al principio de la memoria, personalmente siempre se tuvo un alto interés en todo lo que abarca el área de las tecnologías, concretamente en temas que tienen que ver con la robótica y la electrónica. Qué mejor manera de adquirir una mayor experiencia y desarrollo en estos temas con la realización de un TFG de estas características, en donde se ha tenido que combinar multitud de disciplinas como:

- Instrumentación
- Microcontroladores
- Protocolos de comunicación
- Control de sistemas
- Programación
- Elección de componentes
- Circuitos electrónicos
- Electrónica de potencia

Gracias a la combinación de todas las disciplinas mencionadas, ejecutándolas en su adecuado orden cronológico, se pudo conseguir un TFG con las siguientes características:

- Robot con regulación mediante algoritmo PID
- Realización de un diseño mecánico, donde se ha utilizado SolidWorks
- Montaje completo de una pequeña estructura robótica
- Realización de un diseño electrónico, donde se ha utilizado Designspark
- Montaje y soldadura de los componentes electrónicos en PCBs
- Comunicación y control del robot con un Smartphone mediante tecnología Bluetooth
- Utilización de la plataforma Arduino como base principal de la programación del robot
- Un precio de fabricación bastante económico que se acerca a los analizados en el Estado del Arte

9 LÍNEAS FUTURAS

Se ha conseguido desarrollar un robot, totalmente autónomo, capaz de auto-balancearse con sus dos ruedas. Sin embargo, aún se podría seguir añadiéndole futuras posibles mejoras o desarrollos, las cuales se pasan a describir a continuación:

- Como se comentó en el apartado de justificación del trabajo, este TFG serviría como punto de partida para un proyecto más grande acerca de un vehículo inteligente de transporte basado en el mismo concepto de mantener el equilibrio con dos ruedas.

- Se podría conseguir un mejor comportamiento en la estabilidad del sistema empleando técnicas modernas de control, estas técnicas se basan en la lógica difusa con la que se puede implementar controladores conocidos con el nombre de fuzzy o borrosos.

- Realizando un modelado preciso del sistema y posteriormente un simulador capaz de asemejarse mucho a la realidad, se conseguiría obtener mejores resultados en lo respecto al control del robot.

- Hay la posibilidad de conseguir mayor capacidad de comunicación con otros dispositivos añadiendo, aparte del módulo bluetooth, otro módulo compatible con la tecnología Wifi.

- Un factor que ha condicionado mucho el comportamiento del robot, es la elección de los motores. Por lo tanto, se obtendría mejores resultados con unos motores más grandes capaces de funcionar a 12V y así poder transmitir un mayor par a la estructura robótica.

- El uso de encoders en las ruedas, daría la capacidad de poder implementar un lazo de control en cascada, uno de control de velocidad y otro de control de la posición vertical del robot. De esta manera se consigue un mejor comportamiento con respecto a la estabilidad.

- Incorporando una pequeña pantalla LCD al robot, se daría la posibilidad de visualizar continuamente las variables importantes del sistema.

10 BIBLIOGRAFÍA

[1] http://www.roboticspot.com/robotica.php

[2] http://lema.rae.es/drae/?val=rob%C3%B3tica

[3] http://www.robotics.org/

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

[5] http://www.hitachi.com/rd/portal/highlight/robotics/emiew1_01/index.html

[6] http://www.hitachi.com/rd/portal/highlight/robotics/emiew2_01/index.html

[7] http://www.slashgear.com/amp-music-playing-gyroscopic-dancing-companion-robot-1912138/

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

[9] http://www.segway.com/puma/

[10] https://www.indiegogo.com/projects/balanbot-best-arduino-self-balancing-robot-ever#/story

[11] http://www.geology.smu.edu/~dpa-www/robo/nbot/

[12] Libro: Ingeniería de control moderna, autor: Katsuhiko Ogata, 4ª ed.

[13] https://www.xsens.com/tags/imu/

[14] http://smartdreams.cl/unidad-de-medicion-inercial-imu/

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

[16] http://www.saberesyciencias.com.mx/sitio/component/content/article/10-portada/280-sistemas-micro-electromecanicos

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

[18] Apostolyuk V. Teoría y diseño de giroscopios vibrantes micromecánicos

[19] http://cursos.olimex.cl/giroscopio/

[20] http://www.ebay.es/itm/like/251911413184?limghlpsr=true&hlpv=2&ops=true&viphx=1&hlpht=true&lpid=115

[21] https://www.sparkfun.com/products/10121

[22] http://www.mikroe.com/chapters/view/79/capitulo-1-el-mundo-de-los-microcontroladores/

[23] Apuntes de la asignatura Microprocesadores, Tema 2 Sistemas Basados En Microcontroladores

[24] http://www.arduino.cc/es/pmwiki.php?n=

[25] http://www.pinguino.cc/

[26] http://www.modulo0tutoriales.com/10-razones-para-usar-arduino/

[27] http://www.arduino.cc/en/Main/ArduinoBoardUno

[28] http://www.arduino.cc/en/Main/ArduinoBoardDue

[29] http://www.arduino.cc/en/Main/ArduinoBoardYun

[30] http://www.arduino.cc/en/Main/ArduinoBoardMegaADK

[31] http://www.arduino.cc/en/Main/ArduinoBoardMini

[32] http://www.arduino.cc/en/Main/ArduinoBoardNano

[33] Capítulo 2. Sistemas de Control, archivo de la Universidad Politécnica de Cataluña

[34] Apuntes de la asignatura Control Automático, Tema 6 Propiedades de la realimentación

[35] http://madrid.verkstad.cc/es/course-literature/tipos-de-motores/

[36] http://solorobotica.blogspot.com.es/2011/08/motores-de-corriente-continua.html

[37] http://www.todorobot.com.ar/tutorial-sobre-motores-paso-a-paso-stepper-motors/

[38] http://www.infoab.uclm.es/labelec/solar/electronica/elementos/servomotor.htm

[39] https://www.pololu.com/category/60/micro-metal-gearmotors

[40] http://en.wikipedia.org/wiki/Motor_controller

[41] http://robots-argentina.com.ar/MotorCC_ControlAncho.htm

[42] http://www.electronicafacil.net/tutoriales/Protocolos-Bluetooth.php

[43] http://es.kioskea.net/contents/69-como-funciona-bluetooth

[44] http://robots-argentina.com.ar/Comunicacion_busI2C.htm

[45] http://www.geekmomprojects.com/mpu-6050-redux-dmp-data-fusion-vs-complementary-filter/

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

[47] http://www.geekmomprojects.com/mpu-6050-dmp-data-from-i2cdevlib/

[48] http://www.geekmomprojects.com/mpu-6050-redux-dmp-data-fusion-vs-complementary-filter/

[49] http://www.solidworks.com/

[50] http://raspberryparatorpes.net/glossary/pcb/

[51] http://www.rs-online.com/designspark/electronics/eng/page/designspark-pcb home-page

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

11 ANEXOS

Anexo1: Código de programación empleado para la obtención del ángulo de inclinación.

//Aquí se incluyen todas las librerías a utilizar en el código:

```
#include "I2Cdev.h"
#include "MPU6050_6Axis_MotionApps20.h"
#include "Wire.h"

MPU6050 mpu;

// Variables de control y estado del MPU6050
bool dmpReady = false;  // Variable que cambia a true si la inicialización del DMP fue satisfactoria
uint8_t mpuIntStatus;   // Mantiene el byte de estado actual de interrupción de la MPU
uint8_t devStatus;      // Indicador de estado después de cada operación del dispositivo (0 = éxito, !0 = error)
uint16_t packetSize;    // Tamaño esperado de paquetes de MPU(por defecto es 42 bytes)
uint16_t fifoCount;     // Cuenta de todos los bytes actualmente en FIFO
uint8_t fifoBuffer[64]; // Almacenamiento buffer FIFO

// Variables de orientación y movimiento
Quaternion q;           // [w, x, y, z]         vector de cuaterniones
VectorFloat gravity;    // [x, y, z]            vector de gravedad
float ypr[3];           // [yaw, pitch, roll]   vector de ángulos yaw/pitch/roll

// ================================================================
// ===               RUTINA DE DETECCIÓN DE INTERRUPCIÓN        ===
// ================================================================

volatile bool mpuInterrupt = false;     // Variable booleana que indica el cambio de estado del pin de interrupción MPU.
void dmpDataReady()
{
  mpuInterrupt = true;
}

// ================================================================
// ===                      CONFIGURACIÓN INICIAL                 ===
// ================================================================

void setup()
{
    Wire.begin();   // Inicialización de comunicación con el sensor vía bus I2C

  // Inicio comunicación Serial
  Serial.begin(115200);

  // Inicio dispositivo
  Serial.println(F("Inicializando dispositivos I2C..."));
  mpu.initialize();
```

```
    // Test de conexión
    Serial.println(F("Testeando conexión de dispositivos..."));
    Serial.println(mpu.testConnection() ? F("MPU6050 conexión satisfactoria") : F("MPU6050 conexión fallida"));

    // Configuración y carga del DMP
    Serial.println(F("Inicializando DMP..."));
    devStatus = mpu.dmpInitialize();

    // Configuración de los offset del MPU6050
    mpu.setXGyroOffset(-23);
    mpu.setYGyroOffset(-5);
    mpu.setZGyroOffset(-20);
    mpu.setZAccelOffset(2366);
    mpu.setYAccelOffset(-617);
    mpu.setXAccelOffset(-2994);

    // Comprobación de que todo va bien
    if (devStatus == 0)
    {
        // Encendido del DMP
        Serial.println(F("Habilitando DMP..."));
        mpu.setDMPEnabled(true);

        // Habilitar la detección de interrupción con Arduino
        Serial.println(F("Activando la detección de interrupción (interrupción externa 0) ... "));
        attachInterrupt(0, dmpDataReady, RISING);
        mpuIntStatus = mpu.getIntStatus();

        //Establecer nuestro flag dmpReady para indicar que el bucle principal está listo para ser usado
        Serial.println(F("DMP está listo! Esperando primera interrupción..."));
        dmpReady = true;


  // Obtener el tamaño del paquete DMP esperado,  para su posterior comparación
        packetSize = mpu.dmpGetFIFOPacketSize();
    }
    else  // en caso de que haya algún error
    {
        // ERROR!
        // 1 = la carga de memoria inicial fracasó
        // 2 = las actualizaciones de configuración DMP fracasaron
        Serial.print(F("Inicialización del DMP fallida código("));
        Serial.print(devStatus);
        Serial.println(F(")"));
    }

}

// ================================================================
// ===                      BUCLE DE PROGRAMA PRINCIPAL                                ===
```

```
// ================================================================

void loop()
{
    // Si DMP falla no hacer nada
    if (!dmpReady) return;

    // Restablecer el flag de interrupción y obtener byte INT_STATUS
    mpuInterrupt = false;
    mpuIntStatus = mpu.getIntStatus();

    // Obtener cuenta FIFO actual
    fifoCount = mpu.getFIFOCount();

    // Comprobar si existe desbordamiento
    if ((mpuIntStatus & 0x10) || fifoCount == 1024)
    {
        // Reiniciar FIFO para que poder continuar nuevamente
        mpu.resetFIFO();
        Serial.println(F("FIFO desbordado!"));

    // De lo contrario, comprobar interrupción DMP
    }
    else if (mpuIntStatus & 0x02)
    {
        // Esperar a que la longitud de datos disponibles sea correcta
        while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();

        // Leer un paquete desde FIFO
        mpu.getFIFOBytes(fifoBuffer, packetSize);
       fifoCount -= packetSize;

        // Obtención de los ángulos de trabajo Yaw, Pitch y Roll
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            mpu.dmpGetGravity(&gravity, &q);
            mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
      // Imprimir en el monitor serial los ángulo de trabajo del robot en grados
          Serial.println();
          Serial.print(" Yaw = ");
          Serial.println(ypr[0] * 180/M_PI);
          Serial.println();
          Serial.print(" Pitch = ");
          Serial.println(ypr[1] * 180/M_PI);
          Serial.println();
          Serial.print(" Roll = ");
          Serial.println(ypr[2] * 180/M_PI);
```

Anexo 2: Código de programación para pruebas del Driver Motor

```
//Prueba Driver Motor L298N

//Acción de los motores según el carácter enviado por el monitor serial:
//
// 1 -Motor A Giro Antihorario
// 2 -Motor A Stop
// 3 -Motor A Giro Horario
//
// 4 -Motor B Giro Antihorario
// 5 -Motor B Stop
// 6 -Motor B Giro Horario

// Declaración de pines para el Motor A
int MAPin1 = 6;
int MAPin2 = 4;
int PWMPinA = 5; // Necesita ser un Pin PWM de la placa Arduino

// Declaración de pines para el Motor B
int MBPin1 = 12;
int MBPin2 = 13;
int PWMPinB = 11; // Necesita ser un Pin PWM de la placa Arduino

void setup() //Configuración inicial
{
  Serial.begin(9600); //Inicializamos la comunicación serial a 9600 Baudios
  //Definimos como salidas lo pines de la placa Arduino, que serán entradas de la placa Driver

  pinMode(MAPin1,OUTPUT);
  pinMode(MAPin2,OUTPUT);
  pinMode(PWMPinA,OUTPUT);
  pinMode(MBPin1,OUTPUT);
  pinMode(MBPin2,OUTPUT);
  pinMode(PWMPinB,OUTPUT);
}

void loop() //Bucle de programa principal
{

  if (Serial.available() > 0) //Si hay comunicación serial
  {
    int caracter = Serial.read(); //Lee y guarda en la variable el caracter tecleado y enviado por el monitor
serial

switch (caracter) //Variable que contiene el caracter y será comparada según cada caso
    {
      //_______________Motor A_____________
      case '1': // En caso de haber recibido el caracter '1' Motor A Giro Antihorario
      analogWrite(PWMPinA, 255); //Configuramos la velocidad de giro del motor, valor entre 0-255
      digitalWrite(MAPin1, LOW); //Escribimos el Pin como '0' lógico
      digitalWrite(MAPin2, HIGH); //Escribimos el Pin como '1' lógico
```

```
    Serial.println("Motor A Giro Antihorario"); // Imprimimos en el monitor serial la acción ejecutada
    Serial.println("    "); // Imprimimos un salto de línea
    break;

    case '2': // En caso de haber recibido el caracter '2' Motor A Para-Stop
    analogWrite(PWMPinA, 0); //Configuramos la velocidad de giro del motor, valor entre 0-255
    digitalWrite(MAPin1, LOW); //Escribimos el Pin como '0' lógico
    digitalWrite(MAPin2, HIGH); //Escribimos el Pin como '1' lógico
    Serial.println("Motor A Para-Stop"); // Imprimimos en el monitor serial la acción ejecutada
    Serial.println("    "); // Imprimimos un salto de línea
    break;

    case '3': // En caso de haber recibido el caracter '3' Motor A Giro Horario
    analogWrite(PWMPinA, 255); //Configuramos la velocidad de giro del motor, valor entre 0-255
    digitalWrite(MAPin1, HIGH); //Escribimos el Pin como '1' lógico
    digitalWrite(MAPin2, LOW); //Escribimos el Pin como '0' lógico
    Serial.println("Motor A Giro Horario"); // Imprimimos en el monitor serial la acción ejecutada
    Serial.println("    "); // Imprimimos un salto de línea
    break;

    //_______________Motor B_____________
    case '4': // En caso de haber recibido el caracter '4' Motor B Giro Antihorario
    analogWrite(PWMPinB, 255); //Configuramos la velocidad de giro del motor, valor entre 0-255
    digitalWrite(MBPin1, LOW); //Escribimos el Pin como '0' lógico
    digitalWrite(MBPin2, HIGH); //Escribimos el Pin como '1' lógico
    Serial.println("Motor B Giro Antihorario"); // Imprimimos en el monitor serial la acción ejecutada
    Serial.println("    "); // Imprimimos un salto de línea
    break;

    case '5': // En caso de haber recibido el caracter '5' Motor B Para-Stop
    analogWrite(PWMPinB, 0); //Configuramos la velocidad de giro del motor, valor entre 0-255
    digitalWrite(MBPin1, LOW); //Escribimos el Pin como '0' lógico
    digitalWrite(MBPin2, HIGH); //Escribimos el Pin como '1' lógico
    Serial.println("Motor B Para-Stop"); // Imprimimos en el monitor serial la acción ejecutada
    Serial.println("    "); // Imprimimos un salto de línea
    break;

    case '6': // En caso de haber recibido el caracter '6' Motor B Giro Horario
    analogWrite(PWMPinB, 255); //Configuramos la velocidad de giro del motor, valor entre 0-255
    digitalWrite(MBPin1, HIGH); //Escribimos el Pin como '1' lógico
    digitalWrite(MBPin2, LOW); //Escribimos el Pin como '0' lógico
    Serial.println("Motor B Giro Horario"); // Imprimimos en el monitor serial la acción ejecutada
    Serial.println("    "); // Imprimimos un salto de línea
    break;
   }
  }
}
```

Anexo 3: Código final para el algoritmo de control PID

```
//Declaración Variables a emplear en el algoritmo de control

float kp, ki, kd;      // Se definen las constantes de control que se emplean en el algoritmo
float angulomedido , angulodeseado = 0, Accion_integ, deriv_error, error, anguloanterio;
float SalidaMA, SalidaMB;  //Acción de control que se transmitira a los motores mediante PWM
float TiempoMuestreo = 0.1;  //= x .Muestras cada x tiempo de nuestro algoritmo de control
unsigned long TiempoAnterior = 0;
float Salida;
int TiempoCambio;
//--------------------------------------------------------------------------------------

// ALGORITMO DE CONTROL (PID)

//--------------------------------------------------------------------------------------

void miPID()

{

  //Asignación de valores para nuestras constantes de control
  kp = analogRead(A0)*0.1; // Lee el valor del potenciómetro KP y lo imprime en pantalla
  Serial.print("  kp=");
  Serial.print(kp);
  kd = analogRead(A2)*0.5; // Lee el valor del potenciómetro KD y lo imprime en pantalla
  Serial.print("  kd=");
  Serial.print(kd);
  ki = analogRead(A1)*0.01; // Lee el valor del potenciómetro KI y lo imprime en pantalla
  Serial.print("  ki=");
  Serial.print(ki);
  // Si se quiere asignar valores fijos mediante programación a las constantes de control hay que descomentar  las siguientes tres líneas

  //kp = 0;  Serial.print("  kp=");Serial.print(kp);
  //kd = 0;  Serial.print("  kd=");Serial.print(kd);
  //ki = 0;  Serial.print("  ki=");Serial.print(ki);


 unsigned long Ahora = millis();  // Indicará el tiempo que ha pasado desde que la placa empezó a ejecutar el programa
 TiempoCambio = (Ahora - TiempoAnterior); //Indicará el tiempo transcurrido desde la última vez que se ejecutó el PID

  //Cumple la función de saber cuándo hay que ejecutar el PID o retornar de la función
  if(TiempoCambio >= TiempoMuestreo)
  {
    //Se calculan todas las variables de error
    error = angulodeseado - angulomedido;  //Cálculo del error y se imprime por pantalla
    Serial.print(" error = ");
    Serial.println(error);
    Accion_integ += (ki*(error * TiempoCambio)); //Cálculo separado de la acción integral
    //Limites asignados a la acción integral [255,-255]
    if(Accion_integ > 255)
```

```
      Accion_integ = 255;
   else if(Accion_integ < -255)
      Accion_integ = -255;
   deriv_error = ((angulomedido-anguloanterior) / TiempoCambio);  //Cálculo de la derivada de error

   //Cálculo de la acción de control
   Salida = (kp * error) + Accion_integ - (kd * deriv_error);

  //Limites asignados a la acción de control [255,-255]
   if(Salida > 255)
      Salida = 255;
   else if(Salida < -255)
      Salida = -255;

   //Asignación de la acción de control a la salida de los motores y su impresión en pantalla
   SalidaMA = Salida; Serial.print("  SalidaMA=");Serial.print(SalidaMA);
   SalidaMB = Salida; Serial.print("  SalidaMB=");Serial.println(SalidaMB);

   //Se guardan tanto tiempo como error para el próximo ciclo de cálculo
   anguloanterior = angulomedido; //
   TiempoAnterior = Ahora;
  }
}
```

Anexo 4: Configuración inicial del módulo HC-06

```
//          La configuración Bluetooth se guarda en el módulo, así que solo es necesario configurarlo una vez.
//          La configuración se ha de hacer con el modulo sin emparejar, es decir, sin estar conectado con nada.

// Opciones de configuración:
          char ssid[10]              = "Arduino1";     // Nombre para el modulo Bluetooth.
          char baudios               = '8';            // 1=>1200 baudios, 2=>2400, 3=>4800, 4=>9600 (por defecto), 5=>19200, 6=>38400, 7=>57600, 8=>115200
          char password[10]          = "1234";     // Contraseña para el emparejamiento del módulo.

void setup()
{
  Serial.begin(9600);

  // Tiempo de espera:
   pinMode(13,OUTPUT); //Led Arduino como salida
   digitalWrite(13,HIGH); //Encender Led Arduino
   delay(10000);
   digitalWrite(13,LOW); //Apagar Led Arduino

   // Ahora se procede a la configuración del módulo:

    // Se inicia la configuración:
     Serial.print("AT"); delay(1000);
```

```
  // Se ajusta el nombre del Bluetooth:
    Serial.print("AT+NAME"); Serial.print(ssid); delay(1000);
  // Se ajustan los baudios:
    Serial.print("AT+BAUD"); Serial.print(baudios); delay(1000);
  // Se ajusta la contraseña:
    Serial.print("AT+PIN"); Serial.print(password); delay(1000);
}

void loop()
{
        // Al parpadear el led de Arduino se habrá concluido la configuración:
        digitalWrite(13, !digitalRead(13));
        delay(500);
}
```

Anexo 5: Programación Completo del Robot Arduino

```
//Aquí se incluyen todas las librerías a utilizar en el código:
#include "I2Cdev.h"
#include "MPU6050_6Axis_MotionApps20.h"
#include "Wire.h"

MPU6050 mpu;

//Declaración Variables Bluetooth
char DatoRecibido, Dato; // Variable para recibir datos del puerto serie
int ledpin = 13;    // Pin donde se encuentra conectado el led (pin 13)

//Declaración Variables de control y estado del MPU6050
bool dmpReady = false;  // Variable que cambia a true si la inicialización del DMP fue satisfactoria
uint8_t mpuIntStatus;   // Mantiene el byte de estado actual de interrupción de la MPU
uint8_t devStatus;      // Indicador de estado después de cada operación del dispositivo (0 = éxito, !0 =
error)
uint16_t packetSize;    // Tamaño esperado de paquetes de MPU(por defecto es 42 bytes)
uint16_t fifoCount;     // Cuenta de todos los bytes actualmente en FIFO
uint8_t fifoBuffer[64]; // Almacenamiento buffer FIFO

//Declaración Variables de orientación y movimiento
Quaternion q;           // [w, x, y, z]         vector de cuaterniones
VectorFloat gravity;    // [x, y, z]            vector de gravedad
float ypr[3];           // [yaw, pitch, roll]   vecto de ángulos yaw/pitch/roll

//Se define el valor de los offsets de los motores que mejoran el comportamiento del control
#define Offset_MA 15
#define Offset_MB 15

//Declaración Variables a emplear en el algoritmo de control
float kp, ki, kd;      // Se definen las constantes de control que se emplean en el algoritmo
float angulomedido, angulodeseado = 0, anguloavance = 0.4;
float SalidaMA,SalidaMB;  //Acción de control que se transmitira a los motores mediante PWM
```

```
float TiempoMuestreo = 0.1;  //= x .Muestras cada x tiempo de nuestro algoritmo de control
unsigned long TiempoAnterior;
float Salida;
float accion_integ, deriv_error, error, errAnterior, anguloanterior;
int TiempoCambio;
boolean GD, GI;

//Declaración Variables de los motores
int MA1=4;
int MA2=6;
int PWMA=5; // Necesita ser un Pin PWM de la placa Arduino
int MB1=12;
int MB2=10;
int PWMB=11; // Necesita ser un Pin PWM de la placa Arduino

// ================================================================
// ===         RUTINA DE DETECCIÓN DE INTERRUPCIÓN           ===
// ================================================================

volatile bool mpuInterrupt = false;  // Variable booleana que indica el cambio de estado del pin de interrupción MPU.
void dmpDataReady()
{
   mpuInterrupt = true;
}

// ================================================================
// ===               CONFIGURACIÓN INICIAL                    ===
// ================================================================

void setup()
{
      Wire.begin(); // Inicialización de comunicación con el sensor vía bus I2C

   // Inicio comunicación Serial
   Serial.begin(115200);

   // Inicio dispositivo
   Serial.println(F("Inicializando dispositivos I2C..."));
   mpu.initialize();

   // Test de conexión
   Serial.println(F("Testeando conexión de dispositivos..."));
   Serial.println(mpu.testConnection() ? F("MPU6050 conexión satisfactoria") : F("MPU6050 conexión fallida"));

   // Configuración y carga del DMP
   Serial.println(F("Inicializando DMP..."));
   devStatus = mpu.dmpInitialize();

   // Configuración de los offset del MPU6050
   mpu.setXGyroOffset(-23);
```

```
  mpu.setYGyroOffset(-5);
  mpu.setZGyroOffset(-20);
  mpu.setZAccelOffset(2366);
  mpu.setYAccelOffset(-617);
  mpu.setXAccelOffset(-2994);

  // Comprobación de que todo va bien
  if (devStatus == 0)
  {
    // Encendido del DMP
    Serial.println(F("Habilitando DMP..."));
    mpu.setDMPEnabled(true);

    // Habilitar la detección de interrupción con Arduino
    Serial.println(F("Activando la detección de interrupción (interrupción externa 0) ... "));
    attachInterrupt(0, dmpDataReady, RISING);
    mpuIntStatus = mpu.getIntStatus();

    //Establecer nuestro flag dmpReady para indicar que el bucle principal está listo para ser usado
    Serial.println(F("DMP está listo! Esperando primera interrupción..."));
    dmpReady = true;

    // Obtener el tamaño del paquete DMP esperado,  para su posterior comparación
    packetSize = mpu.dmpGetFIFOPacketSize();
  }
  else // en caso de que haya algún error
  {
    // ERROR!
    // 1 = la carga de memoria inicial fracasó
    // 2 = las actualizaciones de configuración DMP fracasaron
    Serial.print(F("Inicialización del DMP fallida código("));
    Serial.print(devStatus);
    Serial.println(F(")"));
  }

  // Configuración Led Arduino como salida
  pinMode(ledpin, OUTPUT);

  // Declaración los pines para el control de los motores como salidas
  pinMode(MA1, OUTPUT);
  pinMode(MA2, OUTPUT);
  pinMode(MB1, OUTPUT);
  pinMode(MB2, OUTPUT);
  pinMode(PWMA, OUTPUT);
  pinMode(PWMB, OUTPUT);


}

// ================================================================
// ===          BUCLE DE PROGRAMA PRINCIPAL                ===
// ================================================================
```

```
void loop()
{
  // Si DMP falla no hacer nada
  if (!dmpReady) return;

  // Restablecer el flag de interrupción y obtener byte INT_STATUS
  mpuInterrupt = false;
  mpuIntStatus = mpu.getIntStatus();

  // Obtener cuenta FIFO actual
  fifoCount = mpu.getFIFOCount();

  // Comprobar si existe desbordamiento
  if ((mpuIntStatus & 0x10) || fifoCount == 1024)
  {
    // Reiniciar FIFO para que poder continuar nuevamente
    mpu.resetFIFO();
    Serial.println(F("FIFO desbordado!"));
  }
  else if (mpuIntStatus & 0x02) // Si no hay desbordamiento continuar ejecutando código
  {
    // Esperar que la longitud de datos sea la correcta
    while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();

    // Lee un paquete de datos de FIFO
    mpu.getFIFOBytes(fifoBuffer, packetSize);
    fifoCount -= packetSize;

    // Obtención de los ángulos de trabajo Yaw, Pitch y Roll
      mpu.dmpGetQuaternion(&q, fifoBuffer);
      mpu.dmpGetGravity(&gravity, &q);
      mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
      // Imprimir el ángulo de inclinación (pitch) del robot en grados si se desea
//      Serial.println();
//      Serial.print(" angulomedido = ");
//      Serial.println(ypr[1] * 180/M_PI);

    // Acciones para el control del balanceado
    angulomedido = (ypr[1] * 180/M_PI);  // Obtención del ángulo Pitch en grados
    if (abs(angulomedido)<45)  // Empieza a ejecutar el control sólo cuando el robot no esté muy
inclinado
    {
     bluetooth();  //Ejecuta función bluetooth
     miPID();      //Ejecuta función algoritmo de control
     ControlPWM(); //Ejecuta función control de motores
    }
    else  // En caso de que el robot esté muy inclinado no se da potencia a los motores
    {
     analogWrite(PWMA, 0);
     analogWrite(PWMB, 0);
    }
```

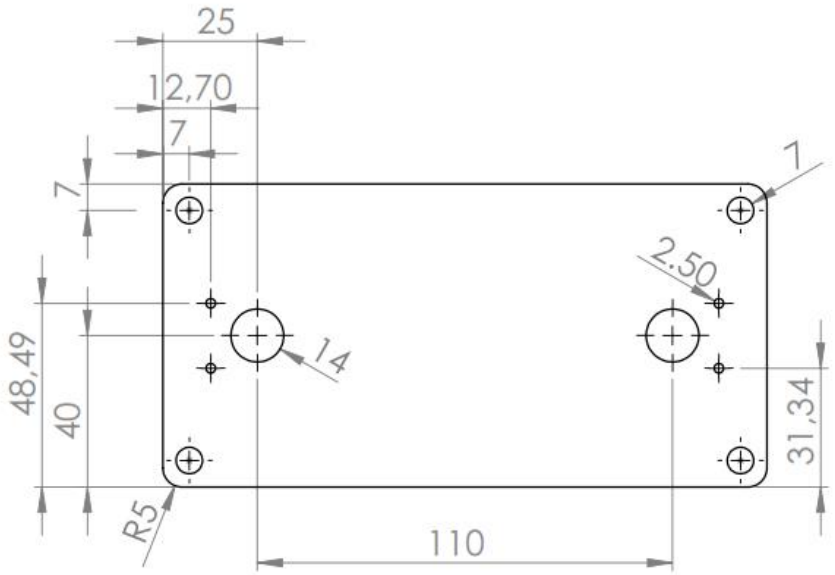
```
    }


}


// ================================================================
// ===          FUNCIÓN PARA MÓDULO BLUETOOTH              ===
// ================================================================
void bluetooth()
{
 if( Serial.available() ) //Si se ha recibido un comando
          {
                  // Leer un byte y colocarlo en variable
                  DatoRecibido = Serial.read();
           Dato = DatoRecibido;
                  // Procesar comando de un solo byte y decidir qué acción ejecuta el robot
                  if( DatoRecibido == 'A' )
                  {
                          digitalWrite(ledpin, HIGH);
                Serial.println("Avance");
                angulodeseado = anguloavance;
                  }
           if( Dato == 'D' )
                  {
                          digitalWrite(ledpin, HIGH);
                Serial.println("Giro Derecha");
                GD=1;
                GI=0;

                  }
           if( Dato == 'I' )
                  {
                          digitalWrite(ledpin, HIGH);
                Serial.println("Giro Izquierda");
                GD=0;
                GI=1;
                  }
                  if( DatoRecibido == 'R' )
                  {
                          digitalWrite(ledpin, LOW);
                          Serial.println("Retroceso");
                angulodeseado = -anguloavance;

                  }

          }

}

// ================================================================
```

```
// ===          FUNCIÓN ALGORITMO DE CONTROL (PID)          ===
// ================================================================
void miPID()
{

 if (DatoRecibido!='A'||DatoRecibido!='R') //Su función es distinguir si se ha recibido el comando A o R de los demás comandos
    DatoRecibido='X';

  kp = analogRead(A0)*0.1; //Se lee el valor analógico del potenciómetro Kp y se imprime en pantalla
  Serial.print(" kp= ");
  Serial.print(kp);
  kd = analogRead(A2)*0.1; //Se lee el valor analógico del potenciómetro Kd y se imprime en pantalla
  Serial.print(" kd= ");
  Serial.print(kd);

  Serial.print(" Ad= "); //Se imprime en pantalla el valor del ángulo de referencia
  Serial.print(angulodeseado);

if (angulodeseado < -0.01 || angulodeseado > 0.01)  //Permite facilitar avance o retroceso poniendo ki=0 durante el movimiento
  ki = 0;

 else if (DatoRecibido=='X'||angulodeseado == 0.00) //Vuelve a establecer el ki configurado
   ki = analogRead(A1)*0.1; //Se lee el valor analógico del potenciómetro Ki

  if (angulodeseado<-0.01 && DatoRecibido=='X') //Si se ha activado retroceso, la referencia volverá a cero gradualmente
     angulodeseado = angulodeseado+0.1;
  else if (angulodeseado>0.01 && DatoRecibido=='X') //Si se ha activado avance, la referencia volverá a cero gradualmente
         angulodeseado = angulodeseado-0.1;
  Serial.print(" ki= ");  //se imprime en pantalla Ki
  Serial.println(ki);
  //Serial.println(DatoRecibido);
  //Serial.println(Dato);

  //kp = 0;  Serial.print("  kp=");Serial.print(kp);
  //kd = 0;  Serial.print("  kd=");Serial.print(kd);
  //ki = 0;  Serial.print("  ki=");Serial.print(ki);

  unsigned long Ahora = millis(); // Indicará el tiempo que ha pasado desde que la placa empezó a ejecutar el programa
  TiempoCambio = (Ahora - TiempoAnterior); //Indicará el tiempo transcurrido desde la última vez que se ejecutó el PID

  //Cumple la función de saber cuándo hay que ejecutar el PID o retornar de la función
  if(TiempoCambio >= TiempoMuestreo)
  {
    //Se calculan todas las variables de error
    error = angulodeseado - angulomedido;
    Serial.print(" Error = "); //Cálculo del error y se imprime por pantalla
```

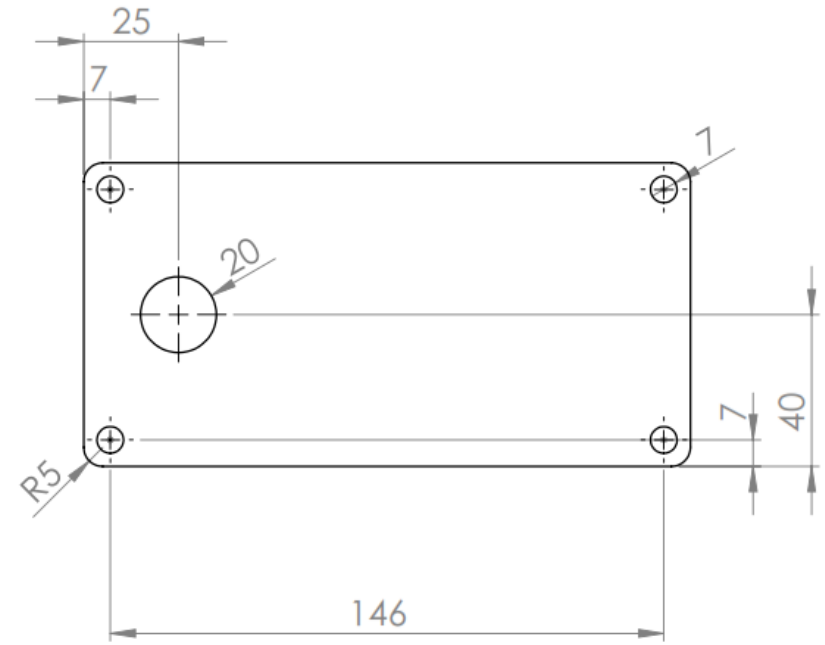
```
  Serial.println(error);
  accion_integ += (ki*(error * TiempoMuestreo)); //Cálculo separado de la acción integral
  //Limites asignados a la acción integral [255,-255]
  if (accion_integ > 255)
    accion_integ = 255;
  else if (accion_integ < -255)
     accion_integ = -255;
  deriv_error = (angulomedido - anguloanterior) / TiempoMuestreo;  //Cálculo de la derivada de error

  //Cálculo de la acción de control
  Salida = kp * error + accion_integ - kd * deriv_error;
  //Limites asignados a la acción de control [255,-255]
  if(Salida > 255)
     Salida = 255;
  else if(Salida < -255)
     Salida = -255;
  Serial.print(" Salida = "); //Salida se imprime por pantalla
  Serial.println(Salida);
  //Asignación de la acción de control a la salida de los motores
  if (GI==1)  //Si se ha recibido comando giro izquierda
  {
    SalidaMA = Salida+30;
    SalidaMB = Salida-30;
  }
  else if (GD==1)  //Si se ha recibido comando giro derecha
  {
    SalidaMA = Salida-30;
    SalidaMB = Salida+30;
  }
  else  // En caso de n recibir ningún comando de giro, mantener equilibrio
  {
    SalidaMA = Salida;
    SalidaMB = Salida;
  }

  //Se guardan tanto tiempo como error para el próximo ciclo de cálculo
  anguloanterior = angulomedido;
  TiempoAnterior = Ahora;
 }
}

// ================================================================
// ===              FUNCIÓN CONTROL DE MOTORES                ===
// ================================================================
void ControlPWM()
{
 if (Dato!='D'||Dato!='I') //Su función es distinguir si se ha recibido algún comando de giro D o I
 {
    Dato='C';
    GI=0;
    GD=0;
 }
```

```
  //Dependiendo la acción de control calculada, el motor A girará en un sentido u otro para mantener el equilibrio
  if(SalidaMA < 0)
  {
    digitalWrite(MA1, HIGH);
    digitalWrite(MA2, LOW);
  }
  else if(SalidaMA > 0)
  {
    digitalWrite(MA1, LOW);
    digitalWrite(MA2, HIGH);
  }
  else  //En el caso de ser 0, motores inhabilitados
  {
    digitalWrite(MA1, HIGH);
    digitalWrite(MA2, HIGH);
  }
  //Dependiendo la acción de control calculada, el motor B girará en un sentido u otro para mantener el equilibrio
  if(SalidaMB < 0)
  {
    digitalWrite(MB1, HIGH);
    digitalWrite(MB2, LOW);
  }
  else if(SalidaMB > 0)
  {
    digitalWrite(MB1, LOW);
    digitalWrite(MB2, HIGH);
  }
  else  //En el caso de ser 0, motores inhabilitados
  {
    digitalWrite(MB1, HIGH);
    digitalWrite(MB2, HIGH);
  }
    // Valor PWM transmitido a cada motor
    analogWrite(PWMA, min(255, abs(SalidaMA) + Offset_MA));  //con la función "min" se logra que no vaya un valor más alto de 255
    analogWrite(PWMB, min(255, abs(SalidaMB) + Offset_MB));
}
```

Anexo 6: Planos



Universidad Pública de Navarra *Nafarroako Unibertsitate Publikoa*	**E.T.S.I.I.T.** **INGENIERO ELÉCTRICO Y ELECTRÓNICO**	DEPARTAMENTO: **DEPARTAMENTO DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA**
PROYECTO: **DISEÑO E IMPLEMENTACIÓN EN PCB DE UN ROBOT AUTO-BALANCEADO BASADO EN ARDUINO**		REALIZADO: **FALCONI LOJA, VICTOR E.** FIRMA:
PLANO: **BASE METÁLICA 1**	FECHA: 15/06/2015 ESCALA: 1:2	Nº PLANO: 1



Universidad Pública de Navarra *Nafarroako Unibertsitate Publikoa*	**E.T.S.I.I.T.** **INGENIERO ELÉCTRICO Y ELECTRÓNICO**	DEPARTAMENTO: **DEPARTAMENTO DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA**
PROYECTO: **DISEÑO E IMPLEMENTACIÓN EN PCB DE UN ROBOT AUTO-BALANCEADO BASADO EN ARDUINO**		REALIZADO: **FALCONI LOJA, VICTOR E.** FIRMA:
PLANO: **BASE METÁLICA 2**	FECHA: 15/06/2015	ESCALA: 1:2 / Nº PLANO: 2