Escuela Politécnica Superior de Jaén

Grado en Ingeniería Electrónica Industrial

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior

Trabajo Fin de Grado

VEHÍCULO AUTOPROPULSADO BASADO EN EL PÉNDULO INVERTIDO

Alumno: Manuel Joaquín Pérez Serrano

Tutor: Prof. D. Carlos Luis Sánchez Martos
Dpto: Ingeniería Eléctrica

Junio, 2019

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

Don CARLOS LUIS SÁNCHEZ MARTOS , tutor del Proyecto Fin de Carrera titulado: VEHÍCULO AUTOPROPULSADO BASADO EN EL PÉNDULO INVERTIDO, que presenta MANUEL JOAQUÍN PÉREZ SERRANO, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

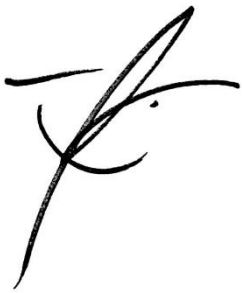
Jaén, JUNIO de 2019

El alumno:

MANUEL JOAQUÍN

PÉREZ SERRANO

Los tutores:



CARLOS LUIS

SÁNCHEZ MARTOS

1 Índice

Índice de ilustraciones

Índice de tablas

2 Objetivo

El objetivo de este Trabajo Fin de Grado es el diseño e implementación de un vehículo autopropulsado basado en el péndulo invertido. Como su propio nombre indica, consiste en la realización de un dispositivo capaz de mantener el equilibrio sobre dos ruedas, además de poder ser controlado de forma remota. Para la realización de este proyecto será necesario, en primer lugar, estudiar la teoría del péndulo invertido. Posteriormente se procederá a la selección de los componentes necesarios para el correcto funcionamiento de este, siendo el último paso la construcción física del dispositivo, así como su programación.

3 Introducción: Teoría del péndulo invertido

El péndulo invertido es conocido por ser uno de los problemas más importantes y clásicos de la teoría de control. Este consiste básicamente en que su centro de masas está situado por encima de un punto pivote siendo un sistema de naturaleza inestable ya que, en ausencia de fuerzas externas, el péndulo caería. Por este motivo es necesario un sistema de control que continuamente monitorice el ángulo de inclinación y según este, desplace el punto pivote para así conseguir estabilizar el sistema en posición vertical.

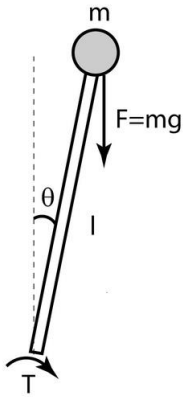


Ilustración 3-1: Modelo péndulo invertido. [1]

La primera solución a este sistema fue descrita por Roberge en 1960 [2] y posteriormente por Schaefer y Cannon en 1966 [3]. Y aunque este concepto cumpla

ya más de 50 años, sigue siendo un problema muy actual, veamos algunos ejemplos:

En 2001 se presentó el "Segway Personal Transporter" siendo el primer dispositivo de transporte con autobalanceo. El ordenador y los motores eléctricos situados en la base mantienen el Segway en posición vertical y este es capaz de moverse gracias a la inclinación definida por el usuario.

Ilustración 3-2: Segway

Más tarde, en 2005 se lanzó en China los "Hoverboard", basados en el mismo principio sólo que ya no presentan manillar, por lo que, para girar el usuario debe inclinar ambos pies por separado.

Ilustración 3-3: Hoverboard

Pero el concepto del péndulo invertido no sólo se implementa en sistemas de transporte, también es utilizado para controlar los primeros momentos en el

despegue de cohetes espaciales o misiles, cuando estos se deben mantener en posición vertical pero la velocidad del aire es tan pequeña que la estabilidad aerodinámica resulta imposible.

Ilustración 3-4: Despegue cohete espacial NASA

Un último ejemplo y que cada vez está teniendo una mayor importancia consiste en la aplicación de la teoría del péndulo invertido sobre el control del desplazamiento de robots humanoides. Estos robots pueden desplazarse siguiendo un sistema bípedo como los humanos o mediante ruedas u otros tipos de dispositivos.

Ilustración 3-5: Robot ASIMO caminando

Ilustración 3-6: Robot Handle

3.1 Descripción matemática

En esta sección se describe mediante formulaciones matemáticas la ecuación que define el comportamiento físico del vehículo.

[4] Página 88

DESCRIPCIÓN	SÍMBOLO
Masa del péndulo (estructura superior)	M_P
Masa de la rueda	M_R
Momento de inercia del péndulo	J_P
Momento de inercia de la rueda	J_R
Coordenada horizontal del centro de gravedad del péndulo	x_P
Coordenada horizontal del centro de gravedad de la rueda	x_P
Coordenada vertical del centro de gravedad del péndulo	y_P
Distancia entre x_P y x_R	L
Radio de la rueda	r
Desplazamiento angular del péndulo	θ_P
Desplazamiento angular de la rueda	θ_R
Aceleración de la gravedad	g

Tabla 3-1: Variables modelo matemático

Como lo que nos interesa es el desplazamiento angular respecto a la posición de equilibrio vertical, escogeremos como coordenadas generalizadas θ_P y θ_r. El siguiente paso es relacionar las distintas variables de posición con estas coordenadas generalizadas de manera que:

$$x_P = x_R + L\sin(\theta_P)$$
(*Fórmula 2.1)*

$$x_R = r\theta_R$$
(*Fórmula 2.2)*

$$x_P = r\theta_R + L\sin(\theta_P)$$
(*Fórmula 1.3)*

$$y_P = L\cos(\theta_P)$$
(*Fórmula 2.4)*

Derivando 2.1, 2.2 y 2.4 obtenemos las velocidades de traslación de las ruedas y del péndulo respectivamente.

$$\dot{x}_R = r\dot{\theta}_R$$
(*Fórmula 2.5)*

$$\dot{x}_P = r\dot{\theta}_R + L\,\dot{\theta}_P\cos(\theta_P)$$
(*Fórmula 2.6)*

$$\dot{y}_P = -L\theta_P\sin(\theta_P)$$
(*Fórmula 2.7)*

Siendo la velocidad de traslación del péndulo:

$$\dot{x}_P + \dot{y}_P = r\dot{\theta}_R + L\,\dot{\theta}_P\cos(\theta_P) - L\theta_P\sin(\theta_P)$$
(*Fórmula 2.8)*

Si tomamos como nuestro eje de referencia el eje de rotación de las ruedas, la energía potencial del sistema sólo dependerá de la estructura superior, es decir, de θ_P ya que la coordenada vertical del centro de gravedad de la rueda es cero.

$$U = M_P g y_P = M_P g L\cos(\theta_P)$$
(*Fórmula 2.9)*

Por otro lado, la energía cinética total del sistema será la suma de la energía cinética de la estructura superior y la energía cinética de las ruedas.

$$T_T = T_p + T_r$$
(*Fórmula 2.10)*

A su vez, cada término de energía cinética está formado por una componente traslacional y otra rotacional. Esto se debe a que nuestro vehículo no sólo será capaz de avanzar en línea recta, sino que podrá girar en ambos sentidos.

Energía cinética del péndulo:

$$T_{P_{tras}} = \frac{1}{2} M_P (\dot{x}_P^2 + \dot{y}_P^2) = \frac{1}{2} M_P \left(\left(r\dot{\theta}_R + L\,\dot{\theta}_P \cos(\theta_P) \right)^2 + (-L\theta_P \sin(\theta_P))^2 \right)$$

(*Fórmula 2.11)*

$$T_{P_{rot}} = \frac{1}{2} J_P\, \dot{\theta}_P^2$$

(*Fórmula 2.12)*

$$T_P = T_{P_{tras}} + T_{P_{rot}} = \frac{1}{2} M_P \left(\left(r\dot{\theta}_R + L\,\dot{\theta}_P \cos(\theta_P) \right)^2 + (-L\theta_P \sin(\theta_P))^2 \right) + \frac{1}{2} J_P\, \dot{\theta}_P^2$$

(*Fórmula 2.13)*

Energía cinética de las ruedas:

$$T_{R_{tras}} = \frac{1}{2} M_R\, \dot{x}_R^2 = \frac{1}{2} M_R\, r^2 \dot{\theta}_R^2$$

(*Fórmula 2.14)*

$$T_{R_{rot}} = \frac{1}{2} J_R\, \dot{\theta}_R^2$$

(*Fórmula 2.15)*

$$2\,T_R = 2(T_{R_{tras}} + T_{R_{rot}}) = M_R\, r\dot{\theta}_R^2 + J_R\, \dot{\theta}_R^2$$

(*Fórmula 2.16)*

Energía cinética total:

$$\begin{aligned} T_T &= \frac{1}{2} M_P \left(\left(r\dot{\theta}_R + L\,\dot{\theta}_P \cos(\theta_P) \right)^2 + (-L\theta_P \sin(\theta_P))^2 \right) + \frac{1}{2} J_P\, \dot{\theta}_P^2 + \frac{1}{2} M_R\, r^2 \dot{\theta}_R^2 + \frac{1}{2} J_R\, \dot{\theta}_R^2 \\ &= \left(\frac{1}{2} M_P + M_R \right) r^2\, \dot{\theta}_R^2 + \frac{1}{2} M_P L^2\, \dot{\theta}_P^2 + M_P r L \dot{\theta}_P \dot{\theta}_R \cos(\theta_P) + \frac{1}{2} J_P\, \dot{\theta}_P^2 + \frac{1}{2} J_R\, \dot{\theta}_R^2 \end{aligned}$$

(*Fórmula 2.17)*

Teniendo en cuenta que el momento de inercia de una masa puntual con respecto a un eje se define como el producto de la masa por la distancia perpendicular al eje elevada al cuadrado, obtenemos:

$$J_P = M_P L^2 \qquad J_R = M_R r^2$$

(*Fórmula 2.18)*

Sustituyendo:

$$T_T = \left(\frac{1}{2} M_P + M_R \right) r^2\, \dot{\theta}_R^2 + \frac{1}{2} M_P L^2\, \dot{\theta}_P^2 + M_P r L \dot{\theta}_P \dot{\theta}_R \cos(\theta_P) + \frac{1}{2} M_P L^2\, \dot{\theta}_P^2 + \frac{1}{2} M_R r^2\, \dot{\theta}_R^2$$

(*Fórmula 2.19)*

Simplificando:

$$T_T = \left(\frac{1}{2} M_P + \frac{3}{2} M_R \right) r^2\, \dot{\theta}_R^2 + M_P L^2\, \dot{\theta}_P^2 + M_P r L \dot{\theta}_P \dot{\theta}_R \cos(\theta_P)$$

(*Fórmula 2.20)*

Finalmente, el Lagrangiano es:

$$L = T_T - U = \left(\frac{1}{2}M_P + \frac{3}{2}M_R\right) r^2\ \dot{\theta}_R^2 + M_P L^2\ \dot{\theta}_P^2 + M_P r L \dot{\theta}_P \dot{\theta}_R \cos(\theta_P) - M_P g L \cos(\theta_P)$$

(*Fórmula 2.21)*

Una vez tenemos el Lagrangiano, necesitamos calcular los términos involucrados en la ecuación de Lagrange:

$$\frac{d}{dt}\left(\frac{\partial L}{\dot{q}i}\right) - \left(\frac{\partial L}{qi}\right) = Q_i, \qquad i = 0, 1, 2, 3 \ldots$$

(*Fórmula 2.22)*

Siendo la ecuación de Lagrange particularizada para las coordenadas generalizadas basadas en el par de los motores (τ):

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}_R}\right) - \left(\frac{\partial L}{\partial \theta_R}\right) = \tau$$

(*Fórmula 2.23)*

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}_P}\right) - \frac{\partial L}{\partial \theta_P} = -\tau$$

(*Fórmula 2.24)*

Si despejamos el par motor:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}_R}\right) - \left(\frac{\partial L}{\partial \theta_R}\right) + \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}_P}\right) - \frac{\partial L}{\partial \theta_P} = 0$$

(*Fórmula 2.25)*

Derivada del Lagrangiano con respecto al ángulo de la rueda:

$$\frac{\partial L}{\partial \theta_R} = 0$$

(*Fórmula 2.26)*

Derivada del Lagrangiano con respecto a la velocidad angular de la rueda:

$$\frac{\partial L}{\partial \dot{\theta}_R} = (M_P + 3M_R) r^2 \dot{\theta}_R + M_P r L \dot{\theta}_P \cos(\theta_P)$$

(*Fórmula 2.27)*

Derivada del Lagrangiano con respecto a la velocidad angular de la rueda con respecto del tiempo:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}_R}\right) = (M_P + 3M_R) r^2 \ddot{\theta}_R + M_P r L \ddot{\theta}_P \cos(\theta_P) - M_P r L \dot{\theta}_P^2 \sin(\theta_P)$$

(*Fórmula 2.28)*

Derivada del Lagrangiano con respecto al ángulo del péndulo:

$$\frac{\partial L}{\partial \theta_P} = -M_P r L \dot{\theta}_P \dot{\theta}_R \sin(\theta_P) + M_P g L \sin(\theta_P)$$

(*Fórmula 2.29)*

Derivada del Lagrangiano con respecto a la velocidad angular del péndulo:

$$\frac{\partial L}{\partial \dot{\theta_P}} = 2M_P L^2 \dot{\theta}_P + M_P r L \dot{\theta}_R \cos(\theta_P)$$

(*Fórmula 2.30)*

Derivada del Lagrangiano con respecto a la velocidad angular del péndulo con respecto del tiempo:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta_P}}\right) = 2M_P L^2 \ddot{\theta_P} + M_P r L \ddot{\theta_R} \cos(\theta_P) - M_P r L \dot{\theta}_R \dot{\theta}_P \sin(\theta_P)$$

(*Fórmula 2.31)*

Sustituyendo todos los términos en la ecuación 2.25:

$$(M_P + 3M_R) r^2 \ddot{\theta_R} + M_P r L \ddot{\theta_P} \cos(\theta_P) - M_P r L \dot{\theta}_P^2 \sin(\theta_P) + 2M_P L^2 \ddot{\theta_P} + M_P r L \ddot{\theta_R} \cos(\theta_P) - M_P r L \dot{\theta}_R \dot{\theta}_P \sin(\theta_P) + M_P r L \dot{\theta}_P \dot{\theta}_R \sin(\theta_P) - M_P g L \sin(\theta_P) = 0$$

(*Fórmula 2.32)*

Para operar más fácilmente utilizaremos las siguientes sustituciones:

$$a = \left(\frac{3}{2} M_R + \frac{1}{2} M_P\right) r^2$$

(*Fórmula 2.33)*

$$b = M_P L^2$$

(*Fórmula 2.34)*

$$c = r L M_P$$

(*Fórmula 2.35)*

$$d = M_P g L$$

(*Fórmula 2.36)*

Obteniendo finalmente el modelo de nuestro sistema:

$$F = \ddot{\theta_R}(2a + c\ \cos(\theta_P)) + \ddot{\theta_P}(c\ \cos(\theta_P) + 2b) - c\ \dot{\theta}_P^2\ \sin(\theta_P) - d\ \sin(\theta_P) = 0$$

(*Fórmula 2.37)*

4 Control de la posición y del desplazamiento

El objetivo de este proyecto, como se expuso anteriormente, es crear un dispositivo capaz de mantenerse vertical a pesar de que el centro de masas esté situado por encima de su punto pivote, tratándose de un sistema de naturaleza inestable. Para conseguir esta meta, se ha partido del uso de un controlador del tipo PID, cuyas siglas significan Proporcional, Integral y Derivativo. Se ha optado por

este controlador ya que es uno de los más sencillos y eficientes a la hora de su implementación. A su vez, es el controlador que más se estudia en las asignaturas de grado dentro de la titulación de ingeniería electrónica industrial.

El controlador PID fue implementado por primera vez por el ingeniero Nicolas Minorsky en el año 1922. Minorsky desarrolló este tipo de sistema de control para poder automatizar la dirección de las naves de la Armada de los Estados Unidos. En el transcurso de su trabajo se dio cuenta de que, para controlar el timón de dichas naves, no sólo bastaba con conocer el error actual de éste, sino que teniendo en cuenta el error pasado, así como la tasa de cambio, mejoraba mucho su comportamiento.

Como se ha expuesto anteriormente, es necesario tener en cuenta el error que sufre el sistema en cada momento. Para esto, se implementa lo que se denomina un control en lazo cerrado o realimentado ya que, en un control de lazo abierto, no podríamos saber en qué estado se encuentra nuestro sistema.

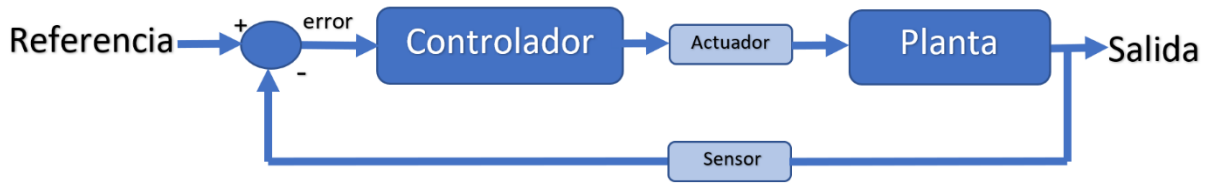


Ilustración 4-1: Control en lazo cerrado

Ilustración 4-2: Control en lazo abierto

Un controlador PID, consta de tres componentes:

- Componente Proporcional: Corresponde con el producto entre la señal de error y una constante a la que se le denomina ganancia proporcional. En esta componente no se considera el tiempo transcurrido, es decir, sólo depende del error actual manteniendo una relación lineal con este.

$$P = K_p \cdot e(t)$$

(*Fórmula 3.1)*

- Componente Integral: Corresponde con el producto entre la integral del error y una constante llamada ganancia integral. Gracias a esta componente se puede llegar a eliminar el error en estado estacionario ya que actúa siempre que exista dicho error. Como integra el error a lo largo del tiempo, podemos decir que responde ante el error acumulado en el tiempo transcurrido.

$$I = k_i \cdot \int e(t)\, dt$$

(*Fórmula 3.2)*

- Componente Derivativa: Corresponde con el producto entre la derivada del error y una constante llamada ganancia derivativa. Esta componente actúa en el momento en que cambia el error, anticipándose a él. El principal objetivo es disminuir el error a la misma velocidad que se produce.

$$D = k_d \cdot \frac{de(t)}{dt}$$

(*Fórmula 3.3)*

En la siguiente tabla podemos ver cómo variará la respuesta del sistema en función de las distintas componentes:

	P	I	D
Velocidad respuesta	Aumenta	Aumenta	Aumenta
Error estacionario	No elimina	Elimina	No elimina
Estabilidad	Disminuye	Disminuye	Aumenta hasta cierto punto
Sobreimpulso	Aumenta	Aumenta	Disminuye
Tiempo establecimiento	Poco cambio	Aumenta	Disminuye

Tabla 4-1: Acciones PID

Finalmente, combinando estas tres acciones, la forma en que quedaría el controlador sería la siguiente:

$$PID = K_p \cdot e(t) + k_i \cdot \int e(t)\,dt + k_d \cdot \frac{de(t)}{dt}$$

(*Fórmula 3.4*)

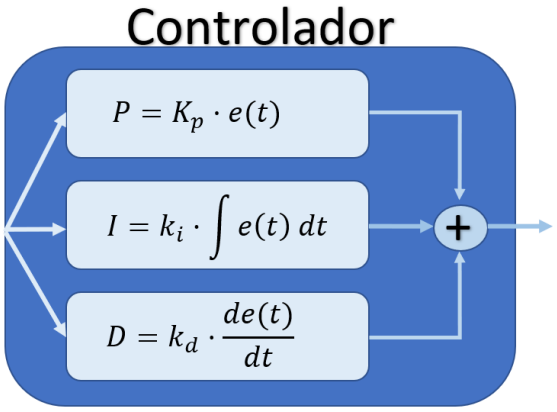


Ilustración 4-3: Control PID

5 Diseño e implementación real del vehículo

En este capítulo se enumerarán los distintos materiales que se han empleado para la realización de este proyecto. Se comenzará con una pequeña introducción teórica analizando posibles alternativas y finalmente se expondrá el por qué se ha elegido un determinado dispositivo. A su vez, se expondrá toda la información necesaria para poder construir el vehículo y entender más a fondo cómo funcionan cada uno de los componentes que intervienen en el proyecto.

5.1 Arduino

Arduino es una plataforma de hardware y software libre (open-source), relativamente fácil de utilizar y destinada a que cualquier persona sea capaz de crear proyectos de electrónica sin necesidad de tener conocimientos avanzados en esta rama de la ingeniería, permitiendo un prototipado rápido para una solución deseada.

Ilustración 5-1: Logo Arduino. Fuente

Aparte de las características hardware y software que presentan las placas, Arduino se caracteriza por una amplia comunidad que apoya esta plataforma compartiendo conocimiento, gracias a esto podemos encontrar infinidad de información, tutoriales, librerías, etc.

5.1.1 Hardware

La mayoría de las placas Arduino constan de un microcontrolador AVR Atmel-8 bits (ATmegaXXX). Estos son una familia de microcontroladores RISC de 8 bits de la empresa ATMEL con arquitectura Harvard, siendo el principal competidor de los PIC de 8 bits. Estos microcontroladores incluyen en su interior las tres principales unidades funcionales de una computadora: unidad central de procesamiento, memoria y periféricos de entrada/salida. La mayoría de las placas incluyen un regulador lineal de 5 V y un oscilador de cristal de 16 MHz.

Además, existe una gran cantidad de módulos adicionales denominados shields que aumentan las características técnicas de la placa Arduino debido a que poseen circuitos específicos que añaden una o más funcionalidades extras a la placa Arduino nativa.

5.1.2 Software

Existe un IDE (*Integrated Develpment Enviroment*) gratuito que, mediante el uso de librerías de funciones muy sencillas, permite el desarrollo de software en C para estos microcontroladores. La interfaz de este IDE se puede observar en la ilustración 5.2

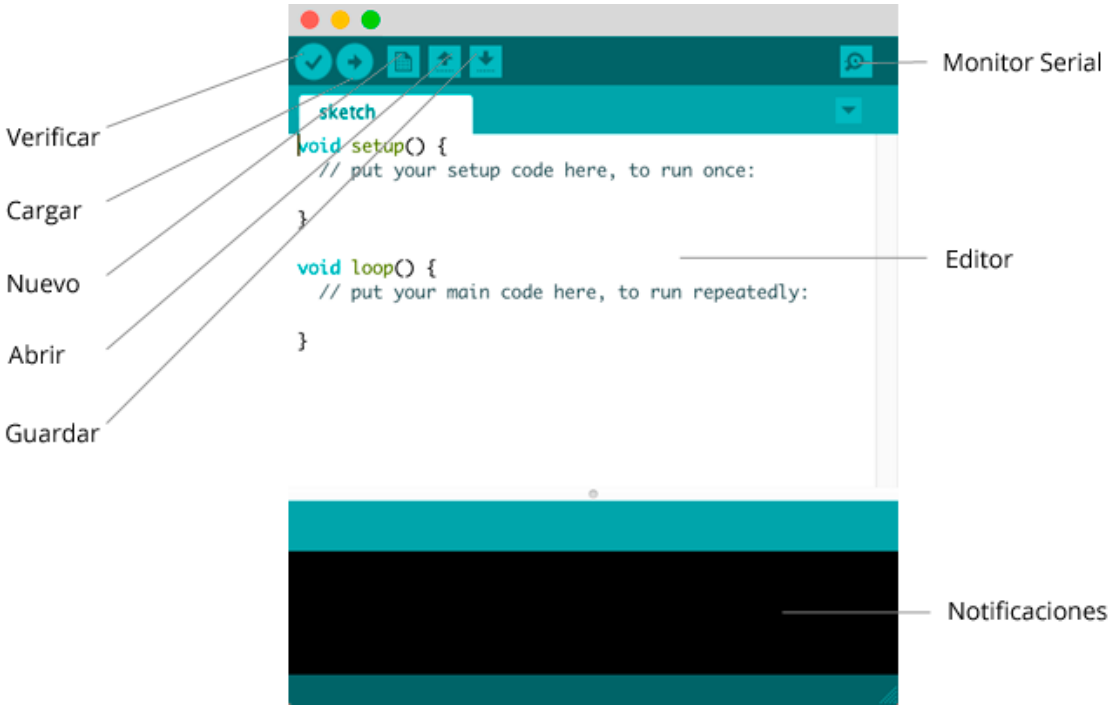


Ilustración 5-2: IDE Arduino

Este software incluye todos los drivers necesarios para cualquier placa Arduino, así como las librerías necesarias para su programación. A pesar de todas sus ventajas, el IDE también tiene una desventaja importante y es que no incorpora un debugger, complicándose la detección de errores.

5.1.3 Ventajas de Arduino

En el mercado existen infinidad de microcontroladores, pero Arduino presenta algunas ventajas para este proyecto:

1. Una placa Arduino incorpora tanto el microcontrolador como todos los elementos necesarios para su funcionamiento y programación. Por ejemplo, en caso de usar un PIC sería necesario diseñar la PCB correspondiente y fabricarla.

2. El software de desarrollo de Arduino está publicado bajo una licencia libre, por lo que es totalmente gratuito.

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

4. Son placas relativamente económicas y suficientemente robustas para este tipo de proyectos.

5. Cuenta con un número suficiente de entradas/salidas mediante las cuales podremos conectar multitud de sensores, actuadores, dispositivos de comunicación, etc. Además de contar con un extenso catálogo de librerías de código abierto, permitiendo una programación sencilla.

Una vez escogida la familia Arduino queda un último paso ¿qué modelo elegir?

5.1.4 Elección Arduino Nano

Se ha optado por este modelo de placa Arduino debido a que en cuanto a funcionalidad es muy parecido al Arduino UNO (el modelo más extendido), teniendo el mismo número de entradas y salidas disponibles, pero es mucho más compacta, siendo esto una ventaja a la hora de implementarla en una placa de prototipo PCB. La única diferencia con el modelo UNO es que no posee conector para alimentación externa, y funciona con un cable USB Mini-B en vez del cable estándar USB B.

El Arduino Nano 3.0 puede ser alimentado usando el cable USB Mini-B, con una fuente externa no regulada de 6-20V (pin 30), o con una fuente externa regulada de 5V (pin 27). En la tabla 5.1 podemos ver sus características principales.

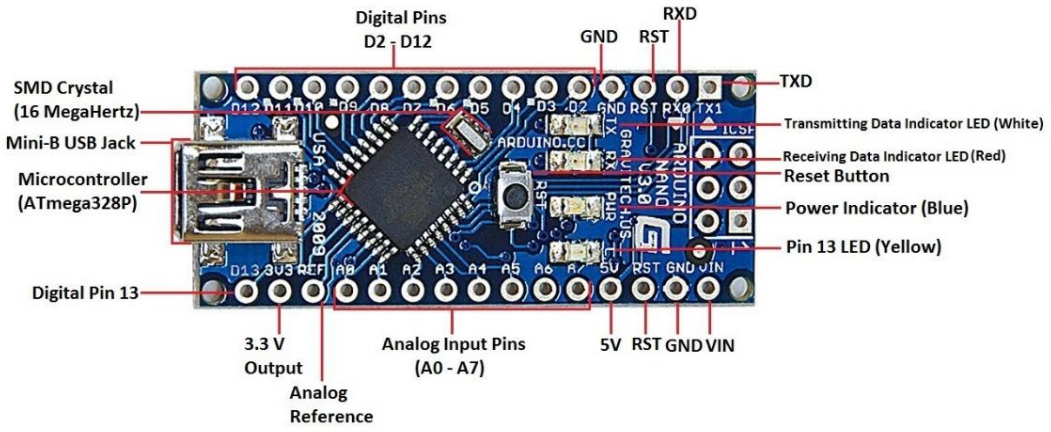


Ilustración 5-3: Arduino NANO

Arduino NANO	Características
Microcontrolador	ATmega328
Arquitectura	AVR
Tensión de trabajo	5 V
Memoria Flash	32 KB
SRAM	2 KB
Velocidad de reloj	16 MHz
Pines analógicos	8
EEPROM	1 KB
Corriente DC por pin	40 mA
Tensión de alimentación	7-12 V
Pines digitales	22 (6 de los cuales son PWM)
Consumo	19 mA
Tamaño PCB	18 x 45 mm
Peso	7 g

Tabla 5-1: Características Arduino NANO

5.2 Comunicación Bluetooth

Para poder controlar el vehículo de forma remota necesitamos algún sistema de comunicación inalámbrica con Arduino existiendo distintas alternativas. Una de ellas sería utilizar un transceptor RF (transmisor + receptor) a una frecuencia entre 2.4GHz a 2.5GHz, una banda libre para uso gratuito. Sin embargo, si optásemos por esta solución, sería necesario emplear dos placas Arduino y dos RF (uno haría de transmisor y otro de receptor) y además algún sistema externo de control como un mando, un joystick, botones, etc.

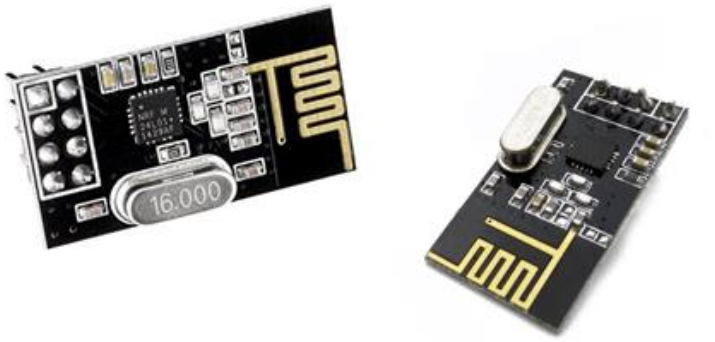


Ilustración 5-4: NRF24L01

Por otro lado, una alternativa interesante y es la que se ha elegido para este proyecto consiste en utilizar un módulo Bluetooth como el de la figura 5.5. Se denomina Bluetooth al protocolo de comunicaciones diseñado especialmente para dispositivos de bajo consumo, que requieren corto alcance de emisión y basados en transceptores de bajo costo. Se trata de una especificación industrial para 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. Los principales objetivos que se pretenden conseguir con esta norma son:

- Facilitar las comunicaciones entre equipos móviles.
- Eliminar los cables y conectores entre estos.
- Ofrecer la posibilidad de crear pequeñas redes inalámbricas y facilitar la sincronización de datos entre equipos personales.

A diferencia de los RF, sólo será necesario un módulo bluetooth conectado a la placa Arduino de nuestro vehículo. Además, no se necesitarán dispositivos externos de control ya que la comunicación se realizará a través de un Smartphone, gracias a una aplicación descargada directamente de Google Play Store.

En concreto utilizaremos el módulo HC-05 capaz de trabajar tanto en modo maestro como en modo esclavo. Este tiene dos pines por los cuales se realizará la comunicación:

- Rx: Recepción de datos.

- Tx: Transmisión de datos.

En Arduino existen dos pines específicos para este tipo de comunicaciones. Sin embargo, se ha preferido no usar estos pines, ya que su uso puede dar problemas. Por ello se empleará la librería de *ArduinoSoftwareSerial*, la cual permite definir por software los pines que harán de Rx y Tx.

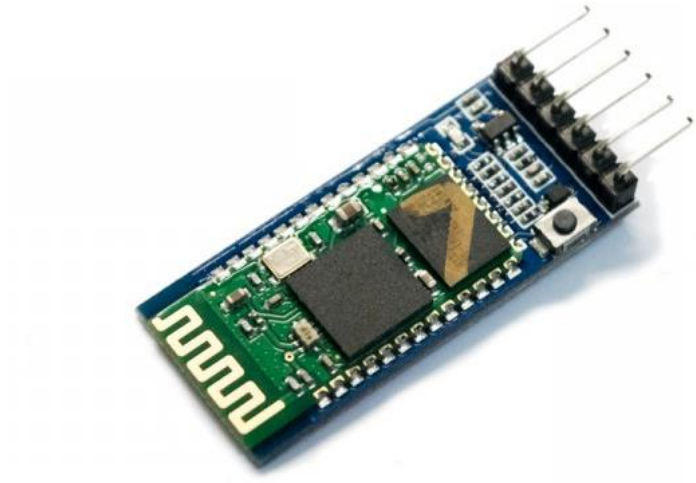


Ilustración 5-5: Módulo HC-05

HC-05	Características
Protocolo Bluetooth	v1.1 / 2.0.
Frecuencia	Banda ISM de 2,4 GHz
Modulación	GFSK
Potencia de transmisión	Menos de 4dBm, Clase 2
Sensibilidad	Menos de -84dBm en el 0,1% BER
Ratio asíncrono	2.1Mbps (Max) / 160 kbps
Ratio síncrono	1Mbps / 1Mbps
Modo	Maestro y esclavo
Alimentación	+ 3.3VDC 50mA (soporta de 3.3 a 6V)

Tabla 5-2: Características HC-05

5.3 Acelerómetro y giroscopio

Para poder controlar la inclinación y orientación del vehículo es necesaria una unidad de medición inercial o IMU (Inertial Measurment Units). Estas unidades suelen estar compuestas por un acelerómetro y un giroscopio. Examinemos más detalladamente en qué consisten estos sensores.

5.3.1 Acelerómetro

Como sabemos, la aceleración es la variación de la velocidad por unidad de tiempo. Así mismo la segunda ley de Newton establece que en un cuerpo con masa constante, la aceleración del cuerpo es proporcional a la fuerza que actúa sobre este. En estos conceptos se basan los acelerómetros para medir aceleración. Los acelerómetros internamente tienen un MEMS (MicroElectroMechanical Systems) que de forma similar a un sistema masa resorte permite medir la aceleración.

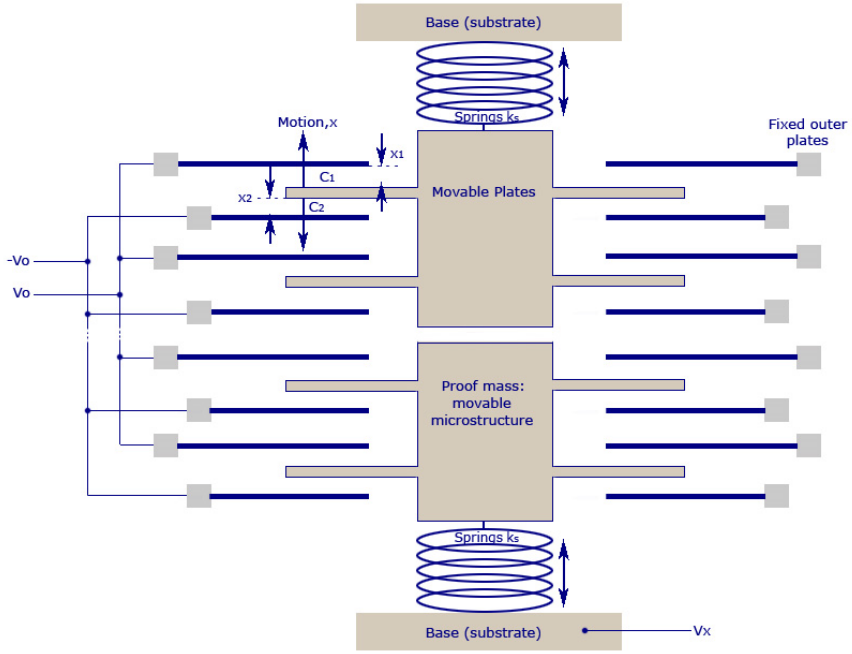


Ilustración 5-6: Acelerómetro MEMS. Fuente

5.3.2 Giroscopio

La velocidad angular es la variación del desplazamiento angular por unidad de tiempo, es decir, cómo de rápido gira un cuerpo alrededor de su eje. Al igual que el acelerómetro, estos sensores se componen internamente de un MEMS pero esta vez no se basa en un sistema masa-amortiguador sino en el efecto Coriolis. Conociendo a qué velocidad gira un cuerpo, y el tiempo que ha estado girando, podemos obtener de forma relativamente sencilla cuántos grados ha rotado dicho cuerpo.

5.3.3 Elección sensor MPU-6050

El sensor MPU-6050 es una IMU con 6 grados de libertad (DoF) ya que combina un acelerómetro de 3 ejes junto con un giroscopio de 3 ejes. Este sensor es muy utilizado en navegación, goniometría, estabilización, etc.

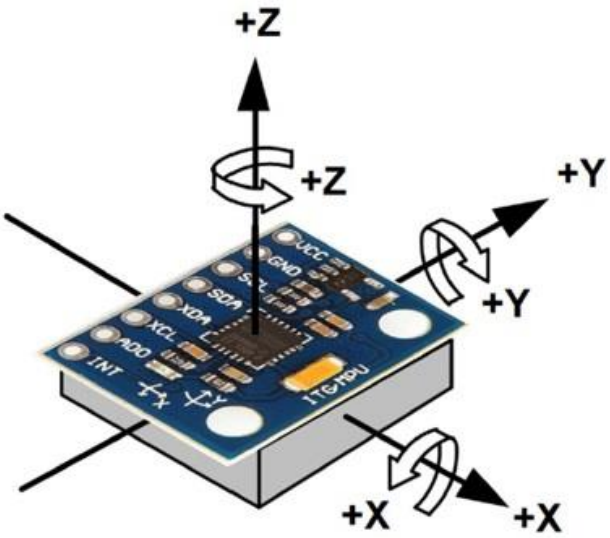


Ilustración 5-7: Orientación MPU-6050.

Se ha elegido este sensor ya que es perfectamente compatible con Arduino y tiene las características necesarias para nuestro proyecto, siendo económico y existiendo multitud de librerías e información disponible.

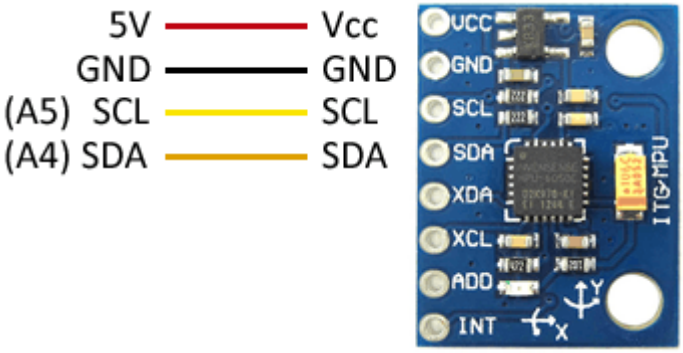


Ilustración 5-8: MPU-6050. Fuente

Para establecer la comunicación entre el sensor y Arduino se utiliza el protocolo I2C. Estas siglas significan Circuito Interintegrado (Inter-Integrated Circuit) y es un protocolo de comunicación serial desarrollado por Phillips Semiconductors. Este protocolo toma e integra lo mejor de los protocolos SPI y UART pudiendo tener a varios maestros controlando uno o múltiples esclavos. Para la comunicación sólo son necesarias dos vías o cables:

- SDA – Serial Data. Es la vía de comunicación entre el maestro y el esclavo a través de la cual se envían los datos.

- SCL – Serial Clock. Es la vía por donde viaja la señal de reloj.

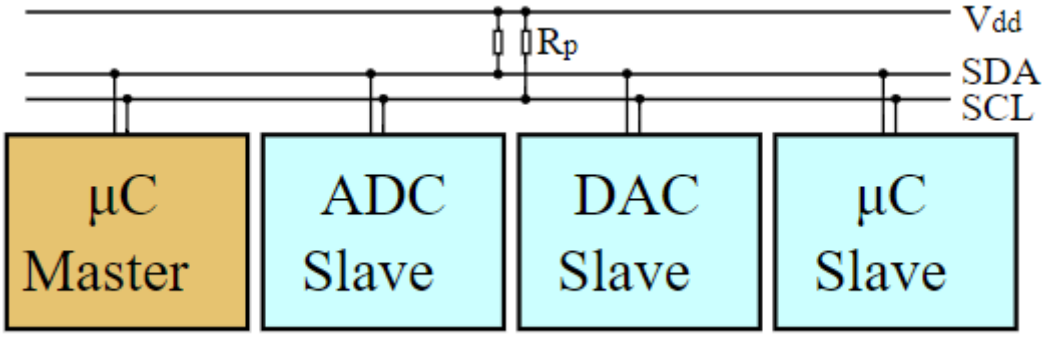


Ilustración 5-9: Protocolo I2C. Fuente

Arduino cuenta con su propio soporte I2C pudiendo acceder a las vías SDA y SCL desde los pines A4 y A5 respectivamente (Ilustración 5.8). También, es común emplear la librería "*Wire.h*", ya que simplifica todo el proceso de comunicación siendo algunos de sus registros más importantes:

Registro	Nombre	Función
TWCR	Two Wire Control Register	Controla las acciones del módulo TWI
TWSR	Two Wire Status Register	Informa del estado del TWI
TWDR	Two Wire Data Register	Contiene los datos
TWBR	Two Wire Bit Rate Register	Controla la frecuencia de reloj

Tabla 5-3: Registros Wire.h

En el datasheet u hoja de características del MPU, encontramos que la velocidad de comunicación I2C es de 400 kHz. Esta velocidad se puede configurar

mediante el registro TWBR a través de la siguiente ecuación, que podemos encontrarla en el datasheet del ATMEGA328.

$$Frecuencia\ de\ reloj = \frac{frecuencia\ de\ reloj\ CPU}{16 + 2 \cdot (TWBR) \cdot Prescaler}$$

(*Fórmula 5.1)*

Despejando:

$$TWBR = \frac{\frac{16 \cdot 10^6}{400 \cdot 10^3} - 16}{2 \cdot 1} = 12$$

(*Fórmula 5.2)*

5.4 Obtención del ángulo con el sensor MPU – 6050

Este sensor no nos proporciona directamente una medida de inclinación en grados, sino que nos ofrece una serie de valores para cada uno de sus seis grados de libertad. Estos valores son valores “crudos”, es decir, son valores que tienen que ser manipulados para tras una serie de operaciones, finalmente obtener el ángulo de inclinación con respecto al eje deseado.

El primer paso es iniciar la configuración del sensor mediante una serie de registros. En el código, todo esto se hace dentro de una rutina llamada “setup_registros_mpu6050()”.

```
/**
   RUTINA QUE CONFIGURA E INICIALIZA MPU6055
*/
void setup_registros_mpu6050() {
  //Activamos MPU-6050
  Wire.beginTransmission(0x68);//Empezamos comunicación (0x68 es la dirección del
                               //MPU-6050)
  Wire.write(0x6B);            //Escribimos en el registro 0x6B
  Wire.write(0x00);            //Ponemos todos los bits a 0
  Wire.endTransmission();      //Terminamos comunicación
  //Configuramos un filtro para eliminar posible ruido
  Wire.beginTransmission(0x68);        //Empezamos comunicación
  Wire.write(0x1A);                    //Escribimos en el registro 0x1A
  Wire.write(0x03);           //Seleccionamos filtro paso baja de ~43Hz  (00000011)
  Wire.endTransmission();              //Terminamos comunicación
  //Establecemos sensibilidad del giroscopio(250dps)
  Wire.beginTransmission(0x68);        //Empezamos comunicación
  Wire.write(0x1B);                    //Escribimos en el registro 0x1B
  Wire.write(0x00);              //Seleccionamos sensibilidad de 250dps (00000000)
  Wire.endTransmission();              //Terminamos comunicación
```

```
  //Establecemos sensibilidad del acelerómetro(+/-4g)
  Wire.beginTransmission(0x68);         //Empezamos comunicación
  Wire.write(0x1C);                     //Escribimos en el registro 0x1C
  Wire.write(0x08);                  //Seleccionamos sensibilidad de +/-4g (00001000)
  Wire.endTransmission();               //Terminamos comunicación
}
```

En esta rutina se han utilizado los siguientes registros:

- El registro 0x1A. Con este registro podemos configurar un Filtro Digital Pasa Baja (DLPF) tanto para el giroscopio como para el acelerómetro. Con este filtro conseguimos eliminar algo de ruido debido a las vibraciones del sensor obteniendo unas medidas más exactas.

Registro (HEX)	Registro (Decimal)	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1A	26	-	-	EXT_SYNC_SET[2:0]			DLPF_CFG[2:0]		

Tabla 5-4: Registro 1A

	ACELERÓMETRO		GIROSCOPIO		
DLPF_CFG	BW (Hz)	Delay (ms)	BW (Hz)	Delay (ms)	Fs (kHz)
0	260	0	256	0.98	8
1	184	2.0	188	1.9	1
2	94	3.0	98	2.8	1
3	44	4.9	42	4.8	1
4	21	8.5	20	8.3	1
5	10	13.8	10	13.4	1
6	5	19.0	5	18.6	1
7	-	-	-	-	8

Tabla 5-5: Selección de la frecuencia del filtro

- El registro 0x1B. Con este registro establecemos la sensibilidad del giroscopio. Como nuestro vehículo no va a girar a gran velocidad sobre un eje, con la mínima sensibilidad será suficiente. Esta establece que el giroscopio será capaz de medir velocidades angulares de hasta 250 º/s.

Registro (HEX)	Registro (Decimal)	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1B	27	XG_ST	YG_ST	ZG_ST	FS_SEL[1:0]		-	-	-

Tabla 5-6: Registro 1B

FS_SEL	FULL SCALE RANGE	UNITS
0	± 250 º/s	131 LSB/(º/s)
1	± 500 º/s	65.5 LSB/(º/s)
2	± 1000 º/s	32.8 LSB/(º/s)
3	± 2000 º/s	16.4 LSB/(º/s)

Tabla 5-7: Selección sensibilidad giroscopio

- El registro 0x1C. Con este registro establecemos la sensibilidad del acelerómetro. Elegimos el valor ± 4 g ya que nos da valores suficientemente precisos y es con el que mejores resultados se han obtenido a la hora de calcular el ángulo de inclinación para nuestro proyecto.

Registro (HEX)	Registro (Decimal)	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1C	28	XA_ST	YA_ST	ZA_ST	AFS_SEL[1:0]		-	-	-

Tabla 5-8: Registro 1C

AFS_SEL	FULL SCALE RANGE
0	± 2 g
1	± 4 g
2	± 8 g
3	± 16 g

Tabla 5-9: Selección sensibilidad acelerómetro

Finalmente, nuestra configuración quedará de la siguiente manera:

Variable	Valor mínimo	Valor central	Valor máximo
Datos MPU	- 32768	0	+ 32767
Aceleración	- 4 g	0 g	+ 4 g
Velocidad angular	- 250 º/s	0 º/s	+ 250 º/s

Tabla 5-10: Configuración MPU6050

Una vez tenemos configurados los registros del MPU-6050, podemos proceder a calcular el ángulo de inclinación. Para esto, dentro del código se ha creado una rutina llamada “datos_mpu6050()”.

```
/**
   RUTINA QUE LEE LOS VALORES "CRUDOS" DEL MPU6050 Y CALCULA EL ÁNGULO DE
INCLINACIÓN
*/
void datos_mpu6050() {
  Wire.beginTransmission(0x68);       //Empezamos comunicación
  Wire.write(0x3B);                   //Empezamos por el registro 0x3B (ACCEL_XOUT)
  Wire.endTransmission();             //Terminamos comunicación
  Wire.requestFrom(0x68, 6); //Obtenemos los primeros 6 registros a partir del 0x3B
  ac_x = Wire.read() << 8 | Wire.read(); //Combinamos los registros 0x3B y 0x3C
                                         //obteniendo la aceleración en X
  ac_y = Wire.read() << 8 | Wire.read(); //Combinamos los registros 0x3D y 0x3E
                                         //obteniendo la aceleración en Y
  ac_z = Wire.read() << 8 | Wire.read(); //Combinamos los registros 0x3F y 0x40
                                         //obteniendo la aceleración en Z

  //Ángulo acelerómetro
  ang_ac = atan(ac_z / sqrt(pow(ac_x,2) + pow(ac_y,2)))*rad_a_gra;
  Wire.beginTransmission(0x68);     //Empezamos comunicación
  Wire.write(0x45);                 //Empezamos por el registro 0x45 (GYRO_YOUT_H)
  Wire.endTransmission();           //Terminamos comunicación
  Wire.requestFrom(0x68, 2);        //Obtenemos los dos registros siguientes a 0x45
  giro_y = Wire.read() << 8 | Wire.read();//Combinamos los registros 0x45 y 0x46
                                          //obteniendo el ángulo en Y

  //Ángulo giroscopio
  dt = (millis() - tiempo_prev);
  tiempo_prev = millis();
  giro_y += offset;                       //Offset
  ang_giro = angulo + (giro_y/131)*dt/1000.0;

  //Filtro complementario
  angulo = 0.95*ang_giro + 0.05*ang_ac;

  //Filtro complementario para tener en cuenta los valores anteriores del ángulo y
  //eliminar ruido y vibraciones del vehículo
  angulo_nuevo = angulo*coeficiente + angulo_anterior*(1-coeficiente);
  angulo_anterior = angulo_nuevo;
}
```

Como podemos ver en el código, primero obtenemos los datos sin procesar del MPU, combinando distintos registros. Los datos tienen una resolución de 16 bits divididos en dos registros de 8 bits cada uno. En la siguiente tabla aparecen recogidos en qué registros se almacenan los datos relativos a los 6 grados de libertad del sensor.

Addr (Hex)	Addr (Dec.)	Nombre del registro	Serial I/F	Bit 7…0
3B	59	ACCEL_XOUT_H	R	ACCEL_XOUT[15:8]
3C	60	ACCEL_XOUT_L	R	ACCEL_XOUT[7:0]
3D	61	ACCEL_YOUT_H	R	ACCEL_YOUT[15:8]
3E	62	ACCEL_YOUT_L	R	ACCEL_YOUT[7:0]
3F	63	ACCEL_ZOUT_H	R	ACCEL_ZOUT[15:8]
40	64	ACCEL_ZOUT_L	R	ACCEL_ZOUT[7:0]
41	65	TEMP_OUT_H	R	TEMP_OUT[15:8]
43	66	TEMP_OUT_L	R	TEMP_OUT[7:0]

43	67	GYRO_XOUT_H	R	GYRO_XOUT[15:8]
44	68	GYRO_XOUT_L	R	GYRO_XOUT[7:0]
45	69	GYRO_YOUT_H	R	GYRO_YOUT[15:8]
46	70	GYRO_YOUT_L	R	GYRO_YOUT[7:0]
47	71	GYRO_ZOUT_H	R	GYRO_ZOUT[15:8]
48	72	GYRO_ZOUT_L	R	GYRO_ZOUT[7:0]

Tabla 5-11: Registros de datos

Una vez obtenidos los datos "crudos" del sensor, es necesario manipularlos para obtener finalmente el ángulo de inclinación en grados.

5.4.1 Cálculo del ángulo de inclinación con el acelerómetro

Teniendo en cuenta que la única fuerza que actúa sobre el sensor es la fuerza de la gravedad, los valores que obtengamos del acelerómetro corresponderán a la medida de dicha fuerza. Como la dirección de la gravedad es siempre vertical, el ángulo de inclinación será el ángulo de la resultante respecto al plano del sensor. En la figura 5.10 se puede ver un ejemplo para el caso en el que sólo nos interesasen dos dimensiones, por lo que el ángulo se correspondería con:

$$\theta = \tan^{-1}\left(\frac{g_x}{g_z}\right)$$

(*Fórmula 5.3)*

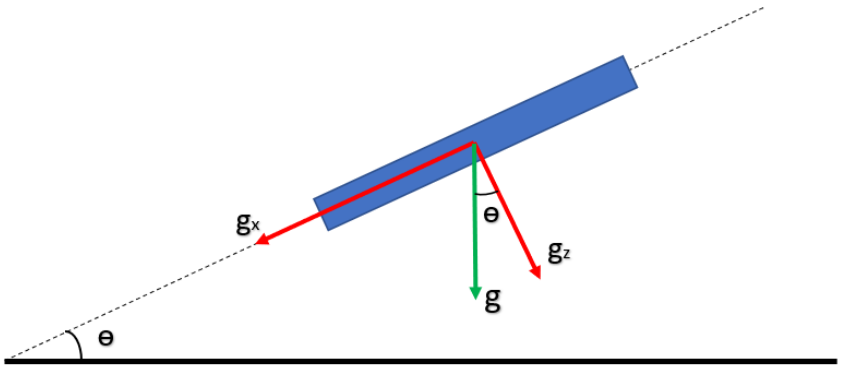


Ilustración 5-10: Fuerzas en plano 2D

Sin embargo, en nuestro caso nos encontramos ante un problema en el que se requieren las tres dimensiones del espacio, por lo que la ecuación para obtener el ángulo quedaría:

$$\theta = \tan^{-1}\left(\frac{g_z}{\sqrt{g_x^2 + g_y^2}}\right)$$

(*Fórmula 5.4)*

Como podemos ver en el código, usamos un factor de conversión “rad_a_gra = 57.3248” para pasar de radianes a grados y así sea más fácil manejar estos datos posteriormente.

5.4.2 Cálculo del ángulo de inclinación con el giroscopio

El giroscopio nos entrega como dato la velocidad angular con la que se rota sobre un eje. Por este motivo, para calcular el ángulo girado es necesario integrar dicha velocidad conociendo el ángulo inicial.

$$\theta_{giro} = \theta_0 + \omega \cdot \Delta t$$

(*Fórmula 5.5)*

En el código, dividimos por 131 ya que este fue el valor elegido para la sensibilidad del giroscopio. Otro punto a tener en cuenta es que el sensor, viene con un pequeño offset de fábrica y además es muy difícil que esté montado perfectamente centrado en el vehículo. Por este motivo, una vez se implementó el sensor en el vehículo y poniendo este en posición totalmente vertical, se calculó dicho offset para que pudiera ser compensado y no obtuviéramos medidas erróneas.

5.4.3 Combinación de ambos ángulos mediante un filtro complementario.

Al obtener la medida del giroscopio podemos observar que no es exacta incluso cuando el sensor permanece completamente inalterado. Esto se debe a que al integrar la velocidad hay un pequeño error como consecuencia de la mala medición del tiempo o del propio ruido del sensor. Este error, se va acumulando con

el tiempo, fenómeno que se conoce como Drift o Deriva lo que hace que la medida del giroscopio sea inservible.

Existen varias soluciones para compensar este fenómeno, siendo la solución óptima la aplicación de un filtro Kalman. Pero esta solución tiene un inconveniente y es que para aplicar este filtro se necesita una gran capacidad de computación debido a su complejidad matemática. Por este motivo, para este proyecto se ha optado por un filtro complementario, siendo este lo suficientemente preciso para nuestra aplicación.

Si sólo trabajásemos con el valor del acelerómetro, los valores serían susceptibles a aceleraciones producto de la vibración del MPU, aunque en tiempos largos el ángulo no acumula errores por lo que evitaríamos el problema de la deriva. Por otro lado, si sólo tomamos los valores del giroscopio, tendríamos el problema del Drift, aunque siendo este bastante menos susceptible a errores debidos a vibraciones o fuerzas externas. El filtro complementario o en inglés "Complementary Filter" es aquel que combina las medidas obtenidas por el giroscopio con las obtenidas por el acelerómetro mediante un parámetro de ponderación.

$$ángulo = 0.95 \cdot ángulo_{giroscopio} + 0.05 \cdot ángulo_{acelerómetro}$$

(Fórmula 4.6)

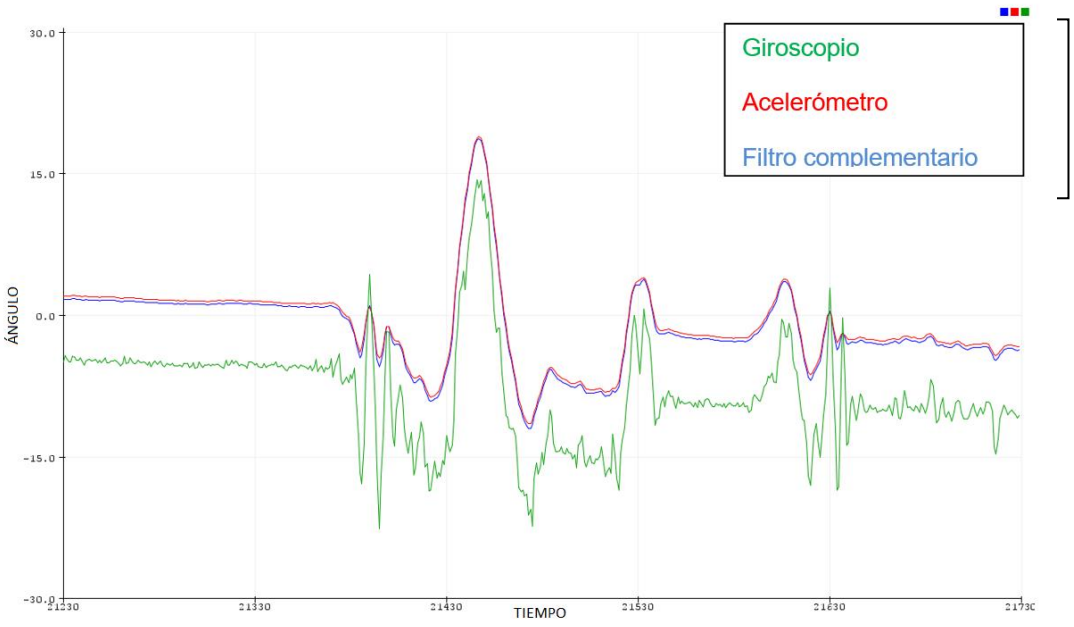


Ilustración 5-11: Efecto filtro complementario

Gracias a esto, las medidas del giroscopio pasan a través de un filtro pasa alta, eliminando el problema del Drift, mientras que las medidas del acelerómetro pasan a través de un filtro pasa baja, amortiguando variaciones bruscas en sus medidas.

5.5 Sistema de propulsión

Existen distintas posibilidades para impulsar el vehículo. Una solución consistiría en utilizar motores de corriente continua (DC motors) o bien, se podría optar por motores paso a paso (stepper motors).

5.5.1 Motor de corriente continua

El motor de corriente continua es una máquina que convierte energía eléctrica en energía en mecánica, provocando un movimiento rotatorio, gracias a la acción de un campo magnético.

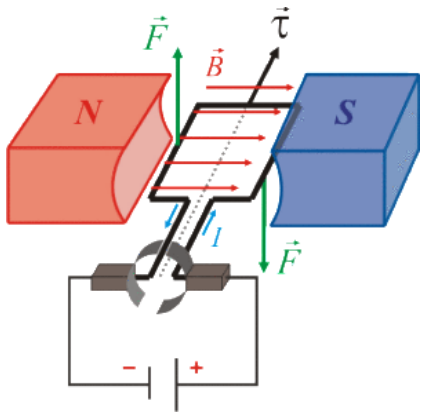


Ilustración 5-12: Esquema de funcionamiento motor DC. Fuente

Se compone principalmente de dos partes. El estátor es la parte fija que da soporte al motor, éste contiene los polos de la máquina los cuales pueden ser o devanados de hilo de cobre sobre un núcleo de hierro, o bien imanes permanentes. El rotor es el componente que gira (rota) generalmente de forma cilíndrica, también devanado y con núcleo. Este se alimenta con corriente continua a través de unos dispositivos denominados delgas, los cuales están en contacto alternante con escobillas fijas (también llamadas carbones). Aunque también podemos encontrar

motores de CC sin escobillas (brushless en inglés) utilizados en aeromodelismo por su bajo par motor y su gran velocidad.

Este tipo de motores fueron los primeros en utilizarse en vehículos eléctricos por sus buenas características en tracción y por la simplicidad de los sistemas de control de consumo desde las baterías. Este tipo de motores son imprescindibles para aplicaciones en las que se precisen controles de par y velocidad.

Por lo tanto, en un motor de corriente continua prima el control de la velocidad y el par frente al control de la posición. Si queremos controlar dicha variable sería necesario incorporar elementos externos como encoders o codificadores rotatorios.

5.5.2 Motor paso a paso

Estos motores convierten una serie de impulsos eléctricos en desplazamientos angulares discretos, lo que significa que es capaz de girar una cantidad de grados (paso o medio paso) dependiendo de sus entradas de control. Por este motivo son motores ideales para el control de la posición con gran precisión.

Al igual que los de corriente continua están formados por un rotor y un estátor con la peculiaridad de que el estátor presenta un mecanizado en forma de dientes mientras que el rotor presenta unos imanes que se alternan entre norte/sur, habiendo tantos imanes como muescas en el estátor.

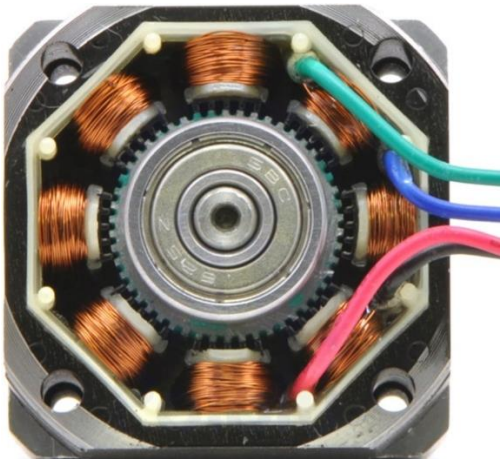


Ilustración 5-13: Interior motor paso a paso bipolar

Los motores paso a paso necesitan un circuito digital para moverse, de forma que nuestro controlador envía exactamente los pasos que queremos, sabiendo de manera precisa el ángulo que va a girar el motor. Además de la posición, nuestro programa puede controlar la velocidad a la que enviamos los pulsos. De esta forma, podemos acelerar y decelerar de forma controlada en todo momento. Otra característica que los diferencia de los motores anteriormente citados es que los motores paso a paso pueden quedar fijos en una posición. (Para que un motor de corriente continua se mantenga quieto, hay que recalcular la posición en un bucle permanente).

Por otro lado, estos motores presentan ciertas desventajas frente a los de corriente continua y es que son poco eficientes, consumiendo más amperios. Además, son motores lentos ya que, para dar una vuelta completa, tienen que recorrer todos los pasos uno a uno.

Existen dos tipos de motores paso a paso:

• Unipolares, en cuyo interior tienen dos pares de bobinas. Estos son los motores más sencillos de programar, pero son menos potentes y no gestionan tan bien la potencia como los bipolares.

• Bipolares, los cuales son más sencillos que los motores unipolares en cuanto a construcción física ya que únicamente tienen 2 bobinas. Sin embargo, la complejidad de estos motores reside en el driver, porque no sólo tiene que permitir pasar la corriente por la bobina, sino que tiene que cambiar la polaridad de la corriente continuamente.

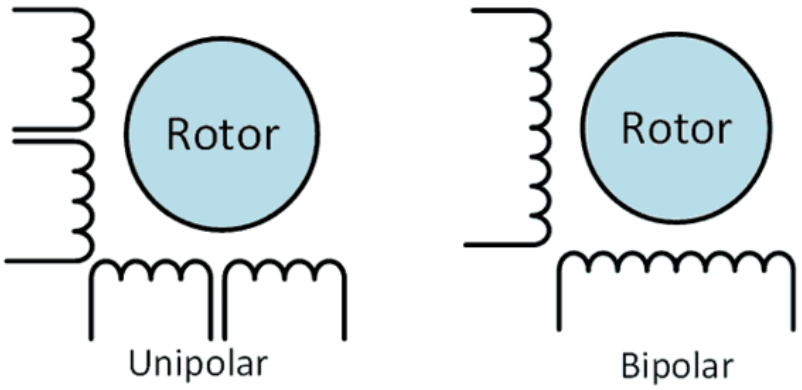


Ilustración 5-14: Esquema motor unipolar y bipolar.

5.5.3 Elección motor paso a paso NEMA 17HS4401

Analizadas las dos posibles opciones en cuanto a sistemas de propulsión se refiere, para este proyecto utilizaremos un motor paso a paso por los siguientes motivos:

- Precisión en el control de la posición y velocidad. Si utilizáramos motores de corriente continua necesitaríamos encoders con sus respectivos drivers lo que dificultaría el sistema de control.

- Los drivers suelen ser más avanzados, gestionan mejor la potencia, e incluyen características más avanzadas como micropasos.

- No es necesario una reductora para reducir la velocidad del motor ya que no son tan rápidos como los de corriente continua.

Para nuestro proyecto utilizaremos el motor NEMA 17HS4401. NEMA corresponde a la Asociación Nacional de Fabricantes Electrónicos o National Electrical Manufacturers Association en inglés. Esta asociación se encarga de realizar estándares industriales aplicados al campo de la electricidad y de la electrónica. En el caso particular, el nombre NEMA 17, hace referencia a un motor paso a paso con un encapsulado de 1.7 x 1.7 pulgadas (43.18 x 43.18 mm) de área transversal. Esta área se requiere que sea estandarizada ya que es donde se suelen atornillar los motores. La longitud de estos puede variar dependiendo de varios factores como la corriente o el torque.

Aparte del tamaño, existen distintos criterios de selección, siendo los más importantes:

- La corriente o intensidad nominal: aparecerá como “Rated Current”, “Phase Current” o “Max Current” y nos da el valor máximo de corriente que podemos hacer circular de manera continua por el motor. Cuanta mayor corriente nominal, más conserva su par el motor a altas velocidades. Además, tendremos que tenerla en cuenta para la selección de los drivers y las baterías. En nuestro caso 1.7 A.

- El número de pasos: cuantos más pasos tiene el motor, vamos a conseguir más precisión, pero menos velocidad. Los motores más comunes suelen tener 200 pasos por vuelta. Esto es 360º/200 = 1. 8º por cada paso.

- El par motor (o torque) de retención: Aparecerá como “Holding Torque” o “Static moment”. Se mide en N·m o N·cm y es uno de los parámetros que indica la fuerza del motor. Nuestro motor presenta un par de 40 N·cm.

En concreto, el modelo elegido es ampliamente utilizado en muchas otras aplicaciones como impresoras 3D, CNC, cortadoras, etc. Esto es debido a sus buenas prestaciones y a su bajo precio. Además, existen multitud de drivers e información disponible para este modelo.

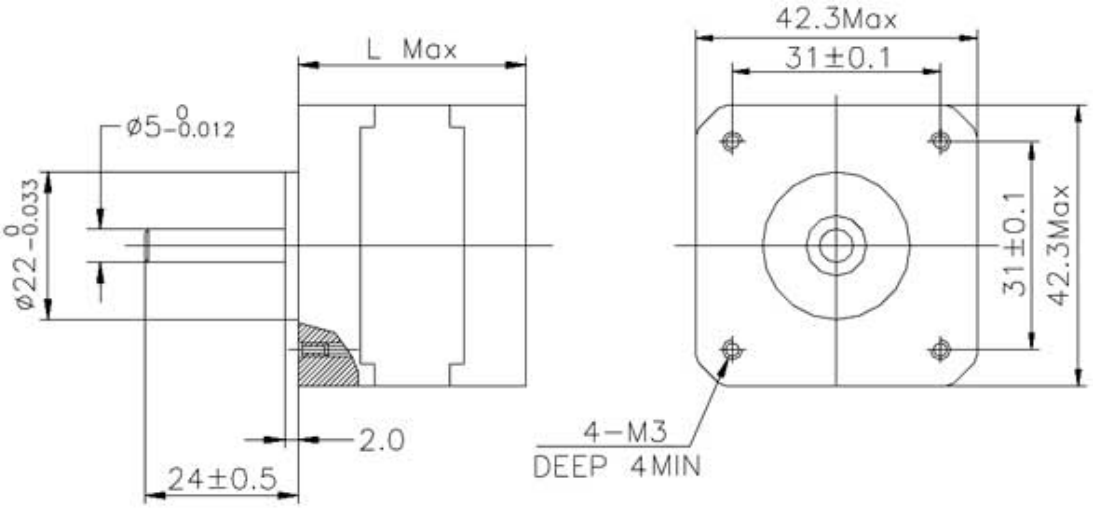


Ilustración 5-15: Dimensiones 17HS4401

17HS4401	Características
Ángulo de paso	1.8º
Longitud del motor	40 mm
Corriente nominal	1.7 A
Resistencia de la fase	1.5 Ω
Inductancia de la fase	2.8 mH
Torque de aguante	40 N·cm min
Torque de detención	2.2 N· cm max
Inercia del rotor	54 g·cm^2
Número de cables	4
Peso del motor	280 g

Tabla 5-12: Características 17HS4401

5.6 Controlador motor paso a paso

Como se comentó anteriormente, existen multitud de drivers para motores paso a paso. Los más conocidos en el mundo de la impresión 3D y la robótica están basados en el chip Allegro A4988 y en el Texas Instruments DRV8825. Siendo este último una versión mejorada del A4988 con características ligeramente superiores.

Estos controladores nos permiten manejar las altas tensiones e intensidades que requieren estos motores, limitan la corriente que circula por el motor, y proporcionan las protecciones para evitar que la electrónica pueda resultar dañada.

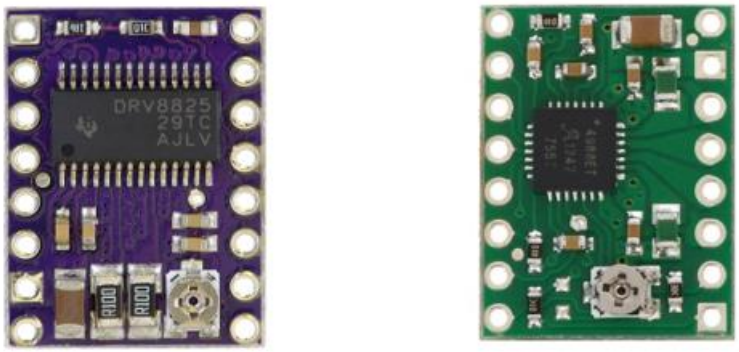


Ilustración 5-16: DRV8825 y A4988

Para su control únicamente requieren dos salidas digitales, una para indicar el sentido de giro (DIR) y otra para comunicar que queremos que el motor avance un paso (STEP). Además, permiten realizar microstepping, una técnica para conseguir precisiones superiores al paso nominal del motor. Esto se consigue dividiendo cada paso del motor en pasos más pequeños, para que la rotación sea más suave sobre todo cuando se trabaja a bajas velocidades. Lo que hace el controlador es manejar la tensión de ancho de pulso (PWM) que envía al motor, controlando así la corriente que atraviesa las bobinas de este. Cuando la corriente aumenta en uno de los devanados, disminuye en el otro. Con esta transferencia gradual de corriente se consigue un movimiento más suave a la vez que se incrementa el par.

Por otro lado, ambos controladores disponen de reguladores de intensidad incorporados. El motivo es que los motores paso a paso de cierto tamaño y potencia,

como por ejemplo los NEMA 23 o en nuestro caso NEMA 17, necesitan tensiones superiores a las que podrían soportar las bobinas por su corriente nominal.

5.6.1 Elección del controlador DRV8825

Para nuestro proyecto utilizaremos el controlador DRV8825 ya que puede trabajar con corrientes superiores al A4988 consiguiendo que el dispositivo no se acerque a los valores máximos que puede manejar mejorando así su vida útil. Además, la diferencia en el precio es mínima. Las características más importantes a la hora de seleccionar un controlador son:

- La tensión de trabajo. El controlador debe ser capaz de trabajar con la tensión requerida por el motor. En nuestro caso, el motor 17HS4401 trabaja a una tensión de 12V por lo que se encuentra dentro del rango de 8.2 – 45 V del controlador.

- La corriente máxima por fase. Para el correcto funcionamiento del controlador, este debe manejar niveles de corriente superiores a los que les va a demandar el motor. La máxima corriente que solicitará nuestro motor será de 1.7 A y como vemos en la tabla 5.14, el DRV8825 puede manejar una corriente de hasta 2.5 A.

- Resolución de microstepping. La configuración del microstepping se consigue mediante los pines M0, M1 y M2. Los tres pines tienen unas resistencias pull-down internas de 100 kΩ por lo que dejando los tres pines desconectados estaríamos en modo Full Step. En nuestro caso utilizaremos una reducción de ¼ para un movimiento más suave. Para ello conectaremos el pin M1 a 5 V dejando los pines M0 y M2 desconectados.

M0	M1	M2	Resolución
Low	**Low**	**Low**	Full Step
High	**Low**	**Low**	1/2 Step
Low	**High**	**Low**	1/4 Step
High	**High**	**Low**	1/8 Step
Low	**Low**	**High**	1/16 Step
High	**Low**	**High**	1/32 Step
Low	**High**	**High**	1/32 Step
High	**High**	**High**	1/32 Step

Tabla 5-13: Tabla de verdad microstepping. Fuente

A4988	Características
Tensión mínima de trabajo	8.2 V
Tensión máxima de trabajo	45 V
Corriente continua por fase	1.5 A
Corriente máxima por fase	2.5 A
Tensión mínima de control	2.5 V
Tensión máxima de control	5.25 V
Resolución microstepping	1/2, 1/4, 1/8, 1/16 y 1/32
Tamaño	0.6″ × 0.8″
Peso	1.3 g

Tabla 5-14: Características DRV8825

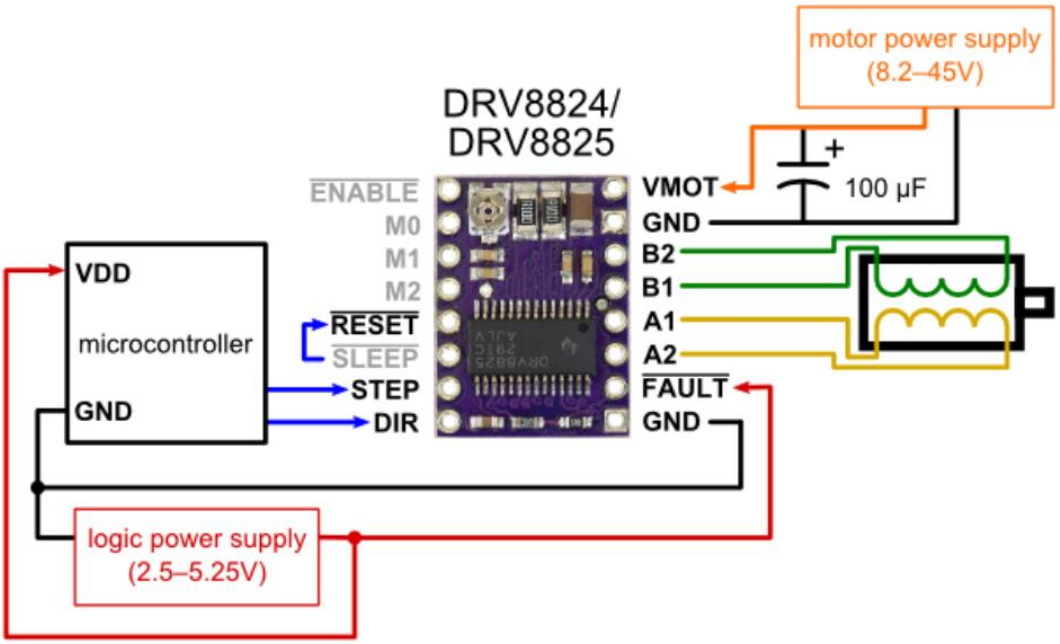


Ilustración 5-17: Pines DRV8825. Fuente

Los pines RESET y SLEEP son los que controlan el funcionamiento del dispositivo, ambos están puestos a nivel bajo mediante resistencias pull-down lo que hace que el DRV8825 desconecte, por defecto, las salidas que van al motor. Si queremos activar el dispositivo, es necesario poner estos pines a nivel alto. Se pueden conectar directamente a un nivel lógico alto entre 2.5 V y 5.25 V o se pueden controlar de forma dinámica conectando estos pines a las salidas digitales de nuestro controlador.

Por otro lado, el estado por defecto del pin ENABLE es bajo por lo que podemos dejarlo desconectado para que el dispositivo permanezca activo.

Hay que tener en cuenta que esta placa utiliza condensadores cerámicos de bajo ESR (resistencia serie equivalente), lo que la hace susceptible a que la placa quede dañada si hay picos de tensión, incluso cuando la tensión de alimentación del motor es tan baja como 12V. Una forma de proteger al conductor de dichos picos es conectar un condensador electrolítico (de al menos 47 µF) entre VMOT y GND en algún lugar cerca de la placa.

A su vez, es importante ajustar la tensión de referencia (VREF) mediante el potenciómetro que llevan incorporadas las placas para que no se dañe el dispositivo y sea capaz de limitar la corriente correctamente. En la hoja de características podemos encontrar la siguiente información:

$$Imax = \frac{Vref}{5\ R_S}$$

(*Fórmula 5.7)*

Donde:

$$Imax = 1.7\ A\ \ (Corriente\ máxima\ motor)$$

$$R_S = 0.1\ \Omega\ (\text{Especificada por el fabricante})$$

Por lo tanto:

$$Vref = 5\ Imax\ R_S = 0.85\ V$$

(*Fórmula 5.8)*

5.7 Control de los motores paso a paso

Este tipo de motores son controlados a través de pulsos enviados mediante el controlador DRV8825. Existen algunas librerías con las que se pueden controlar este tipo de motores como “AccelStepper.h”, pero al no conocer exactamente cómo funcionaban, se decidió crear un código propio para el control de estos. Al principio, se implementó mediante las instrucciones que vienen implementadas en Arduino como “digitalWrite” y “delay” como vemos en el siguiente código.

```
//Activar una direccion y fijar la velocidad con stepDelay
 digitalWrite(dirPin, HIGH);
 stepDelay = 250;
 // Giramos 200 pulsos para hacer una vuelta completa
 for (int x = 0; x < 200; x++) {
    digitalWrite(stepPin, HIGH);
    delayMicroseconds(stepDelay);
    digitalWrite(stepPin, LOW);
    delayMicroseconds(stepDelay);
 }
```

Controlando el número de pulsos (longitud bucle For) y el tiempo entre estos mediante un delay, podíamos controlar la velocidad y el número de pasos que daría el motor. Pero probando este código se llegó a la conclusión de que no era lo suficientemente rápido ni eficiente como para ser controlado a través de un PID. Por este motivo, se decidió controlar los motores a través de interrupciones.

Para ello, se utilizó la interrupción en modo comparación con el Timer 2, de forma que cada un número determinado de microsegundos, se ejecutaría dicha interrupción.

```
/**
   RUTINA QUE CONFIGURA EL TIMER 2 MODO COMPARACIÓN A
*/
void setup_timer2(){
  TCCR2A = 0;                         //Ponemos TCCR2A a cero
  TCCR2B = 0;                         //Ponemos TCCR2B a cero
  TIMSK2 |= (1 << OCIE1A);            //Activamos modo "compare match A interrupt" en
                                      //el registro TIMSK2
  TCCR2B |= (1 << CS21);              //Ponemos el bit CS21 del registro TCCRB a 1
                                      //para prescaler 8
  OCR2A = 39;                         //Para conseguir 20 us, registro de comparación
                                      //A = 39; (20us * (16.000.000MHz / 8)) - 1 = 39
  TCCR2A |= (1 << WGM21);
}
```

Para configurar el Timer 2 se han utilizado los siguientes registros:

Registro	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TCCR2A	COM2A1	COM2A0	COM2B1	COM2B0	-	-	WGM21	WGM20

Tabla 5-15: Registro TCCR2A

Mode	WGM2	WGM1	WGM0	Operación	TOP
0	0	0	0	Normal	0xFF
1	0	0	1	PWM	0xFF
2	0	1	0	CTC	OCRA
3	0	1	1	Fast PWM	0xFF
4	1	0	0	Reservado	-
5	1	0	1	PWM	OCRA
6	1	1	0	Reservado	-
7	1	1	1	Fast PWM	OCRA

Tabla 5-16: Modos timer 2

Con el modo CTC (output compare) conseguimos que cuando el registro del contador TCNT2, alcance un determinado valor, es decir, cuando TCNT2 tome el valor de OCR2A, se resetee el contador y vuelva a comenzar el contaje. Se ha optado por la utilización del Timer 2 ya que tiene una resolución de 8 bits pudiendo contar de 0 hasta 255, valor suficiente para nuestro proyecto. Además, este contador, a diferencia del Timer 0 que tiene la misma resolución, no tiene asociadas funciones de tiempo como *delay(), micros(), millis()…* por lo que evitamos posibles interferencias.

Registro	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TCCR2B	FOC2A1	FOC2A0	-	-	WGM22	CS22	CS21	CS20

Tabla 5-17: Registro TCCR2B

CS22	CS21	CS20	Descripción
0	0	0	Stop
0	0	1	Sin prescaler
0	1	0	Prescaler 8
0	1	1	Prescaler 32
1	0	0	Prescaler 64
1	0	1	Prescaler 128
1	1	0	Prescaler 256
1	1	1	Prescaler 1024

Tabla 5-18: Prescaler

El contador TCNT2, se incrementa en cada pulso de la señal de reloj, es decir, a una frecuencia de 16 MHz. Sin embargo, existe lo que se denomina un prescaler. Estableciendo un prescaler determinado, conseguimos que el contador TCNT2, se incremente cada un cierto número de pulsos determinado por el usuario.

Para que se mande un pulso al motor, necesitamos que pasen como mínimo 20 µs. Para conseguir esto, se ha optado por utilizar un prescaler igual a 8, consiguiendo que el contador del timer se incremente a un ritmo de 2 MHz (16 MHz/ 8). Si queremos que entre a la interrupción ORC2A cada 20 µs el contador tiene que incrementarse 40 veces, de 0 a 39. Este valor es el que se introduce en el registro OCR2A.

$$\frac{20 \cdot 10^{-6}\ s \cdot 2 \cdot 10^{6}\ Hz}{1\ s} = 40\ Hz$$

(*Fórmula 4.9)*

Registro	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TIMSK2	-	-	-	-	WGM22	OCIE2B	OCIE2A	TOIE2

Tabla 5-19: Registro TIMSK2

Con el registro TIMSK2, podemos configurar el tipo de interrupción asociada al Timer 2.

Registro	Interrupción asociada	Se produce cuando
OCIE2A	ISR(TIMER2_COMPA_vect)	TCNT2 = OCR2A
OCIE2B	ISR(TIMER2_COMPB_vect)	TCNT2 = OCR2B
TOIE2	ISR(TIMER2_OVF_vect)	TCNT2 OverFlow

Tabla 5-20: Interrupciones asociadas al timer

Una vez tenemos configurado el Timer 2 y activada la interrupción asociada a dicho timer que se ejecutará cada 20 µs. Procedemos a definir qué tiene que hacer dicha interrupción.

```
/**
   RUTINA DE INTERRUPCIÓN ASOCIADA AL TIMER 2. SE EJECUTA CADA 20 us
*/
ISR(TIMER2_COMPA_vect) {

  //Motor derecho
  //Cada vez que entra se incrementa el contador (equivale a un paso)
  cont_step_der ++;
  if (cont_step_der > max_cont_step_der) {  //No se da un paso hasta que el
                                            //contador no supera el tiempo
                                            //(20us*max_cont_der) que queremos
                                            //entre pasos
    cont_step_der = 0;                      //Se resetea el contador
    max_cont_step_der = pasos_der;      //Se establece un nuevo tiempo entre pasos
    if (max_cont_step_der < 0) {        //Si max_cont_step_der es negativa
      PORTD &= (~(1 << dir_der));   //Ponemos dir_der a low para invertir dirección
      max_cont_step_der *= -1;      //Complementamos max_cont_step_der
    }
    else PORTD |= (1 << dir_der);
  }
  else if (cont_step_der == 1)PORTD |= (1 << step_der);//Se manda un pulso al motor
  else if (cont_step_der == 2)PORTD &= (~(1 << step_der));


  //Motor izquierdo
 //Cada vez que entra se incrementa el contador (equivale a un paso)
  cont_step_izq ++;
  if (cont_step_izq > max_cont_step_izq) { //No se da un paso hasta que el contador
                                           //no supera el tiempo(20us*max_cont_der)
                                           //que queremos entre pasos
    cont_step_izq = 0;                     //Se resetea el contador
    max_cont_step_izq = pasos_izq;         //Se establece un nuevo número de pasos
    if (max_cont_step_izq < 0) {           //Si max_cont_step_der es negativa
      PORTD &= (~(1 << dir_izq));   //Ponemos dir_der a low para invertir dirección
      max_cont_step_izq *= -1;      //Complementamos max_cont_step_izq
    }
    else PORTD |= (1 << dir_izq);
  }
  else if (cont_step_izq == 1)PORTD |= (1 << step_izq);//Se manda un pulso al motor
  else if (cont_step_izq == 2)PORTD &= (~(1 << step_izq));

     }
```

En esta rutina, calculamos cada cuánto tiempo se envía un pulso al DRV8825, que equivale a un paso del motor. Así, controlando la frecuencia entre pasos, controlamos la velocidad de giro del motor. La variable *pasos_der*, almacena el número de veces que hace falta entrar a la interrupción para que finalmente se dé un paso. Si esta variable es negativa, quiere decir que el motor ha de cambiar de dirección. Como vemos, cada vez que se entra a la interrupción, se incrementa un contador. Cuando este alcanza el valor de la variable *pasos_der*, se envía un pulso.

Para enviar estos pulsos, en vez de utilizar funciones como *digitalWrite(),* se han utilizado máscaras con el puerto PORTD del controlador Atmega328, consiguiendo una mayor velocidad de respuesta y eficiencia en el código.

5.8 Control bluetooth del vehículo mediante smartphone

Como se expuso anteriormente, en el vehículo se ha implementado un receptor bluetooth para poder ser controlado desde un smartphone. Para esto, se ha utilizado una aplicación descargada de Play Store llamada BlueTooth Serial Controller.

Ilustración 5-18: Logotipo aplicación Bluetooth

Se ha escogido esta aplicación por ser de código libre y muy intuitiva de configurar. Para crear la interfaz de usuario, simplemente hay que seleccionar el número de botones que queremos que aparezcan, así como su disposición. Seguidamente, se le asigna a cada botón un código específico que será el que reciba nuestro receptor cuando se pulse el botón. Nuestra interfaz, contará con un total de cuatro botones como se puede observar en la siguiente figura:

Ilustración 5-19: Interfaz Bluetooth

En este caso, a cada botón se le ha asociado un número:

- Si se pulsa el botón ADELANTE, nuestro móvil estará enviando un 1.
- Si se pulsa el botón ATRÁS, nuestro móvil estará enviando un 2.
- Si se pulsa el botón IZQUIERDA, nuestro móvil estará enviando un 3.
- Si se pulsa el botón DERECHA, nuestro móvil estará enviando un 4.
- Si no se pulsa ningún botón, se enviará un 0.

```
//Control Bluetooth
  if (HC05.available()){
    dato = HC05.read();
  }
  if (dato == '0'){                                  //STOP
       referencia = 0;
  }
  if (dato == '1'){                                  //Adelante
       referencia -= 0.1;
       if (referencia < -3) referencia = -3;
  }
  if (dato == '2'){                                  //Atrás
       referencia += 0.1;
       if (referencia > 3) referencia = 3;
  }
  if (dato == '3'){                                  //Izquierda
       salida_PID_izq -= 25;
       salida_PID_der += 25;
  }
  if (dato == '4'){                                  //Derecha
       salida_PID_izq += 25;
       salida_PID_der -= 25;
  }
```

Como podemos ver en el código, para conseguir que el robot vaya hacia adelante, se cambia la referencia poco a poco hasta que llegue a los tres grados negativos respecto a la vertical. El mismo procedimiento se emplea si queremos que vaya hacia atrás, pero en este caso la referencia se corresponde con tres grados positivos. Cuando no se pulse ningún botón, la referencia vuelve a su valor predeterminado de cero grados.

Por otro lado, si se desea girar a la derecha, se aumenta la velocidad del motor izquierdo y se disminuye la del derecho. Siguiendo el mismo proceso, pero invertido el vehículo es capaz de girar a la izquierda.

5.9 Control PID

Con las siguientes líneas de código, se consigue que Arduino actúe a modo de controlador PID.

```
//PID
 error = angulo_nuevo - referencia_automatica - referencia;            //Error

 //Para evitar que el error se descontrole
 if(salida_PID > 10 || salida_PID < -10)error += salida_PID * 0.02 ;
 integral += ki*error;                                          //Componente integral
 if (integral > 400) integral = 400;                            //Anti Wind-up
 else if (integral < -400) integral = -400;

 salida_PID = kp*error + integral + kd*(error - error_anterior);  //Salida del PID
if (salida_PID > 400) salida_PID = 400;                           //Anti Wind-up
 else if (salida_PID < -400) salida_PID = -400;

 error_anterior = error;                                    //Actualización del error

//Con lo siguiente, conseguimos un comportamiento más suave y que no esté
//constantemente vibrando
 if(salida_PID < 15 && salida_PID > -15)salida_PID = 0;

if(angulo > 37 || angulo < -37){            //Si se cae, que se paren los motores
   salida_PID = 0;
   integral = 0;
 }

 if(referencia == 0){
   //Se incrementa la referencia automática si el robot va hacia delante
   if(salida_PID < 0)referencia_automatica += 0.02;
   //Se decrementa la referencia automática si el robot va hacia atrás
   if(salida_PID > 0)referencia_automatica -= 0.02;
}


while(loop_timer > micros());         //Timer 2ms
 loop_timer += 2000;
```

Como se puede observar, el error es la diferencia entre la referencia y el ángulo que mide el MPU6050 (*ángulo_nuevo*). Además, se ha añadido un término denominado *referencia_automática* el cual hace que el error aumente o disminuya dependiendo de si el vehículo se está moviendo hacia adelante, o hacia atrás consiguiendo que encuentre su punto de equilibrio más rápidamente.

Para implementar la componente integral, se van sumando los errores pasados con el error actual multiplicado por la constante de integración. Para la componente derivativa, se hace la diferencia entre el error actual y el error anterior, multiplicada por su respectiva ganancia derivativa.

Por otro lado, se han dedicado unas líneas para eliminar el fenómeno denominado Wind – Up. Este fenómeno consiste en que la componente integral puede dispararse si no se consigue eliminar el error por lo que, si tenemos un controlador limitado que no puede seguir los valores de esta componente, se producirá un retraso en el sistema en el momento en que cambie el error. Es decir, el PID "cree" que está enviando una determinada acción de control, pero debida a la limitación física del controlador, se estará enviando una acción de control inferior confundiendo al sistema.

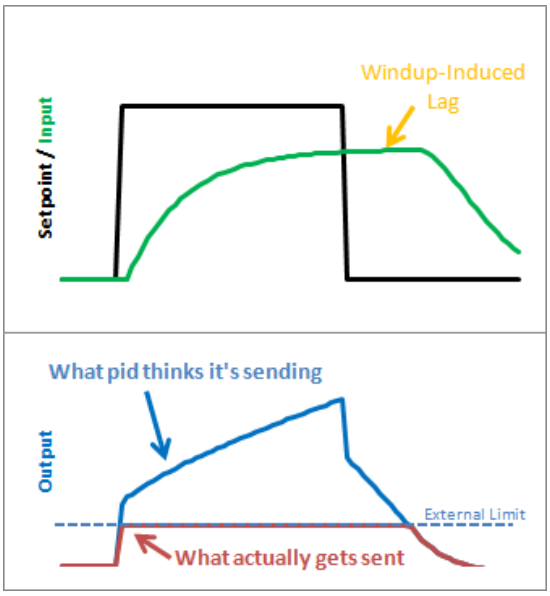


Ilustración 5-20: Efecto Wind-Up

Para evitar este efecto, se limita tanto el valor máximo que puede tomar la acción integral como la acción de control, es decir, la salida del PID.

Finalmente, para que el controlador fuera más estable y las operaciones matemáticas correspondientes a las acciones derivativa e integral no dependieran del tiempo, se decidió implementar un pequeño bucle de espera de 2 ms para que la acción de control siempre se ejecute a periodos regulares de tiempo.

Una vez tenemos el algoritmo de control, el último paso consiste en sintonizar el PID, es decir, dar valores a las distintas ganancias para conseguir que el vehículo se balancee de la manera más precisa, rápida y eficiente posible. Existen muchos métodos como el de Ziegler – Nichols, el método de la Curva de Reacción o el método de Tyreus y Luyben. En nuestro caso, se ha utilizado un método manual,

probando distintos valores de ganancias hasta que se obtuvo el comportamiento deseado. Para esto, se siguieron una serie de pasos:

- Primero, la ganancia integral y derivativa se establecen a cero.
- A continuación, se aumenta la ganancia proporcional hasta que el sistema oscila de forma controlada respecto al punto de equilibrio.
- Una vez tenemos este valor de ganancia proporcional, se disminuye a la mitad y se va incrementando la ganancia integral hasta que el proceso se ajuste en el tiempo requerido sin llegar a ser inestable.
- Finalmente, se incrementa el valor de la ganancia derivativa hasta que el lazo sea lo suficientemente rápido para alcanzar su referencia tras una perturbación brusca.

Este puede ser un proceso muy lento y costoso por lo que se decidió ir enviando los distintos valores de las ganancias por bluetooth. Los valores con los que mejor responde el sistema son los siguientes:

```
//Ganancias PID
float kp = 4.8;        //Ganancia proporcional
float ki = 3.5;        //Ganancia integral
float kd = 6.4;        //Ganancia derivativa
```

5.10 Alimentación

Para que nuestro vehículo sea autónomo y no precise estar conectado a una fuente de alimentación es necesario que éste sea alimentado mediante una batería. Existen multitud de alternativas, siendo las más utilizadas las baterías de Níquel-Metalhidruro (NiMh), las pilas recargables o las baterías de Polímero de Litio (LiPo).

Para nuestro proyecto utilizaremos una batería LiPo ya que suponen la opción más avanzada. Estas presentan una serie de ventajas, siendo las principales:

- Su alta densidad de energía que prácticamente dobla a las de NiMh.
- Presentan un volumen reducido y ofrecen un formato más práctico, lo que las hace más manejables.

- Alto nivel de descarga.
- Alto nivel de tensión por celda.
- Resistencia interna pequeña, lo que hace que se pueda aprovechar casi el 100% de la energía disponible.

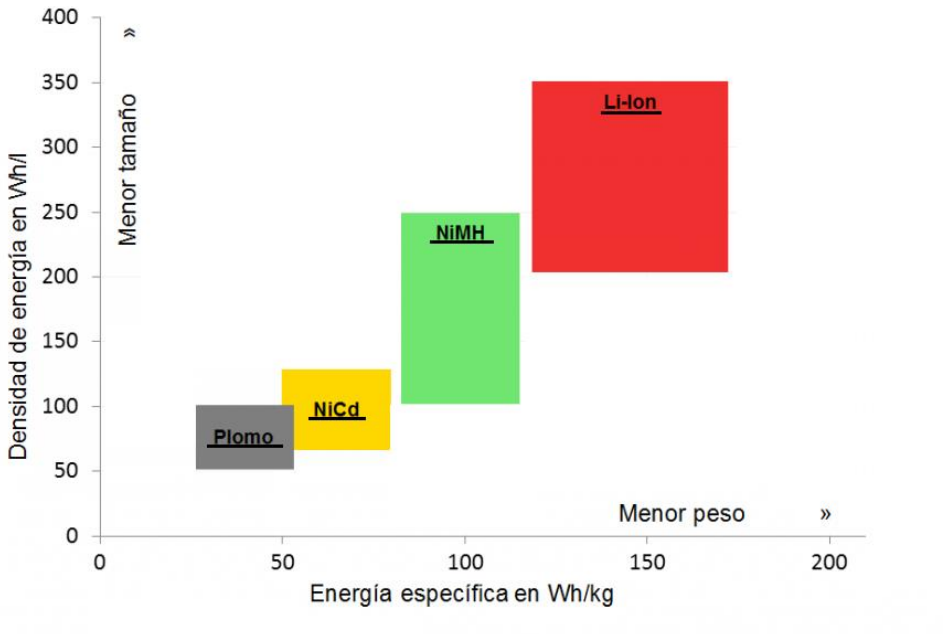


Ilustración 5-21: Comparativa entre materiales para baterías. Fuente

En cuanto a criterios de selección hay que tener en cuenta las siguientes características:

- Número de celdas. Siendo este valor el que nos indica la tensión de la batería ya que cada celda proporciona 3.7 V. Así las LiPo de 2 celdas (2S) proporcionan 7.4V, las de 3 celdas (3S) proporcionan 11.1 V y así sucesivamente.
- Capacidad. Esta se mide en mAh y es una indicación de cuánta corriente se puede extraer de la batería durante una hora. Por ejemplo, una batería de 1300 mAh podría suministrar una corriente de 1.3 A durante 1 hora. Si se consumieran 2.6 A, tardaría en descargarse media hora.

- C Rating o tasa de descarga. Esta es la velocidad con la que se puede descargar la batería, es decir, la intensidad máxima que puede dar la batería de forma segura. La elección de las C de descarga debe hacerse en función del consumo máximo previsto y preferentemente agregando un margen del 50% o 100% adicional para una mayor longevidad de la batería. Por ejemplo, una batería de 2200 mAh y 25C sería capaz de proporcionar como máximo 55 A.

mAh	10C (A)	15C (A)	20C (A)	25C (A)	30C (A)	40C (A)	50C (A)	60C (A)
5000	50	75	100	125	150	200	250	300
2800	28	42	56	70	84	112	140	168
2500	25	37,5	50	62,5	75	100	125	150
2200	22	33	44	55	66	88	110	132
2000	20	30	40	50	60	80	100	120
1500	15	22,5	30	37,5	45	60	75	90
1000	10	15	20	25	30	40	50	60
500	5	7,5	10	12,5	15	20	25	30
250	2,5	3,75	5,0	6,25	7,5	10	12,5	15
200	2,0	3,00	4,0	5,00	6,0	8,0	10	12
150	1,5	2,25	3,0	3,75	4,5	6,0	7,5	9,0
100	1,0	1,50	2,0	2,50	3,0	4,0	5,0	6,0

Tabla 5-21: Tasa de descarga. Fuente

5.10.1 Elección batería

Teniendo en cuenta los criterios de selección, primero debemos seleccionar la tensión de la batería. Tanto Arduino como el resto de dispositivos electrónicos funcionan con una tensión de 5 V, pero los motores paso a paso requieren una tensión de 12 V por lo que necesitamos una batería capaz de suministrar ese nivel de tensión. La que más se acerca con una tensión de 11.1 V es una batería 3S, es decir, de tres celdas.

Ilustración 5-22: Batería utilizada

La gran desventaja de estas baterías es que son muy sensibles a sobrecargas pudiendo llegar a explotar si no se cargan correctamente. También, si su tensión disminuye a niveles inferiores de 9 V, las celdas pueden dañarse de forma permanente quedando la batería totalmente inutilizable. Por este motivo, se han añadido unas líneas al código de Arduino de forma que si la tensión de la batería baja de 9.5 V, nos avise mediante el encendido de un LED y se paren los motores.

Para esto, se ha conectado la batería a la entrada analógica A0 de Arduino, pero hay que tener en cuenta que la tensión máxima que puede medir este pin es de 5 V por lo que se ha diseñado un pequeño divisor de tensión, consiguiendo disminuir la tensión de la batería de 12 V a 5 V. Tanto las ecuaciones empleadas como los valores de las resistencias se pueden ver a continuación:

$$V_o = V_i \frac{R2}{R1 + R2} = 12.5\,V \cdot \frac{2k2\Omega}{2k2\Omega + 3k3\Omega} = 5V \rightarrow\ A0 = 1023$$

(*Fórmula 5.10)*

$$9.5\,V \cdot \frac{2k2\Omega}{2k2\Omega + 3k3\Omega} = 3.8\,V \rightarrow A0 = 810$$

(*Fórmula 5.11)*

Una vez hemos conseguido disminuir la tensión, sabemos que el pin analógico nos va a devolver un valor de 1023 cuando la batería esté totalmente cargada, es decir, cuando en el pin haya conectada una tensión de 5 V (12 V en la batería). Cuando esta disminuya a 3.8 V (9.5 V en la batería), estaremos leyendo en el puerto el valor 810.

```
bateria = analogRead(0);                  //Batería conectada al puerto A0

 if(bateria < 800 && bateria> 700){       //Si la tensión de la batería baja de 9.5 V
   digitalWrite(13, HIGH);                //Se enciende el LED
   motor_der = 0;                         //Se paran ambos motores
   motor_izq = 0;
 }
```

5.11 Diseño CAD

Para diseñar la estructura del vehículo se ha optado por realizar un diseño CAD mediante el software Autodesk Inventor Professional para posteriormente imprimir las distintas piezas con una impresora 3D.

Autodesk Inventor es un paquete de modelado paramétrico de sólidos en 3D producido por la empresa de software Autodesk. Compite con otros programas de diseño asistido por computadora como SolidWorks, Pro/ENGINEER, CATIA y Solid Edge. Se ha optado por este software debido a que es el único que ofrece una licencia gratuita a estudiantes durante tres años.

Ilustración 5-23: Autodesk Inventor Professional

Además, el propio software incorpora una herramienta para guardar las piezas diseñadas en formato “.STL” (Standard Triangle Language). Este es un formato de archivo informático de diseño asistido por computadora (CAD) que define la geometría de objetos 3D, excluyendo información como color, texturas o propiedades físicas que sí incluyen otros formatos CAD. Este archivo será el que posteriormente enviaremos al software de segmentación para poder imprimir en 3D.

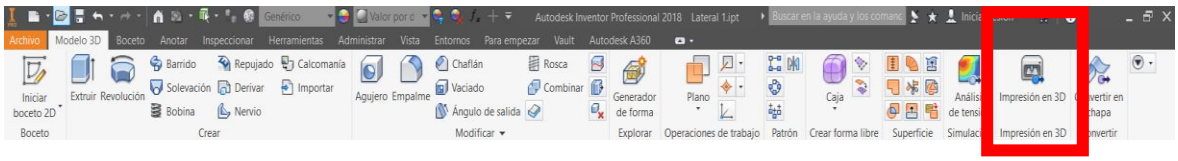


Ilustración 5-24: Herramienta impresión 3D

Todo el conjunto consta de un total de 9 piezas, quedando el ensamblado como se muestra en la figura 5.25.

Ilustración 5-25: Resultado modelado 3D

5.12 Impresión de piezas en 3D

La impresión 3D es un grupo de tecnologías de fabricación por adición donde un objeto tridimensional es creado mediante la superposición de capas sucesivas de material. Se ha optado por esta técnica ya que las impresoras 3D son por lo general más rápidas, más baratas y más fáciles de usar que otras tecnologías de fabricación por adición.

Gracias a esta técnica se han obtenido las distintas piezas que componen el vehículo. Una vez tenemos las distintas piezas en formato .STL, se necesita otro software para segmentar dichas piezas obteniendo así un G–Code ("Código G"). Este lenguaje es el más utilizado en control numérico, es decir, gracias a este lenguaje le podemos decir a la impresora 3D cómo debe moverse, a qué velocidad, qué trayectoria seguir, etc.

En nuestro caso, se ha optado por el software Ultimaker Cura por su sencillez a la hora de manejar el programa y por ser de código abierto. Para la impresión de las distintas piezas se han utilizado los siguientes parámetros básicos de impresión:

Herramientas	Valores
Material	PLA 1.75 mm
Altura de capa	0.3 mm
Grosor de pared	1.2 mm
Relleno	25% Rejilla
Temperatura de impresión	210 ºC
Tipo de adherencia	Falda 3 mm

Tabla 5-22: Características impresión 3D

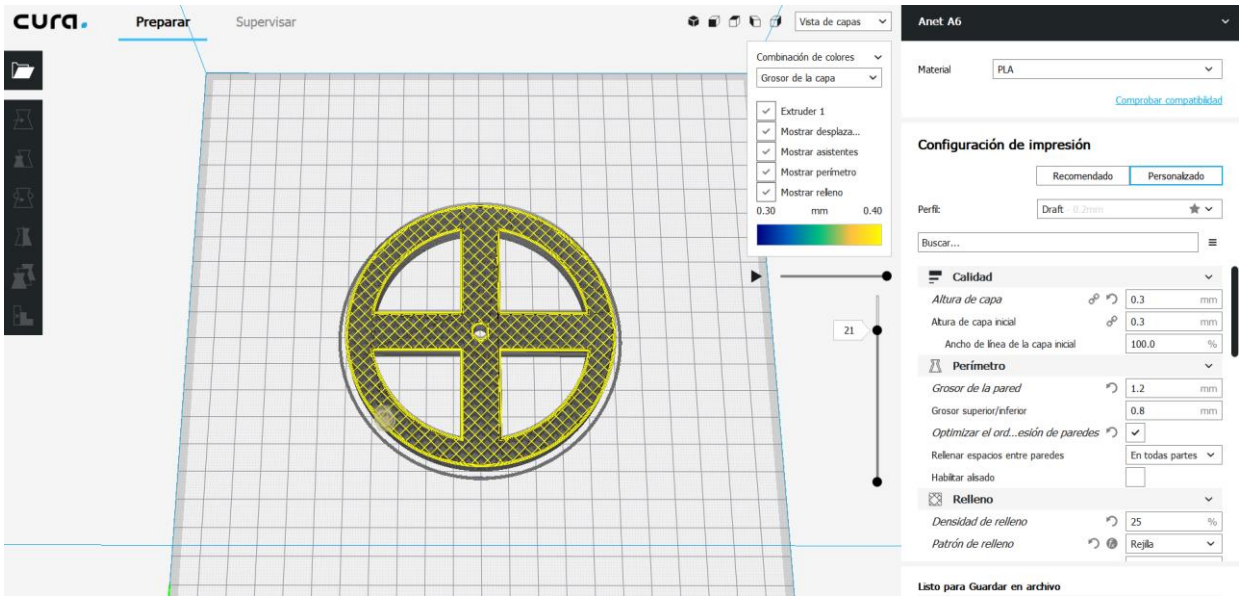


Ilustración 5-26: Software Ultimaker Cura

5.13 Esquemas de conexión

Para diseñar el circuito eléctrico que llevará implementado el vehículo, se ha utilizado el software gratuito Fritzing. Este software presenta la ventaja de que, aparte de ser open source, es muy intuitivo a la hora de su utilización teniendo un gran número de librerías y componentes disponibles. Como podemos ver en las imágenes siguientes, con este programa podemos obtener tres tipos de esquemas de conexión:

Ilustración 5-27: Logotipo Fritzing

- Un esquema más visual, en el que se puede apreciar la apariencia física de los componentes haciéndonos una idea muy aproximada de cómo quedarán conectados cuando lo implementemos físicamente.

- Un esquema más simple en el que aparece una representación de los componentes de acuerdo a su tamaño y número de pines. No es tan visual como el primero, pero puede llegar a ser útil si no sabemos dónde va conectado un determinado componente ya que es un plano más preciso que el anterior.

- Un esquema en el que se representa de manera aproximada cómo quedarían las conexiones si estas las implementamos en una placa de prototipos o PCB. Se pueden mostrar ambas vistas, superior e inferior.

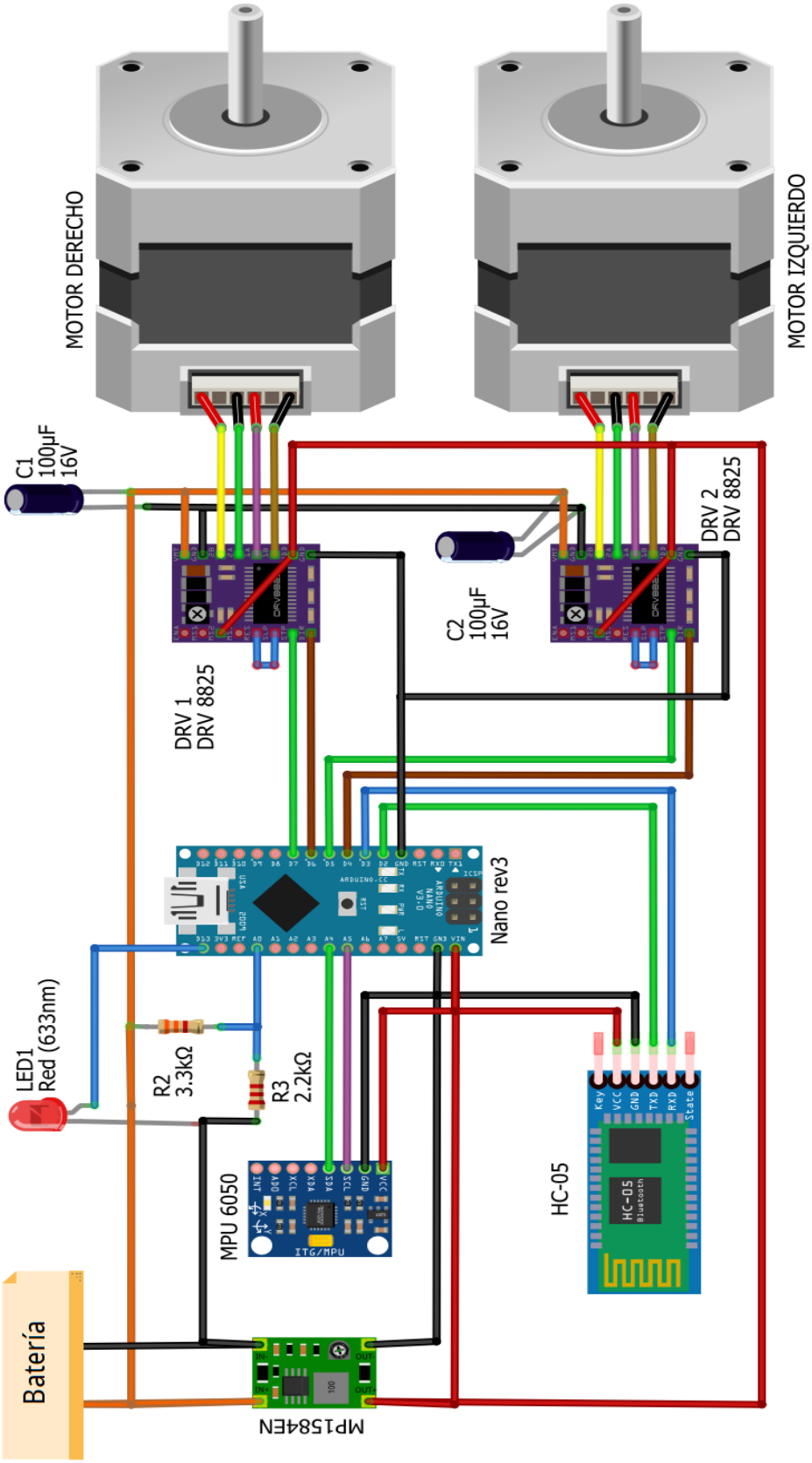


Ilustración 5-28: Esquema conexión 1

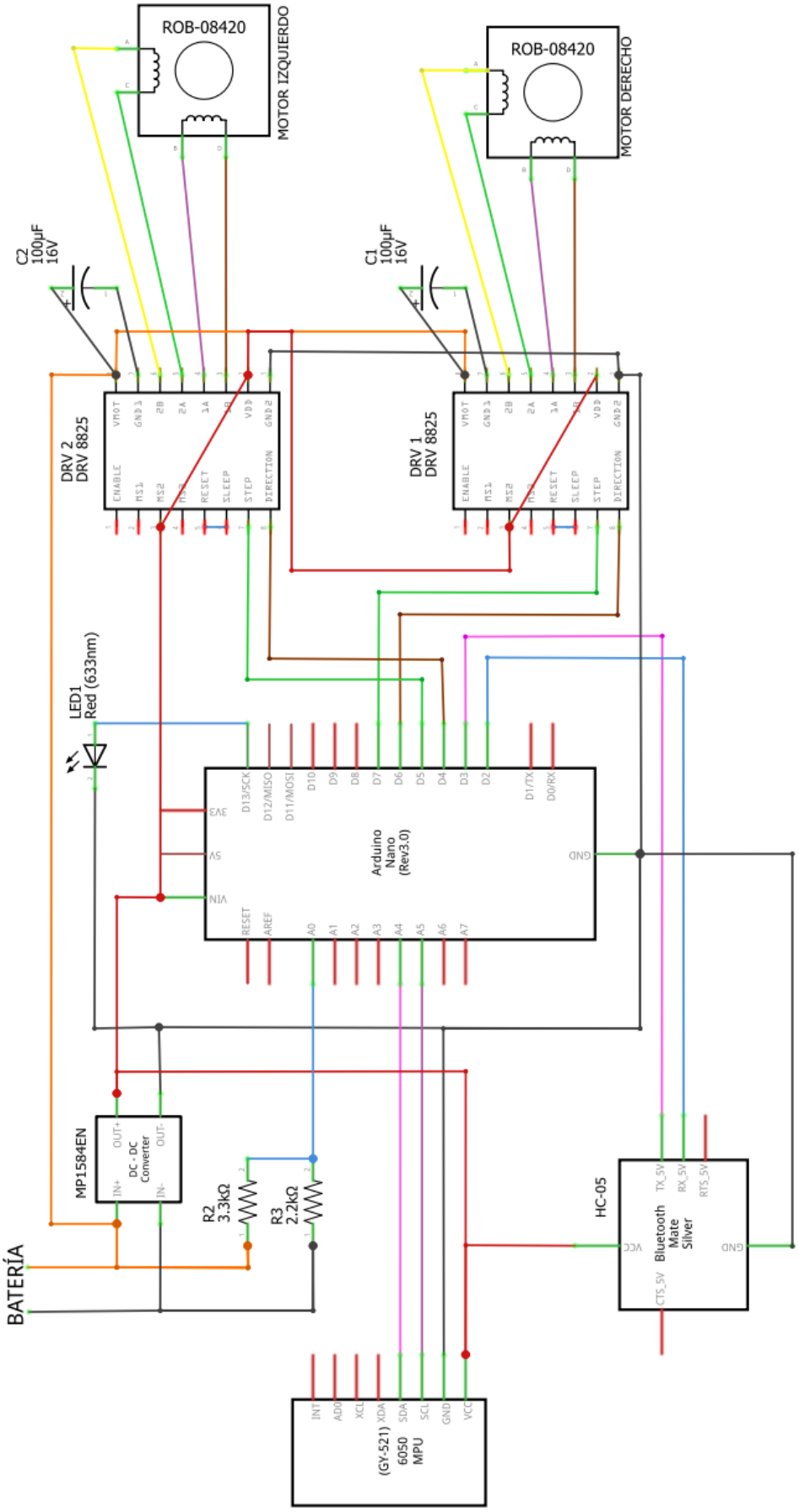


Ilustración 5-29: Esquema conexión 2

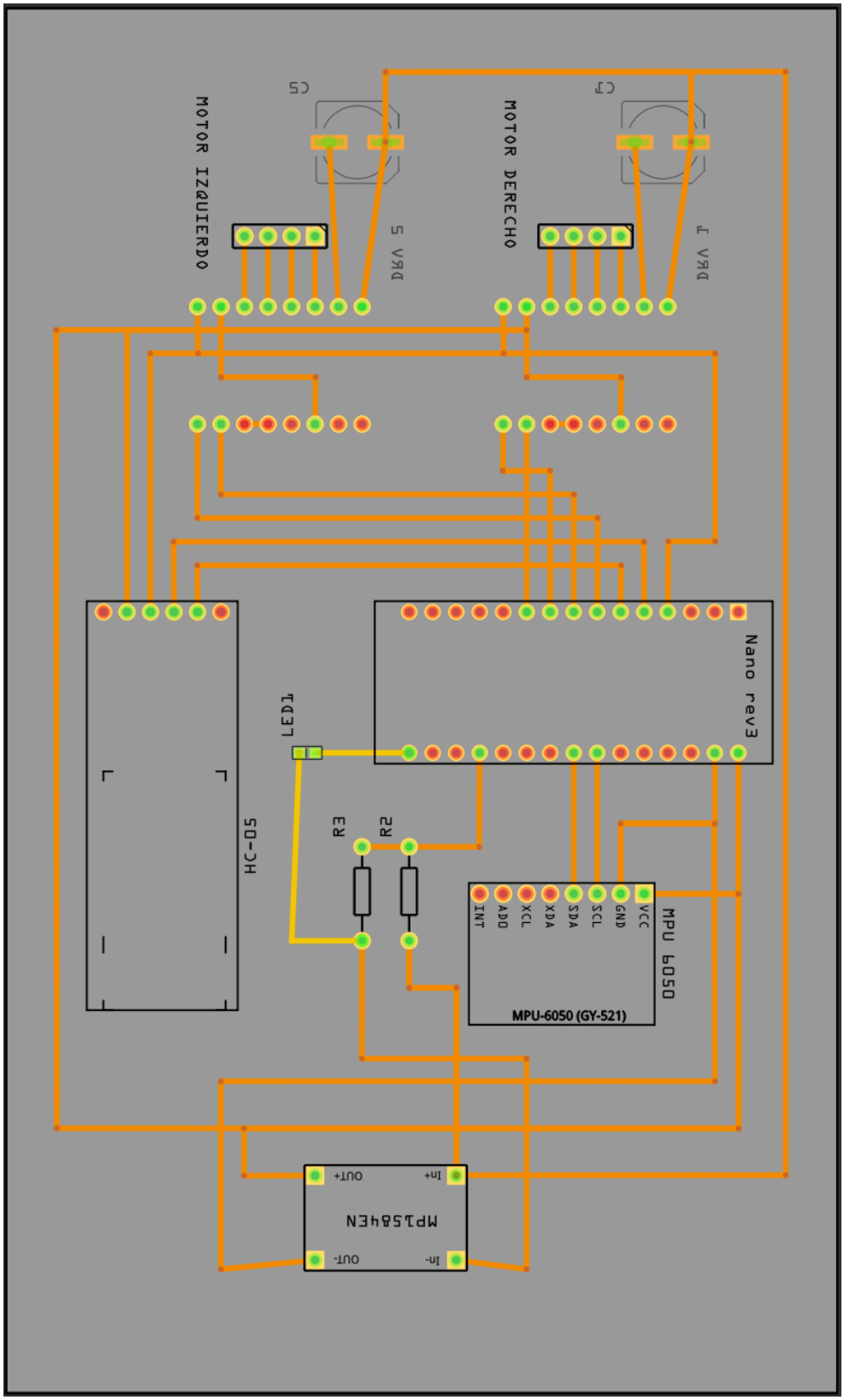


Ilustración 5-30: Esquema conexión 3

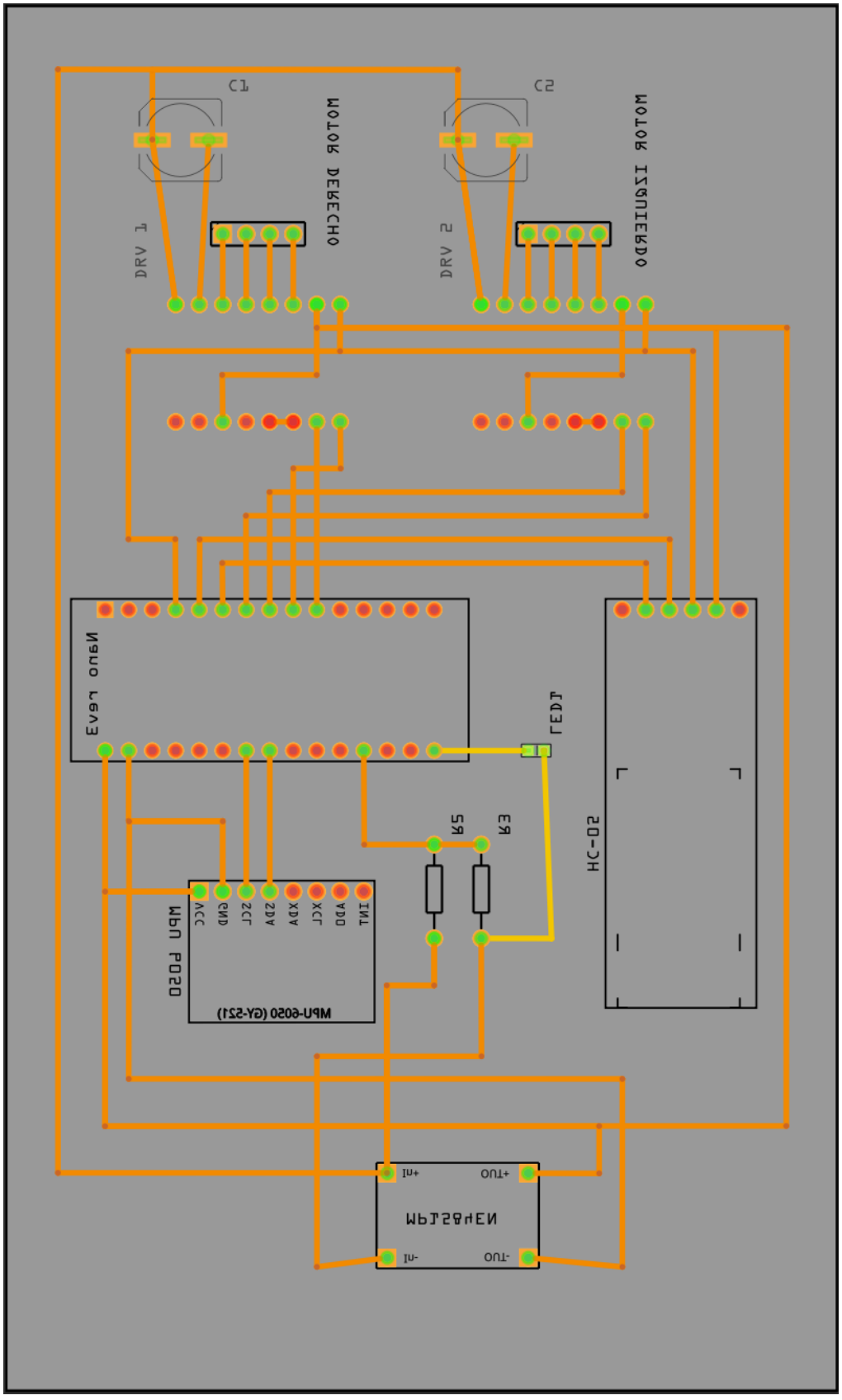


Ilustración 5-31: Esquema conexión 4

5.14 Resultado final

Ilustración 5-32: Perspectiva vehículo

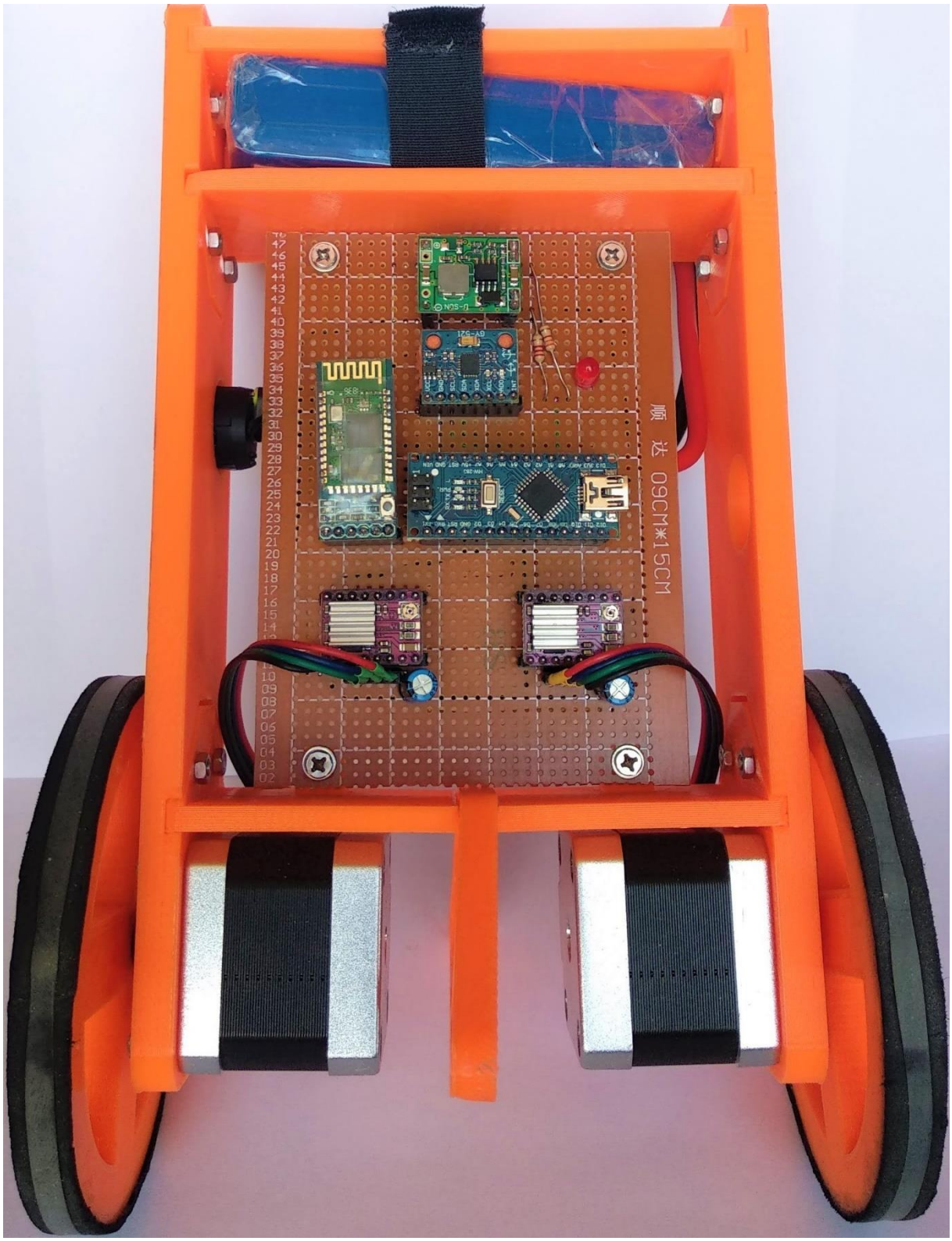


Ilustración 5-33: Frontal vehículo

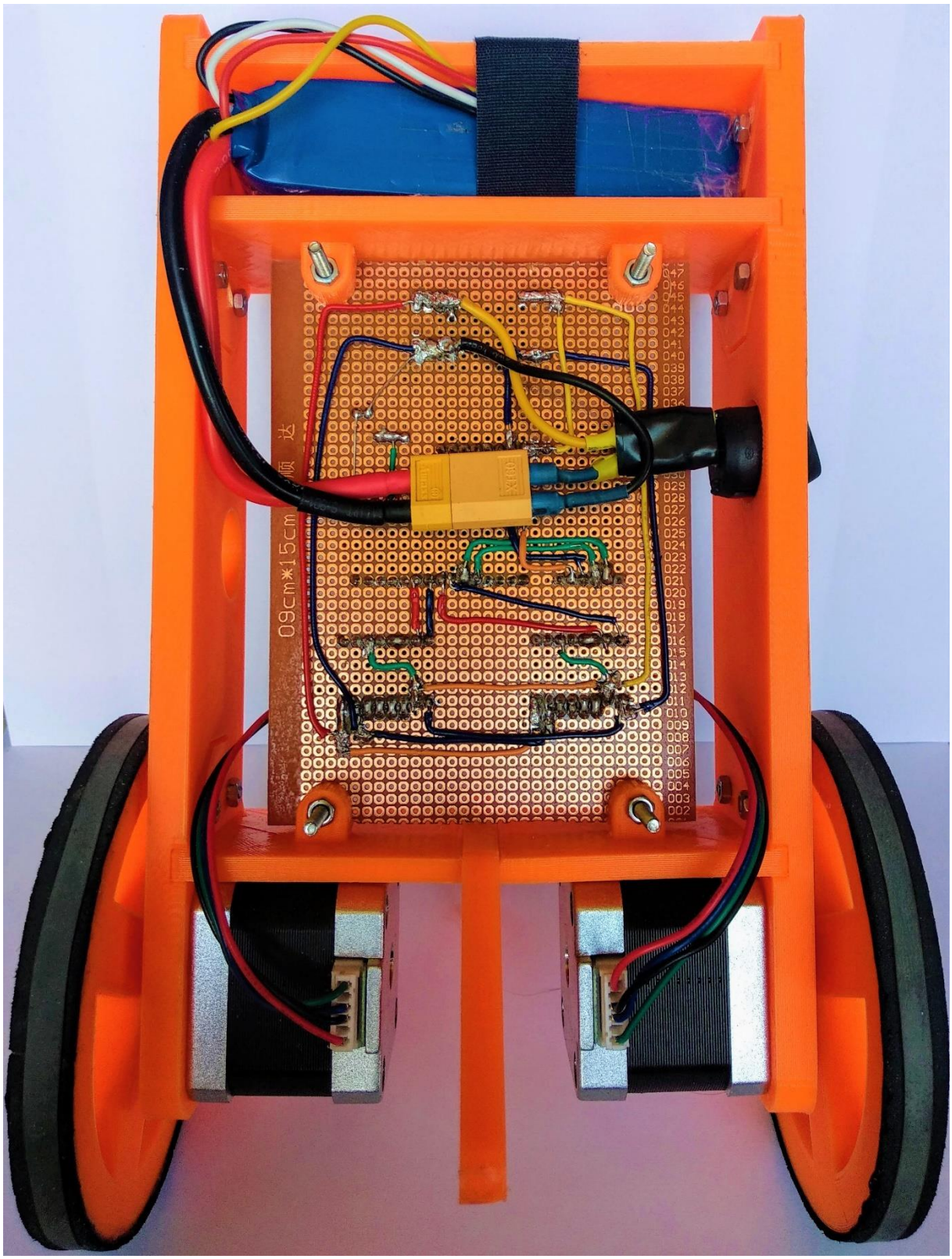


Ilustración 5-34: Posterior vehículo

Ilustración 5-35: Lateral vehículo

6 Ensayos

Una vez tuvimos el vehículo construido nos dimos cuenta de que presentaba algunos fallos los cuales fueron solventados mediante la realización de distintos ensayos.

6.1 Corrección del comportamiento no lineal de los motores

Probando el vehículo, nos dimos cuenta de que los motores no respondían como esperábamos ya que la curva de aceleración de los motores paso a paso no presentaba un comportamiento lineal. Esto supone un gran problema y es que no tiene sentido implementar un controlador PID con un sistema no lineal. Si representamos la frecuencia entre pulsos de los motores obtenemos la siguiente gráfica:

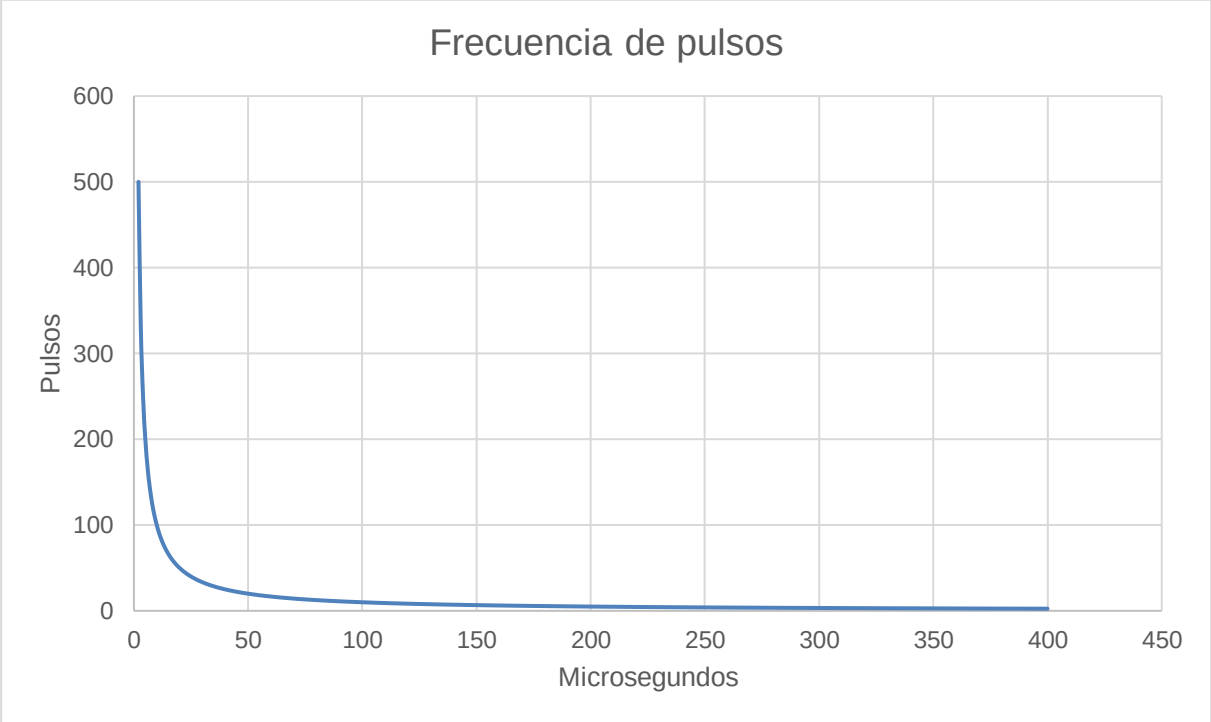


Ilustración 6-1: Gráfica frecuencia entre pulsos

Observando esta gráfica, podemos comprobar que se asemeja bastante a una función logarítmica por lo que aplicando una serie de operaciones matemáticas

podemos linealizar dicha función. Probando distintos valores con la ayuda de una hoja de datos, conseguimos invertir la función como podemos ver en la siguiente gráfica:

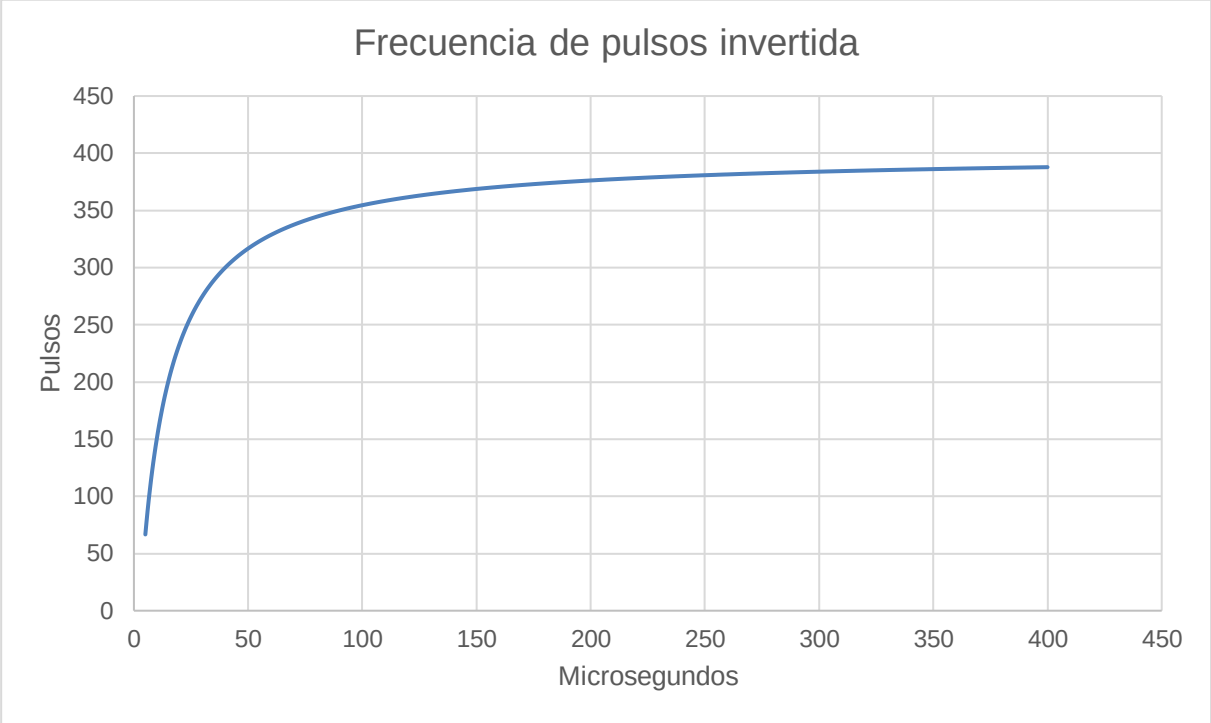


Ilustración 6-2: Gráfica frecuencia entre pulsos invertida

```
//Para compensar el comportamiento no lineal de los motores
  if(salida_PID_izq > 0)salida_PID_izq = 395 - (1/(salida_PID_izq +
10))*5000;
  else if(salida_PID_izq < 0)salida_PID_izq = - 395 -
(1/(salida_PID_izq -   10))*5000;

  if(salida_PID_der > 0)salida_PID_der = 395 - (1/(salida_PID_der +
10))*5000;
  else if(salida_PID_der < 0)salida_PID_der = - 395 -
(1/(salida_PID_der -   10))*5000;
```

Una vez tenemos la función invertida, para linealizarla hay que aplicar:

```
 //Calculamos el tiempo entre pulsos de los motores
  if(salida_PID_izq > 0)motor_izq = 400 - salida_PID_izq;
  else if(salida_PID_izq < 0)motor_izq = - 400 - salida_PID_izq;
  else motor_izq = 0;

  if(salida_PID_der > 0)motor_der =  400 - salida_PID_der;
  else if(salida_PID_der < 0)motor_der =  - 400 - salida_PID_der;
  else motor_der = 0;
```

Obteniendo así una salida (motor_der y motor_izq) prácticamente lineal.

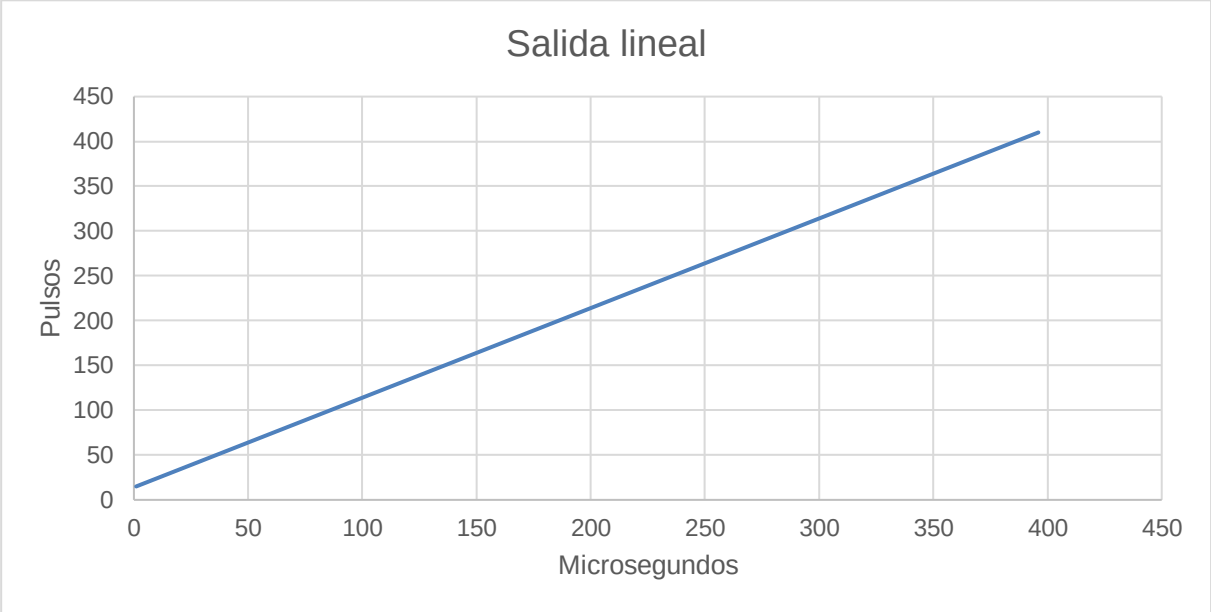


Ilustración 6-3: Salida lineal

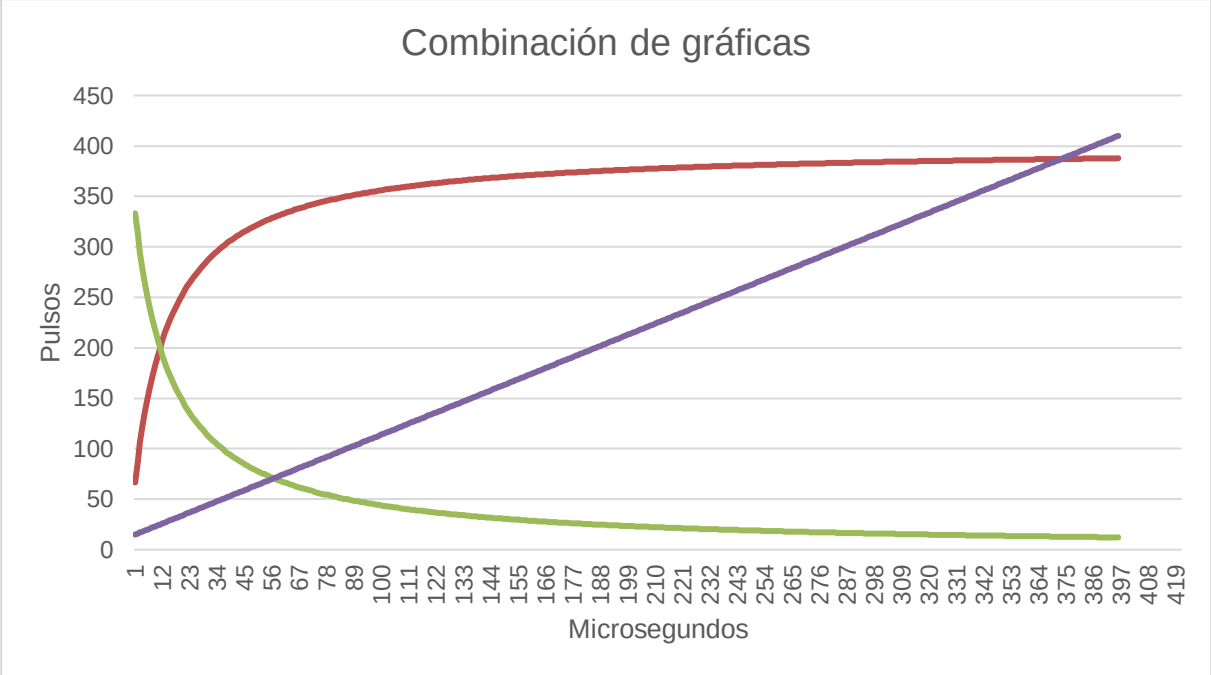


Ilustración 6-4: Combinación de gráficas

6.2 Corrección ante vibraciones

Una vez obtuvimos una respuesta más o menos lineal de los motores, conseguimos que el robot empezara a balancearse ajustando las distintas ganancias del controlador PID. Sin embargo, por más que probamos distintos valores para estas ganancias y aunque el robot conseguía mantenerse cerca de la vertical, estaba constantemente vibrando. Se llegó a la conclusión de que no era un problema del controlador PID, sino del sensor MPU6050, que era demasiado sensible al ruido y había muchos picos en sus medidas provocando que el vehículo estuviera continuamente oscilando. Para solucionar esto, se añadieron las dos líneas de código que se ven a continuación dentro de la función "datos_mpu6050()" para que el valor del ángulo que se estaba enviando al control PID, no fuera sólo el valor actual, sino que se tuvieran en cuenta las medidas pasadas mediante un parámetro de ponderación. Con esto se consiguió suavizar la respuesta del sensor y amortiguar las vibraciones en el vehículo.

```
 //Filtro complementario para tener en cuenta los valores anteriores
del ángulo y eliminar ruido y vibraciones del vehículo

angulo_nuevo = angulo*coeficiente + angulo_anterior*(1-coeficiente);
angulo_anterior = angulo_nuevo;
```

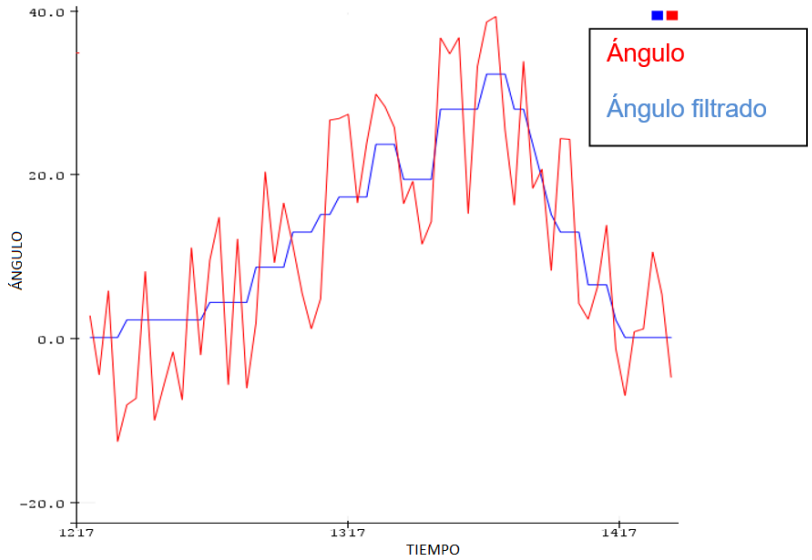


Ilustración 6-5:Ángulo filtrado

7 Conclusiones

El objetivo de este trabajo fin de grado era implementar un sistema capaz de mantener el equilibrio sobre dos ruedas. A la hora de la elección, me pareció un proyecto muy interesante ya que la rama de la ingeniería electrónica industrial que más me llama la atención es aquella relacionada con el control, la automática y la robótica. Además, este trabajo no consistía solamente en un estudio teórico del péndulo invertido, sino que había que implementar un dispositivo real con lo que verdaderamente iba a aprender en qué se basa dicha teoría ya que como dijo Confucio: "*dímelo y lo olvidaré, muéstramelo y lo recordaré, involúcrame y lo aprenderé*".

Durante la realización de este proyecto me he dado cuenta de lo importante que es tener una base teórica de aquello que vas a realizar, es decir, la importancia de todo lo que he aprendido durante las asignaturas del grado. Pero esta base teórica no lo es todo a la hora de realizar un proyecto de ingeniería, es necesario saber aplicar esta teoría a la práctica y ahí es donde entra el ingenio del ingeniero.

En este proyecto, me he tenido que enfrentar a varios retos gracias a los cuales he vivido de primera mano lo que es construir un dispositivo desde cero. Todo comienza con la búsqueda de información, y una vez crees que tienes toda la información necesaria puedes empezar a trabajar. Sin embargo, en la mayoría de los casos, las cosas no funcionan como deberían por lo que hay que reformularlo todo y comenzar de nuevo. En definitiva, me he dado cuenta que cometiendo muchos fallos es como verdaderamente se aprende. Por ejemplo, en este proyecto me he tenido que enfrentar a retos como:

- Diseño de circuitos en placa perforada.
- Soldar componentes.
- Diseño e impresión 3D de piezas.
- Programación del sistema de control.
- Elección y configuración de los componentes electrónicos.
- Manejo y configuración de timers y registros internos de Arduino.
- Interpretación y extracción de información de hojas de características.
- Comunicación y manejo vía bluetooth desde smartphone.

Con todo esto, podemos decir que ha sido un TFG que engloba a muchas disciplinas dentro de la ingeniería y con el que realmente he aprendido conceptos tan importantes como el control PID. Me siento muy satisfecho de ver cómo el trabajo duro, la dedicación y la pasión de hacer aquello que realmente te gusta, finalmente da sus resultados.

Como dijo Albert Einstein: "*Hay una fuerza motriz más poderosa que el vapor, la electricidad y la energía atómica: la voluntad"*.

8 Bibliografía

[1] O. Boubaker and R. Iriarte, “The inverted pendulum in control theory and robotics : from theory to new innovations,” p. 394.

[2] J. K. Roberge, “The Mechanical Seal, Bachelor’s thesis,” 1960.

[3] R. H. C. J. F. Schaefer, “On the control of unstable mechanical systems,” vol. 3, pp. 12–24, 1966.

[4] M. B. Ortiz Moctezuma, *Sistemas dinámicos en tiempo continuo: Modelado y simulación*. 2015.

[5] “Segway.” [Online]. Available: https://es.wikipedia.org/wiki/Segway.

[6] “Hoverboard.” [Online]. Available: https://es.wikipedia.org/wiki/Tabla_de_dos_ruedas_autoequilibrada.

[7] K. Ogata, *Ingeniería de control*, Tercera. Prentice Hall.

[8] “Control PID.” [Online]. Available: https://www.picuino.com/es/arduprog/control-pid.html.

[9] D. Brett, “Arduino PID - Guía de uso de la librería,” pp. 1–28, 2011.

[10] J. L. Lagrange and W. R. Hamilton, “Dinámica analítica,” 1865.

[11] “Arduino.” [Online]. Available: https://www.arduino.cc/.

[12] “Qué es arduino, cómo funciona y qué puedes hacer con uno.” [Online]. Available: https://www.xataka.com/basics/que-arduino-como-funciona-que-puedes-hacer-uno.

[13] “Aprendiendo Arduino HC05.” [Online]. Available: https://aprendiendoarduino.wordpress.com/tag/hc-05/.

[14] L. Llamas, “Comunicación inalámbrica con NRF24L01.” [Online]. Available: https://www.luisllamas.es/comunicacion-inalambrica-a-2-4ghz-con-arduino-y-nrf24l01/.

[15] “Bus I2C.” [Online]. Available: https://aprendiendoarduino.wordpress.com/2016/11/14/bus-i2ctwi/.

[16] “Bluetooth.” [Online]. Available: https://es.wikipedia.org/wiki/Bluetooth.

[17] “Serial Bluetooth Controller.” [Online]. Available: https://play.google.com/store/apps/details?id=nextprototypes.BTSerialController&hl=es_419.

[18] “MPU6050 acelerómetro y giroscopio.” [Online]. Available: https://naylampmechatronics.com/blog/45_Tutorial-MPU6050-Acelerómetro-y-Giroscopio.html.

[19] L. Llamas, “Determinar la orientación con MPU6050.” [Online]. Available: https://www.luisllamas.es/arduino-orientacion-imu-mpu-6050/.

[20] “MPU6050 y su programación.” [Online]. Available: https://arduproject.es/wp-content/cache/page_enhanced/arduproject.es/mpu6050-y-su-programacion/_index.html_gzip.

[21] I. Inc., “MPU-6000 and MPU-6050 Product Specification,” *Inven. Inc. Prod. Specif.*, vol. 3.4, no. 408, pp. 1–57, 2013.

[22] B. Ave, D. Number, and R. Date, “MPU-6000 and MPU-6050 Register,” vol. 1, no. 408, pp. 1–46, 2013.

[23] “Motor de corriente continua.” [Online]. Available: https://es.wikipedia.org/wiki/Motor_de_corriente_continua.

[24] “Motor paso a paso.” [Online]. Available: https://www.prometec.net/motores-paso-a-paso/.

[25] L. Llamas, “Motores paso a paso con driver A4988 y DRV8825.” [Online]. Available: https://www.luisllamas.es/motores-paso-paso-arduino-driver-a4988-drv8825/.

[26] “Guía para comprar motor paso a paso.” [Online]. Available: https://www.staticboards.es/blog/motores-paso-paso/.

[27] MotionKing, “2 Phase Hybrid Stepper Motor,” p. 1106, 2012.

[28] Texas Instruments, “DRV8825 Stepper motor Controller IC,” *DRV8825 Stepper Mot. Controll. datasheet*, p. Texas Instruments. (2014, juli). DRV8825 STEPER MO, 2014.

[29] “DRV8825.” [Online]. Available: https://www.pololu.com/product/2133.

[30] L. Llamas, “Opciones para alimentar Arduino con baterías.” [Online]. Available: https://www.luisllamas.es/alimentar-arduino-baterias/.

[31] “La unidad C en las baterías.” [Online]. Available: http://aeromodelismo.epiel.com/c_baterias.html.

[32] “Batería de polímero de litio.” [Online]. Available: https://es.wikipedia.org/wiki/Batería_de_polímero_de_litiov.

[33] “Autodesk inventor.” [Online]. Available: https://es.wikipedia.org/wiki/Autodesk_Inventor.

[34] “Formato .STL.” [Online]. Available: https://es.wikipedia.org/wiki/STL.

[35] “Impresión 3D.” [Online]. Available: https://es.wikipedia.org/wiki/Impresión_3D.

[36] “Ultimaker Cura.” [Online]. Available: https://ultimaker.com/en/products/ultimaker-cura-software.

[37] “Fritzing.” [Online]. Available: http://fritzing.org/home/.

[38] A. V. R. Microcontroller, “ATmega328P,” pp. 1–294.

[39] C. Gonzalez, I. Alvarado, and D. M. La Peña, “Low cost two-wheels self-balancing robot for control education,” *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 9174–9179, 2017.

ANEXO

PRESUPUESTO

En este anexo aparece detallado el coste de los distintos materiales utilizados en el proyecto.

Nota: Los precios ya incluyen el IVA.

	Precio unitario	Unidades	Precio total
Arduino Nano	20,00 €	1	20,00 €
DRV8825	3,46 €	2	6,92 €
NEMA 17	12,5 €	2	25,00 €
MPU 6050	2,11 €	1	2,11 €
Batería	16,83 €	1	16,83 €
HC - 05	3.93 €	1	3,93 €
MP1584EN	1,05 €	1	1,05 €
Interruptor	1,30 €	1	1,30 €
Pines PCB	1,31 €	1	1,31 €
Cables	2,00 €	1	2,00 €
PCB	3,65 €	1	3,65 €
Tornillos M3x12mm	0,08 €	20	1,60 €
Tuercas M3	0,07 €	12	0,84 €

Precio total	**86,54 €**

Coste aproximado de impresión de piezas en 3D. Fuente

	Precio unitario	Unidades	Precio total
Base1.2	10,35 €	1	10,35 €
Base1	10,09 €	1	10,09 €
Base2	9,94 €	1	9,94 €
Lateral1	13,11 €	1	13,11 €
Lateral2	13,22 €	1	13,22 €
Rueda	9,08 €	2	18,16 €
Soporte	3,26 €	1	3,26 €
Tapa	6,83 €	1	6,83 €
		Precio total	**84,96 €**

Materiales	86,54 €
Impresión 3D	84,96 €
Total proyecto	**171,5 €**

Asciende el presente presupuesto a la expresada cantidad de CIENTO SETENTA Y UN EUROS con CINCUENTA CÉNTIMOS.

ANEXO

CÓDIGO

```
/**
 *  VEHÍCULO AUTOPROPULSADO BASADO EN EL PÉNDULO INVERTIDO
 *  TRABAJO FIN DE GRADO INGENIERÍA ELECTRÓNICA INDUSTRIAL
 *  2019
 *  REALIZADO POR: MANUEL JOAQUÍN PÉREZ SERRANO
 */
#include <Wire.h>              //Librería necesaria para comunicarnos con el mpu6050
#include <SoftwareSerial.h>    //Librería necesaria para comunicación bluetooth


//Variables HC05
SoftwareSerial HC05(2,3);      // Tx pin 2; Rx pin 3
char dato = 0;

//Variables MPU6050
#define rad_a_gra 57.3248
#define offset 855
int ac_x,ac_y,ac_z;
int giro_y;
float ang_ac, ang_giro, angulo, angulo_nuevo, angulo_anterior = 0;
long tiempo_prev;
float dt;
float coeficiente = 0.75;

//Step y dir DRV8825
#define step_der 7
#define dir_der 6
#define step_izq 5
#define dir_izq 4
#define led 13


//Variables motores paso a paso
int cont_step_der, max_cont_step_der, pasos_der, motor_der;
int cont_step_izq, max_cont_step_izq, pasos_izq, motor_izq;

//Ganancias PID
float kp = 4.8;        //Ganancia proporcional
float ki = 3.5;        //Ganancia integral
float kd = 6.4;        //Ganancia derivativa

//Variables PID
float salida_PID, salida_PID_izq, salida_PID_der, referencia = 0.0,
referencia_automatica;
float integral;
float error, error_anterior;

int bateria;
unsigned long loop_timer;
```

```
void setup() {
  HC05.begin(9600);           //Comenzamos comunicación con HC05 a 9600 baudios
  Serial.begin(9600);         //Comenzamos comunicación serie a 9600 baudios
  TWBR = 12;                  //Configuramos la frecuencia de comunicación I2C a
                              //400kHz (MPU 6050)
  setup_timer2();             //Configuramos timer 2 modo comparación para crear
                              //pulsos para motores
  setup_registros_mpu6050();  //Inicio y configuración MPU6050

  pinMode(step_der, OUTPUT);  //Configuración salidas digitales
  pinMode(dir_der, OUTPUT);
  pinMode(step_izq, OUTPUT);
  pinMode(dir_izq, OUTPUT);
  pinMode(led, OUTPUT);

  loop_timer = micros() + 2000;     //Inicializamos variable loop_timer (2ms)
}

void loop() {

  bateria = analogRead(0);              //Batería conectada al puerto A0

  if(bateria < 800 && bateria> 700){    //Si la tensión de la batería baja de 9.5 V
    digitalWrite(13, HIGH);             //Se enciende el LED
    motor_der = 0;                      //Se paran ambos motores
    motor_izq = 0;
  }

  datos_mpu6050();                      //Obtenemos el ángulo de inclinación

  Serial.print("angulo:");Serial.print(angulo_nuevo);Serial.print("\t");
  Serial.print("PID:");Serial.print(salida_PID);Serial.print("\t");
  Serial.print("motor: ");Serial.println(motor_der);Serial.print("\n");


  //PID
  error = angulo_nuevo - referencia_automatica - referencia;            //Error

  //Para evitar que el error se descontrole
  if(salida_PID > 10 || salida_PID < -10)error += salida_PID * 0.02 ;
  integral += ki*error;                                          //Componente integral
  if (integral > 400) integral = 400;                            //Anti Wind-up
  else if (integral < -400) integral = -400;

  salida_PID = kp*error + integral + kd*(error - error_anterior);  //Salida del PID
 if (salida_PID > 400) salida_PID = 400;                           //Anti Wind-up
  else if (salida_PID < -400) salida_PID = -400;

  error_anterior = error;                                  //Actualización del error

//Con lo siguiente, conseguimos un comportamiento más suave y que no esté
//constantemente vibrando
  if(salida_PID < 15 && salida_PID > -15)salida_PID = 0;

 if(angulo > 37 || angulo < -37){            //Si se cae, que se paren los motores
    salida_PID = 0;
    integral = 0;
  }

  if(referencia == 0){
    //Se incrementa la referencia automática si el robot va hacia delante
    if(salida_PID < 0)referencia_automatica += 0.02;
    //Se decrementa la referencia automática si el robot va hacia atrás
    if(salida_PID > 0)referencia_automatica -= 0.02;
 }

   salida_PID_izq = salida_PID;
   salida_PID_der = salida_PID;
```

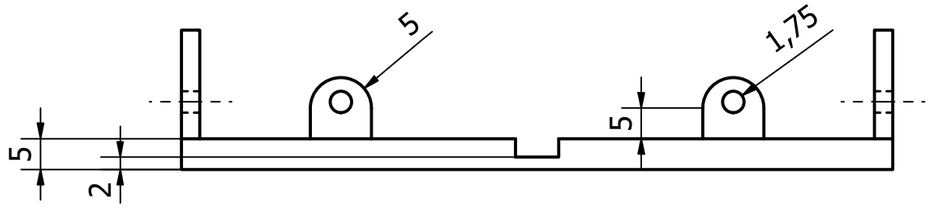
```
  //Control Bluetooth
  if (HC05.available()){
    dato = HC05.read();
  }
  if (dato == '0'){                                  //STOP
       referencia = 0;
  }
  if (dato == '1'){                                  //Adelante
       referencia -= 0.1;
       if (referencia < -3) referencia = -3;
  }
  if (dato == '2'){                                  //Atrás
       referencia += 0.1;
       if (referencia > 3) referencia = 3;
  }
  if (dato == '3'){                                  //Izquierda
       salida_PID_izq -= 25;
       salida_PID_der += 25;
  }
  if (dato == '4'){                                  //Derecha
       salida_PID_izq += 25;
       salida_PID_der -= 25;
  }

 //Para compensar el comportamiento no lineal de los motores
  if(salida_PID_izq > 0)salida_PID_izq = 395 - (1/(salida_PID_izq + 10))*5000;
  else if(salida_PID_izq < 0)salida_PID_izq = - 395 - (1/(salida_PID_izq -
  10))*5000;

  if(salida_PID_der > 0)salida_PID_der = 395 - (1/(salida_PID_der + 10))*5000;
  else if(salida_PID_der < 0)salida_PID_der = - 395 - (1/(salida_PID_der -
  10))*5000;

  //Calculamos el tiempo entre pulsos de los motores
  if(salida_PID_izq > 0)motor_izq = 400 - salida_PID_izq;
  else if(salida_PID_izq < 0)motor_izq = - 400 - salida_PID_izq;
  else motor_izq = 0;

  if(salida_PID_der > 0)motor_der =  400 - salida_PID_der;
  else if(salida_PID_der < 0)motor_der =  - 400 - salida_PID_der;
  else motor_der = 0;

  pasos_izq = motor_izq;
  pasos_der = motor_der;

  while(loop_timer > micros());         //Timer 2ms
  loop_timer += 2000;
}


/**
   RUTINA DE INTERRUPCIÓN ASOCIADA AL TIMER 2. SE EJECUTA CADA 20 us
*/
ISR(TIMER2_COMPA_vect) {
  /**
 En esta rutina calculamos cada cuánto tiempo se envía un pulso al DRV8825 que
 equivale a un paso del motor. Así, controlando la frecuencia entre pasos,
 controlamos la velocidad de giro del motor.
  */

  //Motor derecho
  //Cada vez que entra se incrementa el contador (equivale a un paso)
  cont_step_der ++;
  if (cont_step_der > max_cont_step_der) {   //No se da un paso hasta que el
                                             //contador no supera el tiempo
                                             //(20us*max_cont_der) que queremos
                                             //entre pasos
    cont_step_der = 0;                       //Se resetea el contador
```

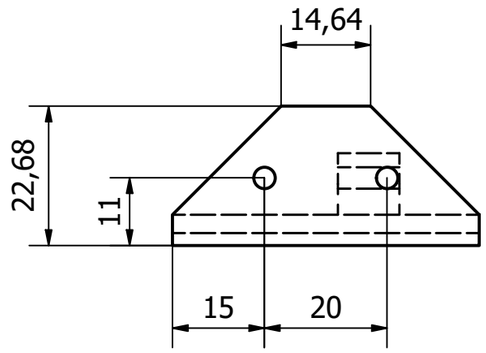
```
    max_cont_step_der = pasos_der;      //Se establece un nuevo tiempo entre pasos
    if (max_cont_step_der < 0) {        //Si max_cont_step_der es negativa
      PORTD &= (~(1 << dir_der));   //Ponemos dir_der a low para invertir dirección
      max_cont_step_der *= -1;      //Complementamos max_cont_step_der
    }
    else PORTD |= (1 << dir_der);
  }
  else if (cont_step_der == 1)PORTD |= (1 << step_der);//Se manda un pulso al motor
  else if (cont_step_der == 2)PORTD &= (~(1 << step_der));


  //Motor izquierdo
 //Cada vez que entra se incrementa el contador (equivale a un paso)
  cont_step_izq ++;
  if (cont_step_izq > max_cont_step_izq) { //No se da un paso hasta que el contador
                                           //no supera el tiempo(20us*max_cont_der)
                                           //que queremos entre pasos
    cont_step_izq = 0;                     //Se resetea el contador
    max_cont_step_izq = pasos_izq;         //Se establece un nuevo número de pasos
    if (max_cont_step_izq < 0) {           //Si max_cont_step_der es negativa
      PORTD &= (~(1 << dir_izq));   //Ponemos dir_der a low para invertir dirección
      max_cont_step_izq *= -1;      //Complementamos max_cont_step_izq
    }
    else PORTD |= (1 << dir_izq);
  }
  else if (cont_step_izq == 1)PORTD |= (1 << step_izq);//Se manda un pulso al motor
  else if (cont_step_izq == 2)PORTD &= (~(1 << step_izq));
}


/**
   RUTINA QUE CONFIGURA E INICIALIZA MPU6055
*/
void setup_registros_mpu6050() {
  //Activamos MPU-6050
  Wire.beginTransmission(0x68);//Empezamos comunicación (0x68 es la dirección del
                               //MPU-6050)
  Wire.write(0x6B);            //Escribimos en el registro 0x6B
  Wire.write(0x00);            //Ponemos todos los bits a 0
  Wire.endTransmission();      //Terminamos comunicación
  //Configuramos un filtro para eliminar posible ruido
  Wire.beginTransmission(0x68);         //Empezamos comunicación
  Wire.write(0x1A);                     //Escribimos en el registro 0x1A
  Wire.write(0x03);           //Seleccionamos filtro paso baja de ~43Hz  (00000011)
  Wire.endTransmission();               //Terminamos comunicación
  //Establecemos sensibilidad del giroscopio(250dps)
  Wire.beginTransmission(0x68);         //Empezamos comunicación
  Wire.write(0x1B);                     //Escribimos en el registro 0x1B
  Wire.write(0x00);              //Seleccionamos sensibilidad de 250dps (00000000)
  Wire.endTransmission();               //Terminamos comunicación
  //Establecemos sensibilidad del acelerómetro(+/-4g)
  Wire.beginTransmission(0x68);         //Empezamos comunicación
  Wire.write(0x1C);                     //Escribimos en el registro 0x1C
  Wire.write(0x08);               //Seleccionamos sensibilidad de +/-4g (00001000)
  Wire.endTransmission();               //Terminamos comunicación
}


/**
   RUTINA QUE LEE LOS VALORES "CRUDOS" DEL MPU6050 Y CALCULA EL ÁNGULO DE
INCLINACIÓN
*/
void datos_mpu6050() {
  Wire.beginTransmission(0x68);       //Empezamos comunicación
  Wire.write(0x3B);                   //Empezamos por el registro 0x3B (ACCEL_XOUT)
  Wire.endTransmission();             //Terminamos comunicación
  Wire.requestFrom(0x68, 6); //Obtenemos los primeros 6 registros a partir del 0x3B
  ac_x = Wire.read() << 8 | Wire.read(); //Combinamos los registros 0x3B y 0x3C
                                         //obteniendo la aceleración en X
```

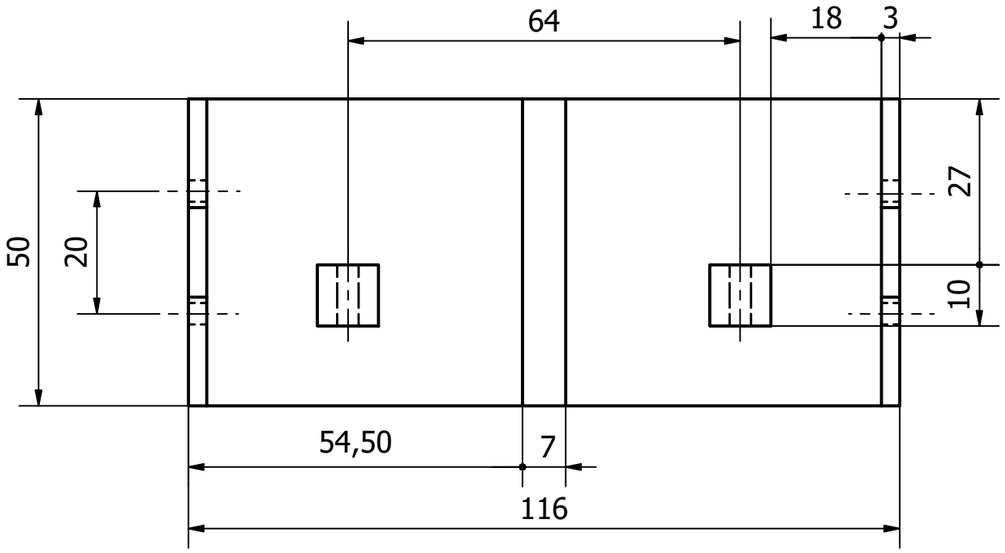
```
  ac_y = Wire.read() << 8 | Wire.read(); //Combinamos los registros 0x3D y 0x3E
                                         //obteniendo la aceleración en Y
  ac_z = Wire.read() << 8 | Wire.read(); //Combinamos los registros 0x3F y 0x40
                                         //obteniendo la aceleración en Z

  //Ángulo acelerómetro
  ang_ac = atan(ac_z / sqrt(pow(ac_x,2) + pow(ac_y,2)))*rad_a_gra;
  Wire.beginTransmission(0x68);      //Empezamos comunicación
  Wire.write(0x45);                  //Empezamos por el registro 0x45 (GYRO_YOUT_H)
  Wire.endTransmission();            //Terminamos comunicación
  Wire.requestFrom(0x68, 2);         //Obtenemos los dos registros siguientes a 0x45
  giro_y = Wire.read() << 8 | Wire.read();//Combinamos los registros 0x45 y 0x46
                                         //obteniendo el ángulo en Y

  //Ángulo giroscopio
  dt = (millis() - tiempo_prev);
  tiempo_prev = millis();
  giro_y += offset;                        //Offset
  ang_giro = angulo + (giro_y/131)*dt/1000.0;

  //Filtro complementario
  angulo = 0.95*ang_giro + 0.05*ang_ac;

  //Filtro complementario para tener en cuenta los valores anteriores del ángulo y
  //eliminar ruido y vibraciones del vehículo
  angulo_nuevo = angulo*coeficiente + angulo_anterior*(1-coeficiente);
  angulo_anterior = angulo_nuevo;
}

/**
   RUTINA QUE CONFIGURA EL TIMER 2 MODO COMPARACIÓN A
*/
void setup_timer2(){
  TCCR2A = 0;                        //Ponemos TCCR2A a cero
  TCCR2B = 0;                        //Ponemos TCCR2B a cero
  TIMSK2 |= (1 << OCIE1A);           //Activamos modo "compare match A interrupt" en
                                     //el registro TIMSK2
  TCCR2B |= (1 << CS21);             //Ponemos el bit CS21 del registro TCCRB a 1
                                     //para prescaler 8
  OCR2A = 39;                        //Para conseguir 20 us, registro de comparación
                                     //A = 39; (20us * (16.000.000MHz / 8)) - 1 = 39
  TCCR2A |= (1 << WGM21);
}
```

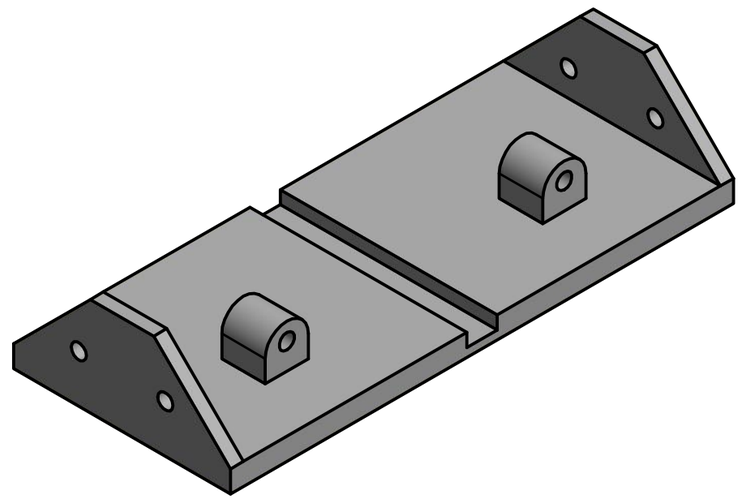
ANEXO

PLANOS





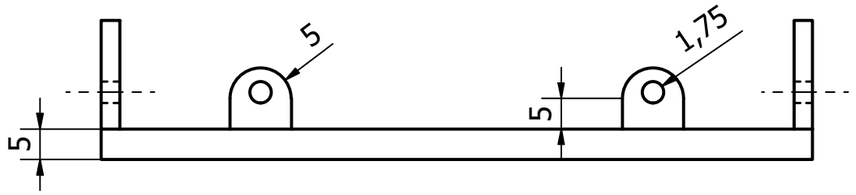


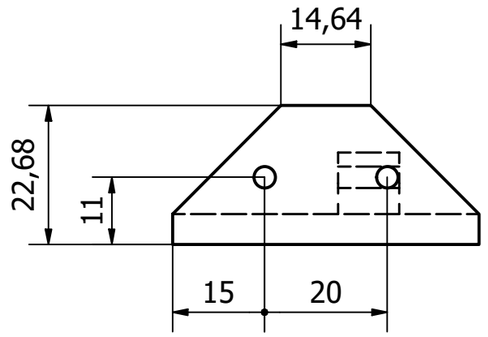


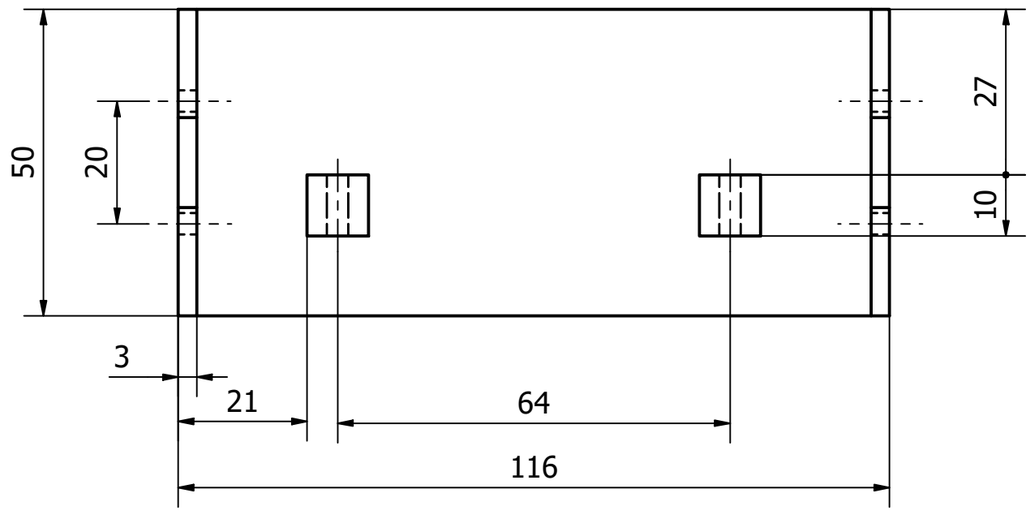
Apellidos, Nombre	Pérez Serrano, Manuel Joaquín
Fecha	Junio 2019
Departamento	Ingeniería electrónica y automática
Título	Vehículo autopropulsado basado en el péndulo invertido
Escala 1:1	Plano Base 1

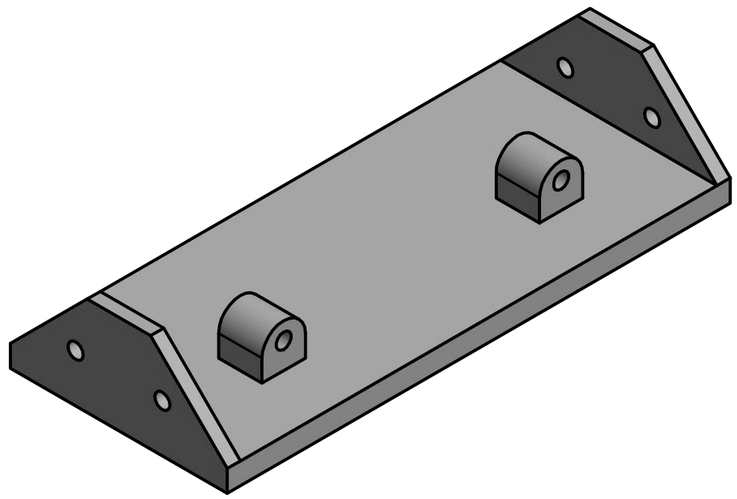
Universidad de Jaén

Escuela Politécnica Superior





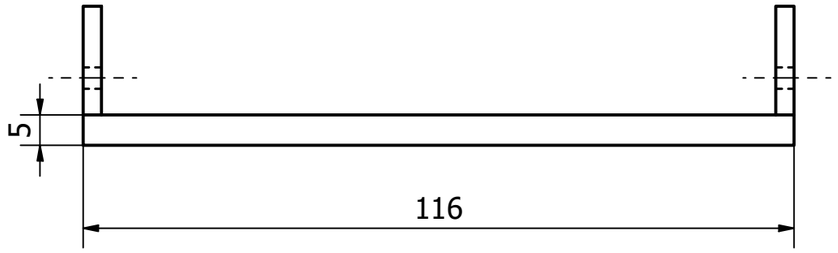


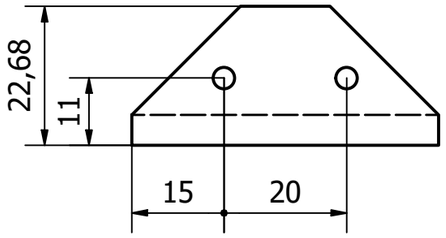


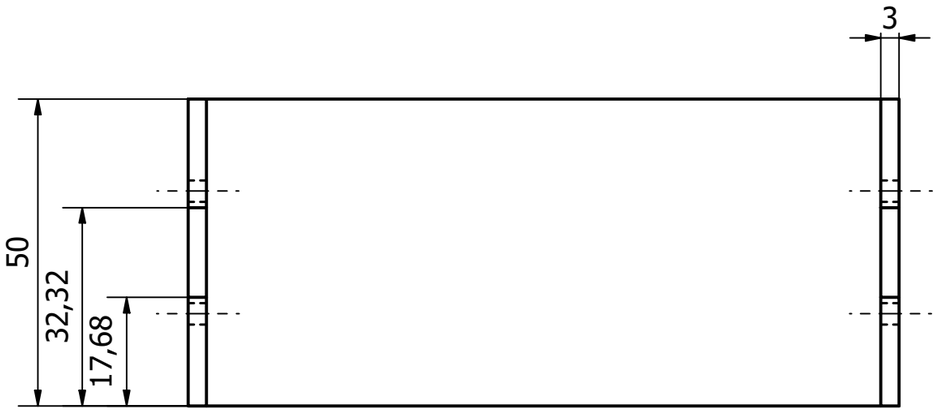
Apellidos, Nombre	Pérez Serrano, Manuel Joaquín
Fecha	Junio 2019
Departamento	Ingeniería electrónica y automática
Título	Vehículo autopropulsado basado en el péndulo invertido
Escala	1:1
Plano	Base 2

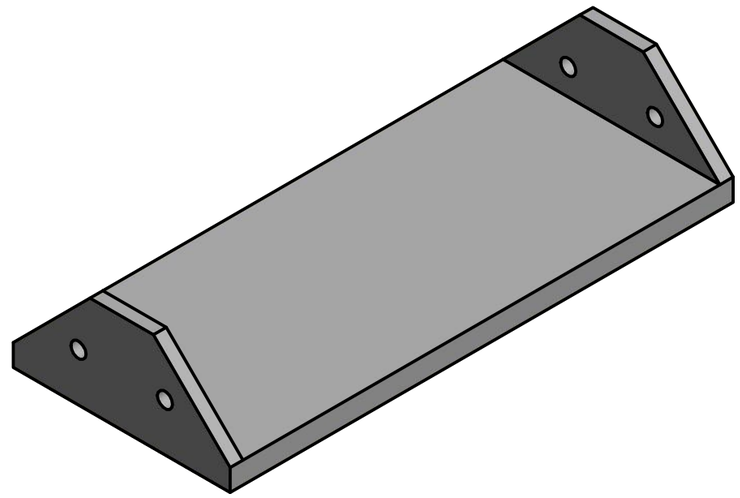
Universidad de Jaén

Escuela Politécnica Superior





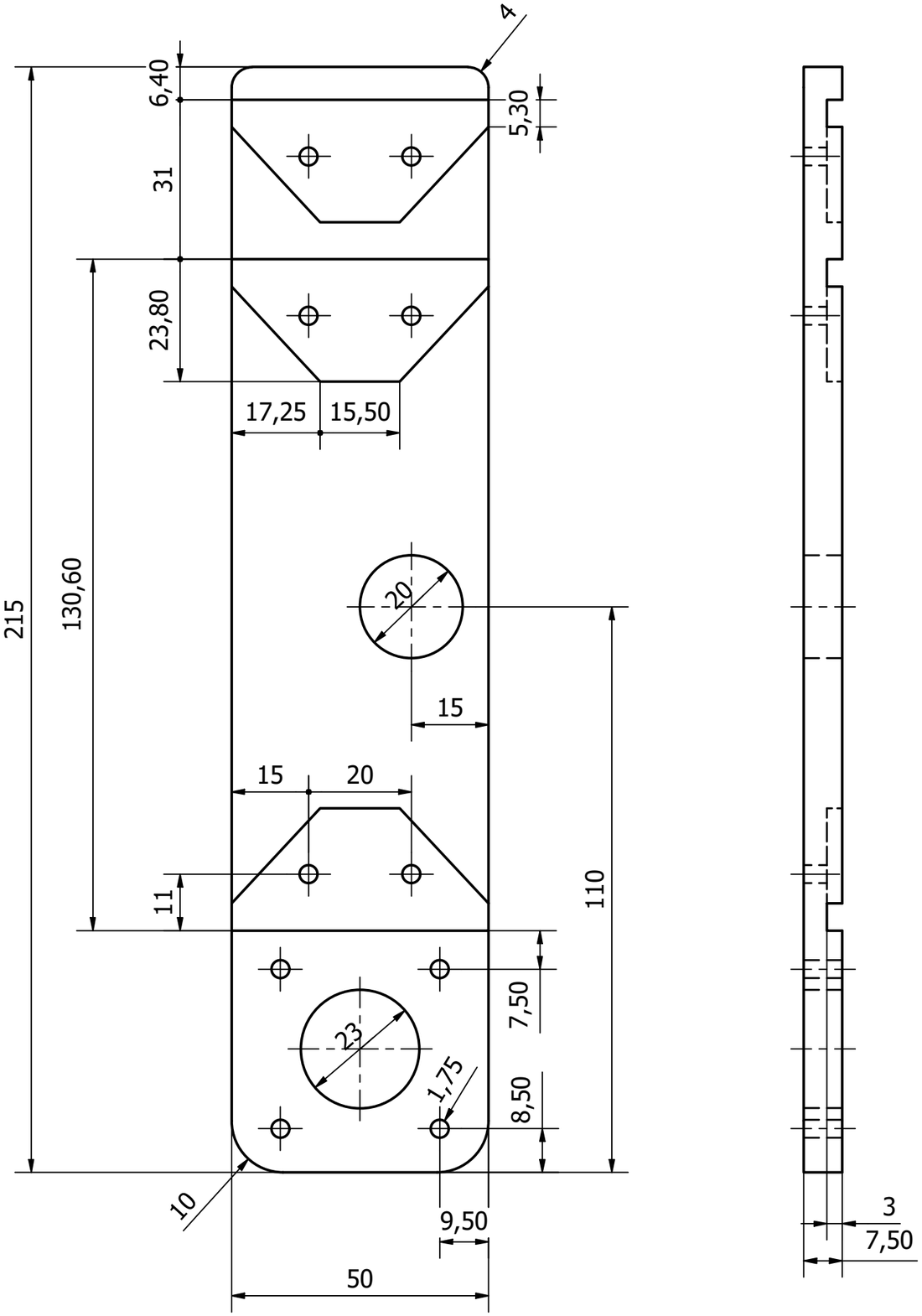


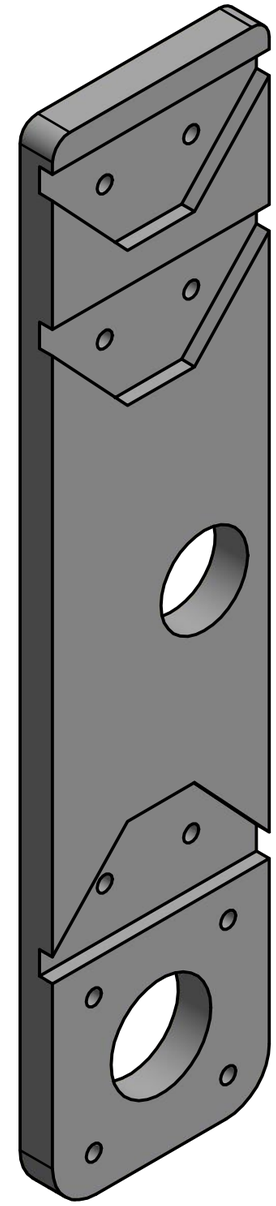


Apellidos, Nombre	Pérez Serrano, Manuel Joaquín
Fecha	Junio 2019
Departamento	Ingeniería electrónica y automática
Título	Vehículo autopropulsado basado en el péndulo invertido
Escala	Plano
1:1	Base 3

Universidad de Jaén

Escuela Politécnica Superior

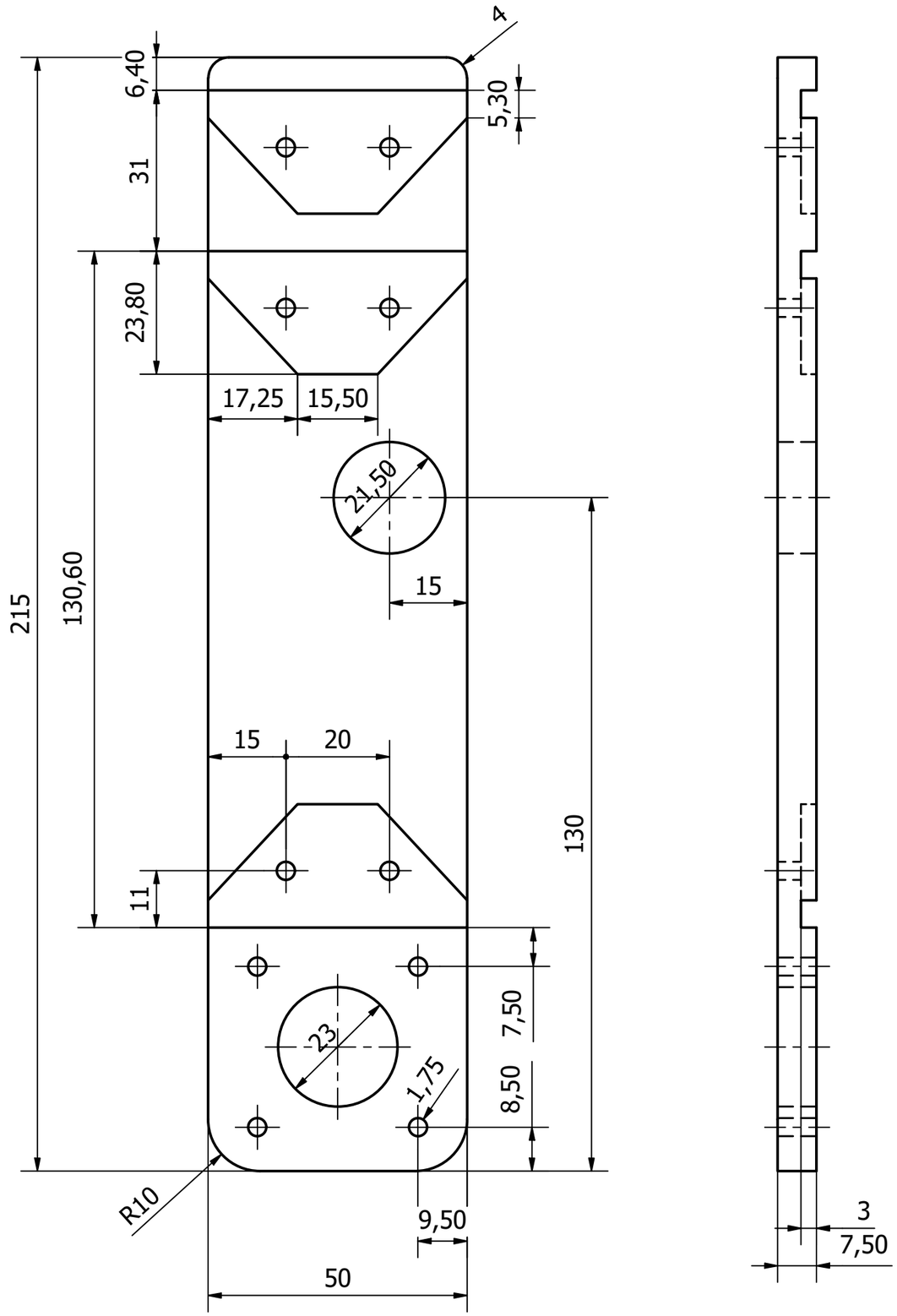


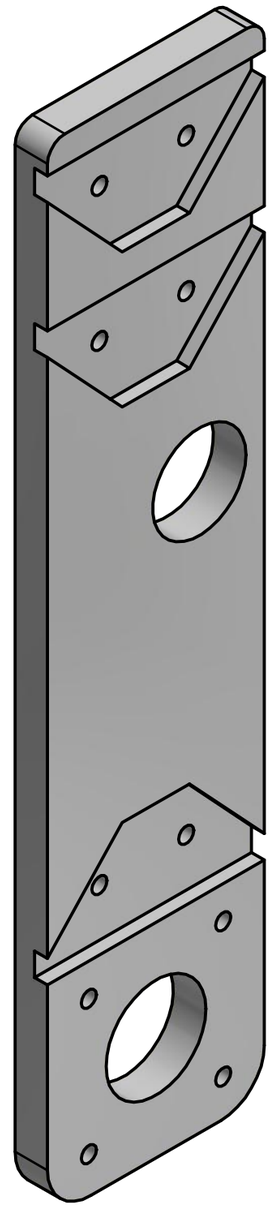


Apellidos, Nombre	Pérez Serrano, Manuel Joaquín
Fecha	Junio 2019
Departamento	Ingeniería electrónica y automática
Título	Vehículo autopropulsado basado en el péndulo invertido
Escala	1:1
Plano	Lateral derecho

Universidad de Jaén

Escuela Politécnica Superior

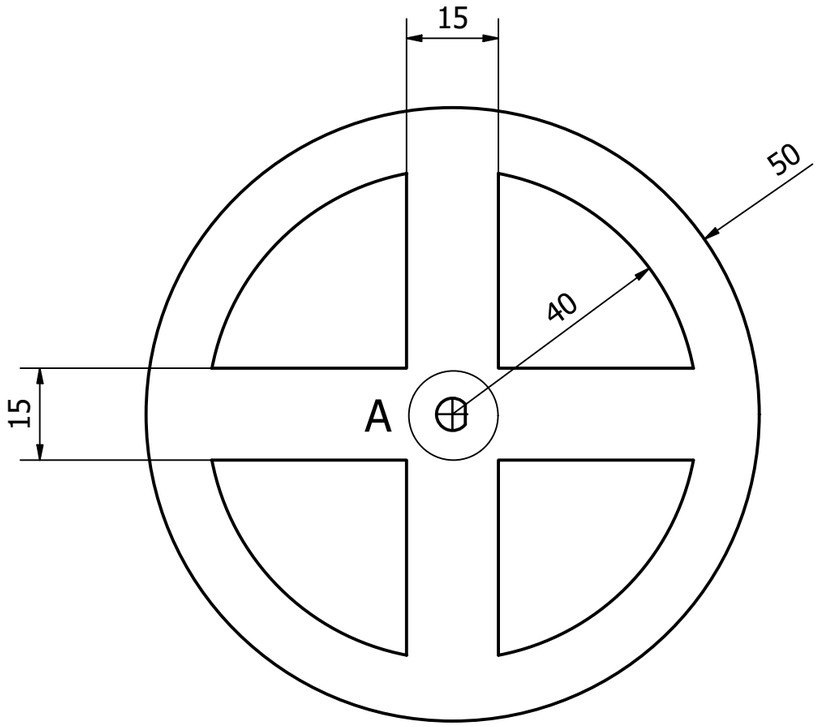


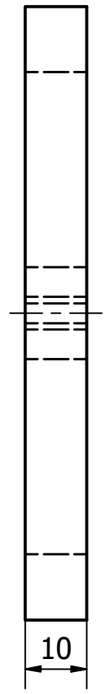


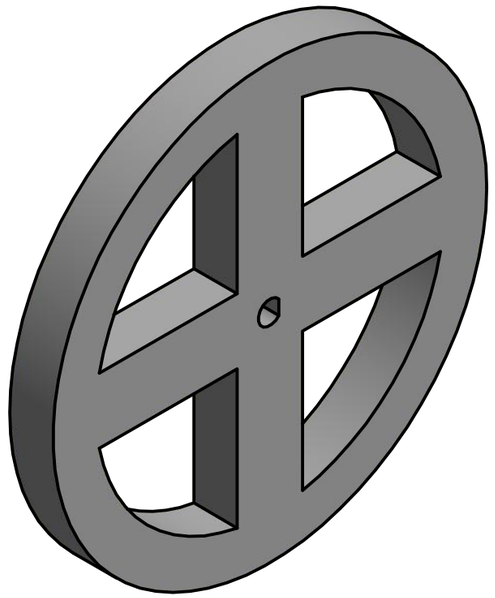
Apellidos, Nombre	Pérez Serrano, Manuel Joaquín
Fecha	Junio 2019
Departamento	Ingeniería electrónica y automática
Título	Vehículo autopropulsado basado en el péndulo invertido
Escala	1:1
Plano	Lateral izquierdo

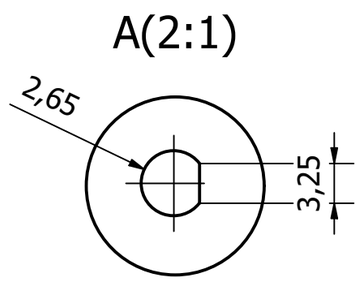
Universidad de Jaén

Escuela Politécnica Superior





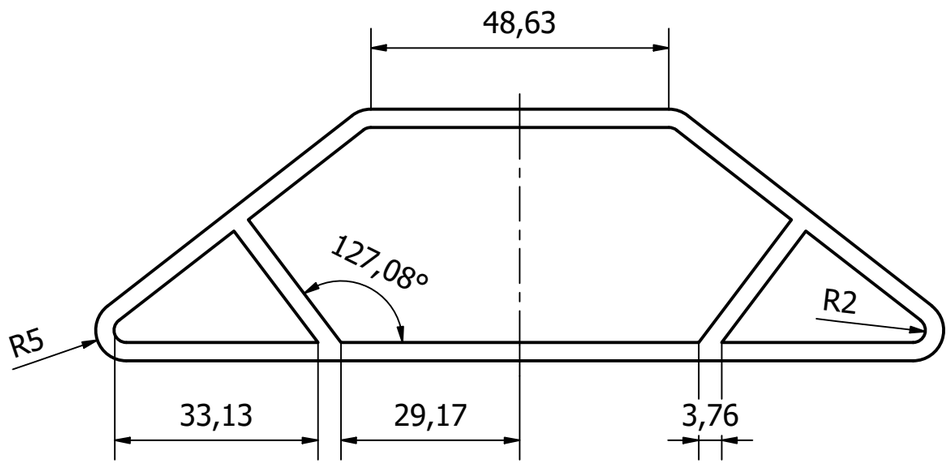


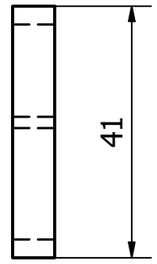


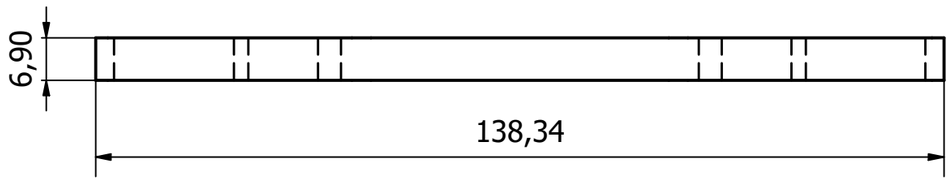
Apellidos, Nombre	Pérez Serrano, Manuel Joaquín
Fecha	Junio 2019
Departamento	Ingeniería electrónica y automática
Título	Vehículo autopropulsado basado en el péndulo invertido
Escala	Plano
1:1	Rueda

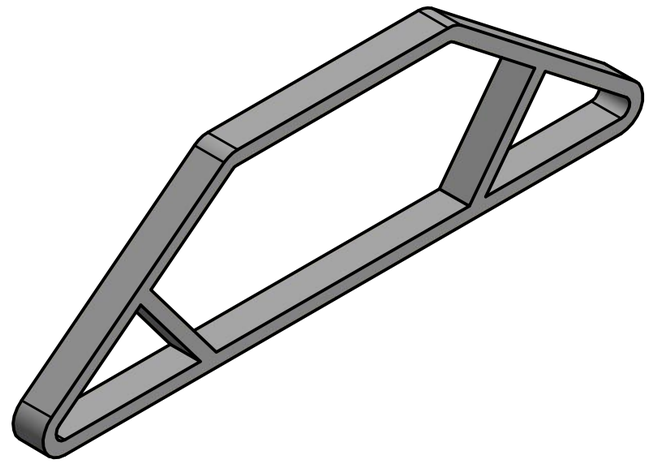
Universidad de Jaén

Escuela Politécnica Superior





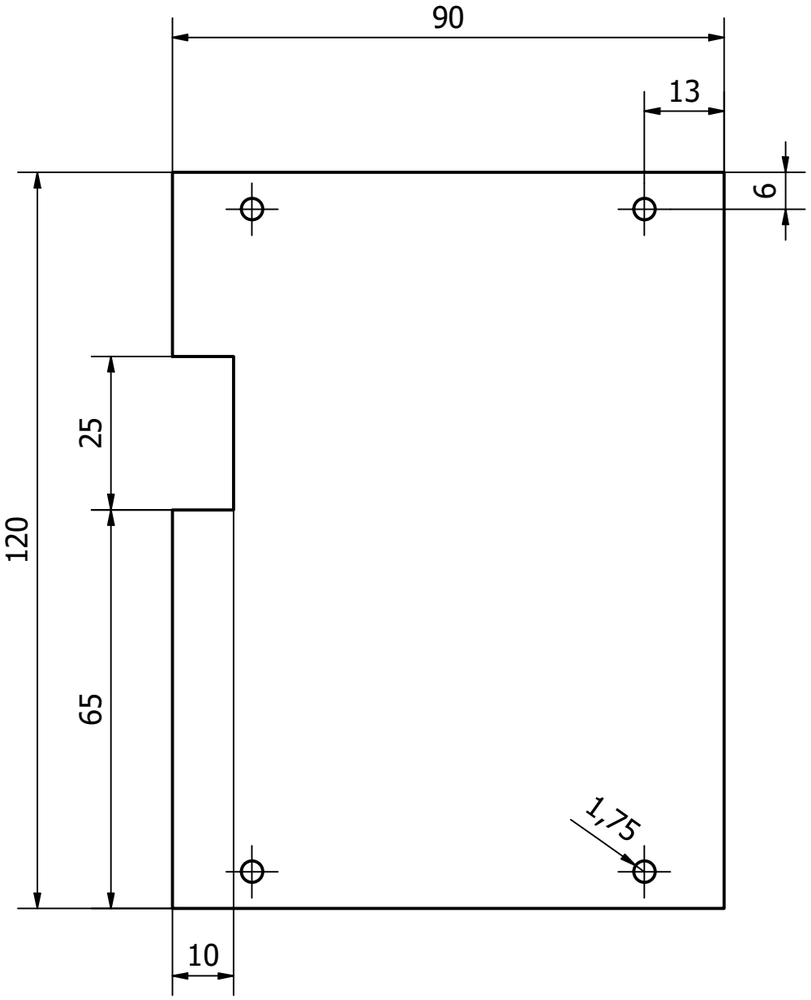


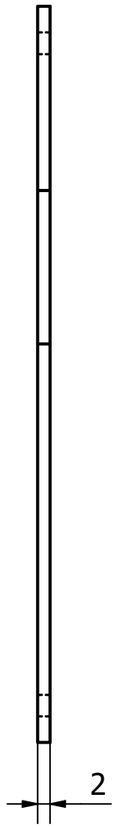


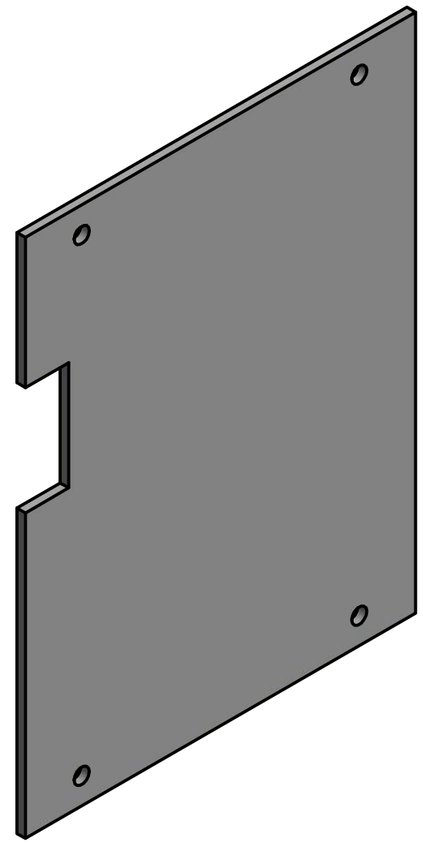
Apellidos, Nombre	Pérez Serrano, Manuel Joaquín
Fecha	Junio 2019
Departamento	Ingeniería electrónica y automática
Título	Vehículo autopropulsado basado en el péndulo invertido
Escala	Plano
1:1	SOPORTE

Universidad de Jaén

Escuela Politécnica Superior







Apellidos, Nombre	Pérez Serrano, Manuel Joaquín
Fecha	Junio 2019
Departamento	Ingeniería electrónica y automática
Título	Vehículo autopropulsado basado en el péndulo invertido
Escala	Plano
1:1	Tapa trasera

Universidad de Jaén

Escuela Politécnica Superior