Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería Electrónica y Automática Industrial

Trabajo Fin de Grado

Diseño e implementación de un algoritmo de control para un robot de dos ruedas

Autor: César William Lu Shih

Tutor/es: Melquíades Carbajo Martín

2021

UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería en Electrónica y Automática Industrial

Trabajo Fin de Grado

"Diseño e implementación de un algoritmo de control para un robot de dos ruedas"

Autor: César William Lu Shih

Tutor: Melquiades Carbajo Martín

Tribunal:

Presidente: Javier de Pedro Carracedo

Vocal 1º: Javier Alonso Ruiz

Suplente : Iván García Daza

Fecha de depósito: 23 septiembre 2021

Agradecimientos

Con este trabajo de fin de grado se termina una etapa importante de mi vida, pero también representa el inicio de nuevas oportunidades, nuevos conocimientos y nuevas experiencias que se nos presenta día a día las cuales hay que aprovechar.

Para finalizar este capítulo estudiantil quiero agradecer en primer lugar a mi familia que me ha proporcionado apoyo y ayuda de manera incondicional en todo momento.

Mencionar también a mis compañeros de clase los cuales han hecho esta etapa mucho más divertida y amena, en especial mención a Sergio, Laura, Almu, Pablo, Víctor con quienes siempre he compartido buenos momentos que jamás olvidaré.

También a mis amigos de toda la vida Ita, Guillermo, Álvaro, Rubén, Alejandro, María que da igual lo infinito que sea el tiempo sabes que siempre puedes contar con ellos.

Y una especial mención a Mónica que ha estado siempre en el final de esta etapa apoyándome en los momentos más duros y sobre todo motivándome con todo su amor y cariño a que me convierta en mejor estudiante y persona.

"Solamente hay una cosa en el mundo que ni el hombre más rico del mundo puede comprar, el tiempo."

ÍNDICE

ÍNDICE DE FIGURAS

Resumen

Este trabajo de fin de grado se basa principalmente en el estudio, diseño, modelado, y construcción de un robot de dos ruedas capaz de balancearse por sí mismo mediante técnicas de control.

Todo esto implementado en un microcontrolador capaz de responder y actuar ante perturbaciones externas registradas por un sensor de movimiento que detecta el ángulo de inclinación de dicho robot y mediante esta información actuar sobre los motores paso a paso.

Palabras clave: Arduino, Motor Stepper, Nema 17, MPU 6050, Controlador PID, Driver

Abstract

This Bachelor´s Degree Final Project is focused in the study, design, modelling and construction of a two wheeled robot capable of maintaining balance by itself using control techniques.

This is done by a microcontroller which is able to answer and respond to external perturbances registered by a movement sensor that can detect the angle of the robot´s tilt. With this information it will send and instruction for the stepper motors to act.

Keywords: Arduino, Motor Stepper, Nema17, MPU 6050, PID controller, Driver.

1 CAPITULO I : Introducción.

1.1 Principio de funcionamiento

La creación de los robots autobalanceables ha sido un gran hito de la ingeniería robótica ya que, gracias a estos simples y primeros modelos de robots, se han desarrollado y realizado nuevas tecnologías que han cambiado poco a poco la actualidad a nivel global.
Todo parte del simple concepto de equilibrar un péndulo invertido el cual es un sistema inestable por naturaleza. Se pretende mantener en posición vertical una masa sobre un punto de apoyo muy pequeño consiguiendo la estabilidad del sistema, es decir convertir lo inestable en estable.

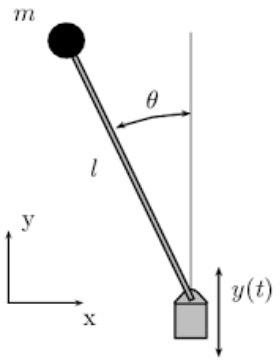


FIG. 1 PÉNDULO INVERTIDO

Dicho problema asociado a la estabilización del péndulo invertido es un problema muy básico y ampliamente estudiado dentro de la ingeniería de control el cual usará un sistema de control que actuará ante una entrada que en este caso sería el desbalanceo del péndulo y dará una salida o respuesta al sistema.
La mayoría de los robots que se autobalancean parten de resolver el problema de péndulo invertido, por lo que casi todos tienen una estructura similar entre ellos.
Se pueden distinguir varias cosas en común como pueden ser:
-Tienen un punto de apoyo muy reducido ya que generalmente suelen apoyarse sobre ruedas.
-Poseen un sensor capaz de detectar la posición del robot, tanto su inclinación como rotación.
-Están implementados con un sistema de control que más adelante se explicar

1.2 Aplicaciones

Gracias a estos estudios es posible tener a día de hoy muchos otros objetos que se basen en este principio como pueden ser los famosos Segways o los más actuales Hoverboard en los cuales una persona se monta y esta máquina es capaz de mantener el equilibrio además de desplazarse en todas direcciones, cambiando así la forma de desplazarse de las personas.

FIG. 2 HOVERBOARD

También se ven en ámbitos como en juguetes de radio control, teniendo como ejemplo al juguete BB8 de Star-Wars, el cual su mecanismo es más complicado, pero se basa tambien en el mismo principio de péndulo invertido y balanceo.

FIG. 3 ROBOT BB8

Si se desarrolla aún más estas ideas y estos principios se puede conseguir hitos como han realizado recientemente los ingenieros de Honda

Estos presentaron una motocicleta de dos ruedas capaz de mantener el equilibrio sin la ayuda de un tercer punto de apoyo como las motocicletas convencionales y todo con la ayuda de un sistema de control muy preciso y elaborado.

FIG. 4 MOTOCICLETA HONDA

Incluso avanzando aún más en los sistemas de control y la idea principal del péndulo se puede lograr auténticas proezas como ha realizado la empresa Boston Dynamic y su robot humanoide Atlas.

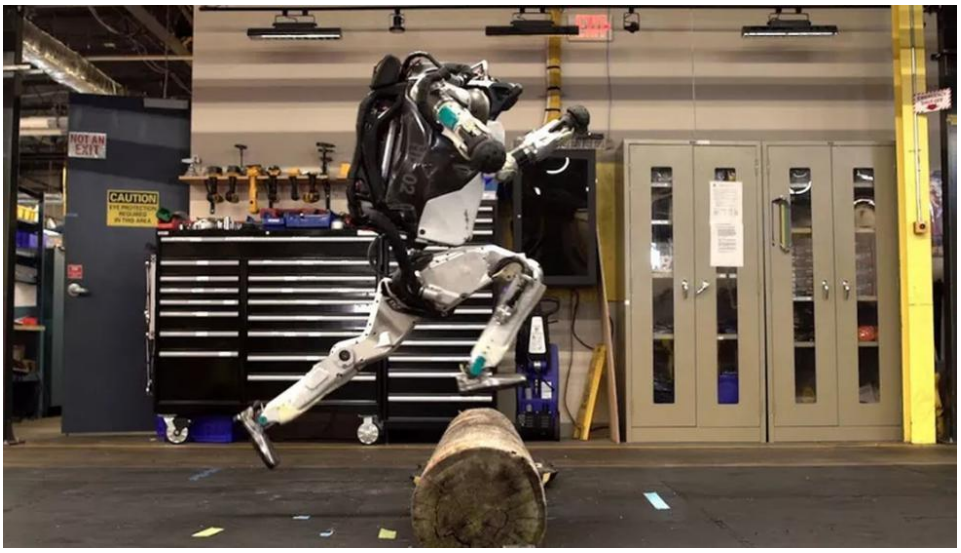


FIG. 5 ATLAS BOSTON DYNAMICS

El robot Atlas se puede moldear también como un péndulo invertido ya que el movimiento de andar o simplemente el hecho de mantenerse en pie hace que se comporte como un péndulo invertido donde el cuerpo del robot se balancea libremente sobre las piernas del robot manteniendo su postura.

1.3 Procesos.

El robot propuesto para este TFG es capaz de auto balancearse con la ayuda de un controlador PID implementado con la placa Arduino nano, el cual controlará todo el sistema recibiendo información de un IMU y actuando a través de dos motores steppers o paso a paso que sustentarán la estructura en conjunto.

Para realizar dicho proyecto se seguirá una serie de hitos a realizar:

-Estudio del modelo matemático

-Estudio del sistema de control

-Estudio de cada componente empleado.

-Conexionado de los dispositivos.

-Puesta en funcionamiento del IMU

-Puesta en funcionamiento de los motores paso a paso.

-Implementación en conjunto del IMU con los motores.

-Ensamblaje de la estructura principal con todos los componentes

-Sintonización del PID por prueba y error.

2 CAPITULO II : Estudio del sistema.

2.1 Modelo matemático.

El robot que diseñar se basa principalmente en un péndulo invertido, es por tanto que estudiaremos la física de este.

El centro de masas de un péndulo invertido está a mayor altura del punto sobre el que este se sostiene, por tanto, de manera natural se balanceará saliendo de la posición de equilibrio.

Realizaremos un programa de control con tal de evitar que esto ocurra y mantener en equilibrio a nuestro sistema.

El modelo matemático a estudiar dependerá de las condiciones iniciales de nuestro sistema. Partiendo de un péndulo estático en el que el péndulo solo se puede balancear en solo un plano y donde no se considera rozamiento ni se le aplica ninguna otra fuerza o momento, salvo el provocado por la gravedad y el ejercido por el coche para su equilibrio podemos determinar las ecuaciones de movimiento que rigen el sistema [5].

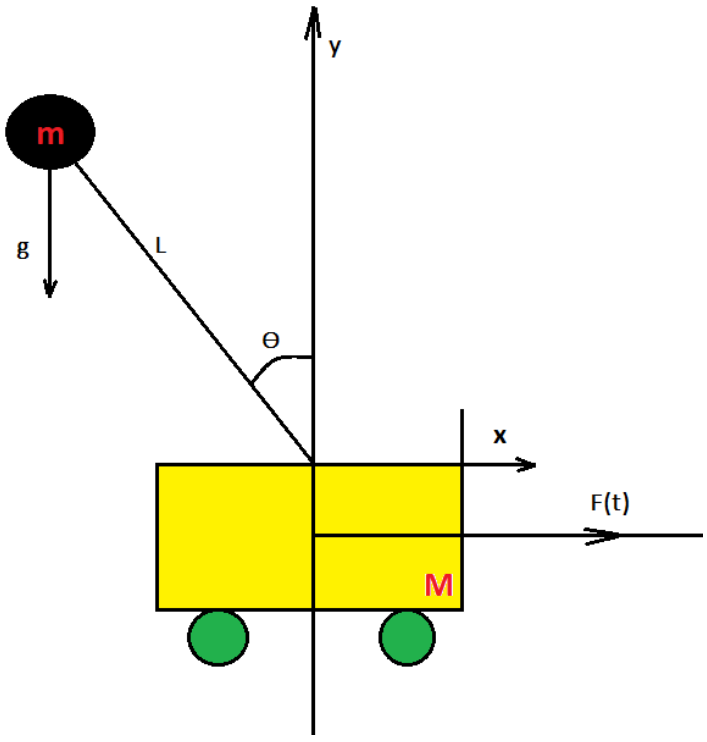


FIG. 6. ESQUEMA PÉNDULO INVERTIDO

Donde:

Θ : ángulo con respecto a la vertical del coche.

m : masa del péndulo

M : masa del carro

F : fuerza ejercida en el carro

L = longitud del péndulo

g = aceleración de la gravedad

Partiendo de que la masa "m" del péndulo se encuentra a una distancia (x_m , y_m) podremos expresar la distancia de la masa del péndulo con respecto al origen del sistema como:

$$x_m = x - l * sin\theta \qquad [1]$$

$$y_m = l * cos\theta \qquad [2]$$

Sabiendo su posición si se deriva una vez con respecto al tiempo se obtendría la velocidad:

$$\dot{x}_m = \dot{x} - l\dot{\theta}cos\theta \qquad [3]$$

$$\dot{y}_m = -l\dot{\theta}sin\theta \qquad [4]$$

Una vez tengamos posicionado las masas del sistema se procede a emplear la aproximación de Lagrange muy conveniente para este tipo de sistemas, planteándose en formas de energía potencial (V) y cinética (T).

Primero se halla la energía potencial obteniendo la siguiente ecuación:

$$V = mgy_m \qquad [5]$$

Sustituyendo la ecuación [2] sobre la [5] obtenemos:

$$V = mglcos\theta \qquad [6]$$

Para la energía cinética del sistema se obtiene la siguiente expresión:

$$T = \frac{1}{2}M\dot{x}^2 + \frac{1}{2}m(\dot{x}_m^2 + \dot{y}_m^2) \qquad [7]$$

Empleando las expresiones [3] y [4] sobre la ecuación [7], se obtiene la siguiente expresión:

$$T = \frac{1}{2}M\dot{x}^2 + \frac{1}{2}m\left(\dot{x}^2 - 2l\dot{\theta}\dot{x}cos\theta + l^2\dot{\theta}^2(cos^2\theta + sin^2\theta)\right) \quad [8]$$

Donde $(cos^2\theta + sin^2\theta)$ se sustituye directamente por 1 debido a las propiedades trigonométricas y agrupando terminos se obtiene:

$$T = \frac{1}{2}(M+m)\dot{x}^2 + \frac{1}{2}ml^2\dot{\theta}^2 - ml\dot{\theta}\dot{x}cos\theta \quad [9]$$

Una vez determinado la energía potencial y cinética del sistema se procede a implementarse la ecuación de Lagrange

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = Qi \quad [10]$$

Donde L es:

$$L = T - V \quad [11]$$

Si sustituimos las ecuaciones [6] y [9] se obtiene:

$$L = \frac{1}{2}(M+m)\dot{x}^2 + \frac{1}{2}ml^2\dot{\theta}^2 - ml\dot{\theta}\dot{x}cos\theta - mglcos\theta \quad [12]$$

Finalmente, para obtener las ecuaciones de movimiento emplearemos la ecuacion [10].

Para el término "x" se obtiene derivando L respecto a $\dot{x}$ es decir:

$$(M+m)\ddot{x} - ml\ddot{\theta}cos\theta + ml\dot{\theta}^2sin\theta = F(t) \quad [13]$$

Si consideramos que el ángulo de balanceo Θ es aproximadamente 0, podemos reducir la ecuación tal que así:

$$(M+m)\ddot{x} - ml\ddot{\theta} = F(t) \quad [14]$$

Esta fuerza F(t) será la que aplique las ruedas sobre el sistema gracias a la señal de control u(t) que se encarga de dar los valores al contador implementado en el ISR para controlar la aceleración de los motores mediante PWM.

La fuerza que proporcione los motores vendrá determinada por el momento de inercia y la aceleración angular que tenga las ruedas de los motores. El cual viene determinado de la siguiente manera:

$$\tau = I * \alpha \quad [15]$$

Donde *I* es la inercia de la rueda y α la aceleración angular que experimenta. Controlando los números de paso que dé el motor en un determinado tiempo podemos modificar la aceleración que experimente la rueda y por tanto controlar el torque o fuerza que experimenta el sistema.

El encargado de determinar el número de veces que se ponga el motor a nivel alto será el controlador PID.

Para representar el modelo completo del sistema se ha realizado el siguiente diagrama de bloques.

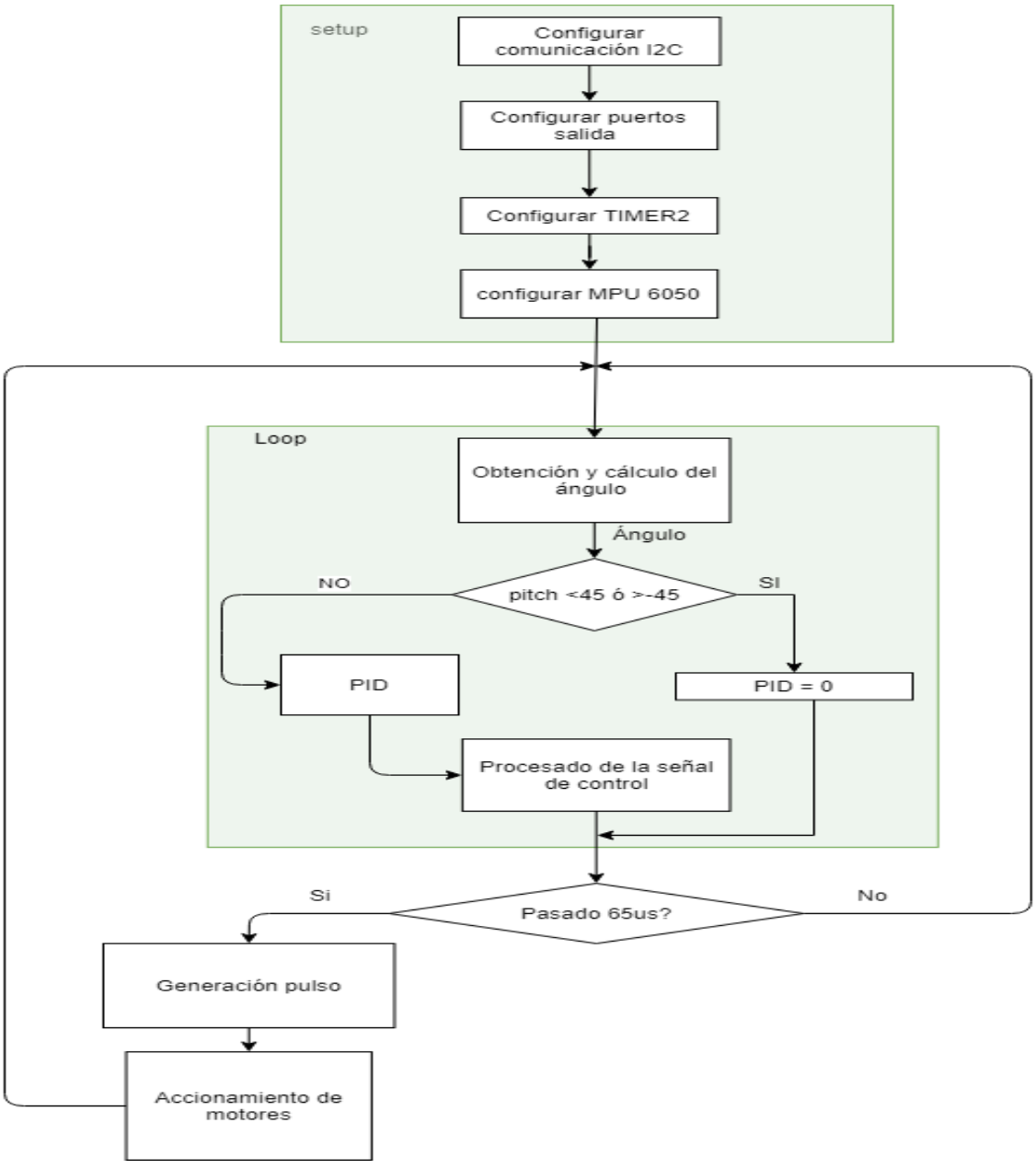


FIG. 7 DIAGRAMA DE FLUJO DEL SISTEMA

Donde la u de control es proporcionada por el PID que se implementa en los motores steppers, actuando directamente sobre el PWM de estos. El ángulo del pitch se obtiene como salida del sistema al que volvemos a insertar dentro del sistema a través del sensor formando así una realimentación de lazo cerrado.

2.2 Control del sistema.

Para controlar el sistema, va a ser necesario implementar un sistema de control que se encargue de corregir los errores que ocurran en el sistema, el cual en nuestro caso será la diferencia de ángulo deseado y el ángulo real medido.

Implementaremos un controlador PID, proporcional integrador y derivativo, el cual se encargará de enviar una respuesta de corrección sobre nuestro sistema.

Un controlador PID se puede representar principalmente con la siguiente ecuación:

$$u(t) = Kp * e(t) + Ki \int e(t) * dt + Kd * \frac{d}{dt} e(t) \qquad [4]$$

Para entender el sistema y la implementación de esta ecuación acudiremos a la siguiente imagen.

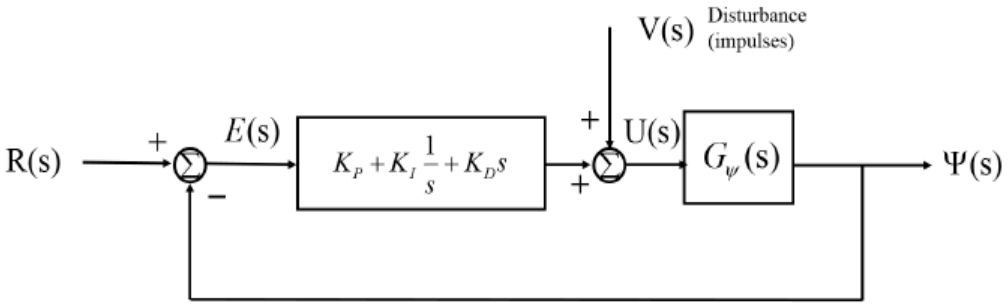


FIG. 8. ESQUEMA DE CONTROL

Como podemos observar, esta sería una de las representaciones gráficas de nuestro sistema completo.

La señal e(t), es el error del sistema, el cual es la diferencia de nuestro punto de referencia y del valor real que estamos obteniendo.

Esta señal de error se introduce en nuestro controlador PID, el cual realizará las operaciones necesarias y dará como respuesta una salida c(t), el cual será procesada por nuestro sistema, que hará funcionar a los actuadores.

El sensor recogerá dicha salida del sistema y lo introducirá de nuevo a la entrada para así tener un feedback o realimentación de lo que está ocurriendo y generar así una nueva señal de error.

Estudiaremos las siguientes partes para entender mejor el concepto del PID.

- Proporcional

La parte proporciona o P, realiza el producto del error por una constante proporcional Kp. El objetivo de este es intentar que el régimen estacionario sea nulo a la vez que se aumenta la velocidad de respuesta del sistema, esto puede ser ideal o no según qué respuesta queremos obtener, ya que puede generar sobrepicos indeseados o incluso llegar a un sistema oscilante o completamente inestable

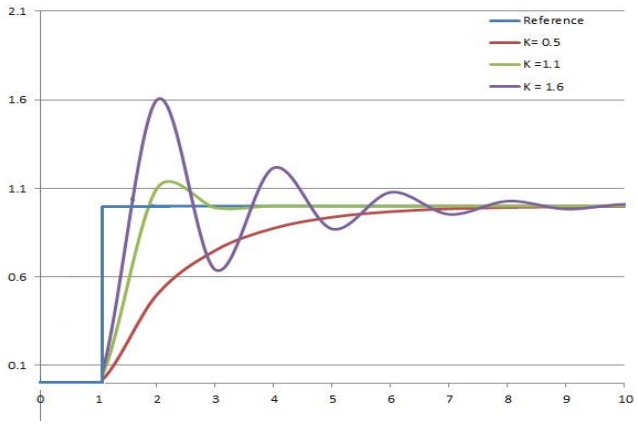


FIG. 9 RESPUESTA PROPORCIONAL

Como observamos en esta imagen, si variamos los valores de nuestra ganancia Kp veremos cómo se va comportando el sistema.

A mayor valor de Kp, vemos como alcanzamos el valor de referencia en un breve periodo de tiempo, incluso se llega a pasar produciendo un sobrepico. Obtenemos una respuesta muy rápida con un Kp alto, pero producimos oscilaciones a lo largo del tiempo. En cambio, un Kp bajo, consigue llegar al valor de referencia, pero en un tiempo mayor, sin llegar a provocar oscilaciones en el régimen permanente, estabilizándose en un tiempo menor.

- Integrador

La parte integral o I, es la encargada de sumar todos los errores a lo largo del tiempo. El resultado de esto es que incluso un error mínimo hará que la parte integral aumente poco a poco con el tiempo a no ser que el error sea cero. El objetivo de la parte integral es eliminar por completo el error en régimen permanente. Tener un integrador es beneficioso para el régimen permanente, pero puede provocar problemas en la estabilidad, ya que un error muy grande puede provocar que la parte integradora se vuelva completamente inestable.

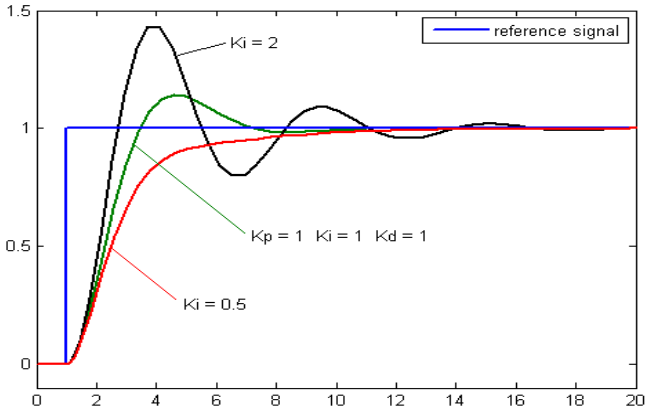


FIG. 10 RESPUESTA INTEGRADOR

Como se observa en la imagen de arriba, se ve como el integrador se encarga de mejorar el error en régimen permanente en el menor tiempo posible, pero al ser un integrador, la suma de dichos errores transitorios puede provocar oscilaciones muy severas si tenemos una ganancia excesivamente alta.

- Derivativo

La parte Derivativa o D, es el encargado de realizar ajustes sobre cambios bruscos del error, intentando aplanar la curva del error, reduciendo así el overshooting que se pueda producir ante una perturbación. Manteniendo el error al mínimo proporcionalmente con la misma velocidad que se produce.

El derivador calcula la pendiente del error con respecto al tiempo y multiplica esta proporción con la ganancia Kd. Esto provoca que intente "predecir" con antelación los comportamientos del sistema, mejorando así el tiempo de establecimiento y en general la estabilidad. Un problema de la parte derivativa es que es sensible al ruido que se pueda producir en el sistema y por ello reaccionar a dicho ruido de manera no deseada.

2.3 Sintonización del PID.

El proceso por de encontrar las ganancias adecuadas para el proporcional, integrador y derivativo se conoce como sintonización del PID. Hay varios métodos para encontrar dichos valores, como puede ser ensayo y error o el método de Zieger-Nichols.

- Zieger-Nichols

El método de Zieger-Nichols para un sistema de lazo cerrado consiste en poner a cero la ganancia del integral y del derivativo y ajustar la ganancia proporcional de tal manera que el sistema empiece a ser oscilante al límite de la inestabilidad hallando así la ganancia proporcional máxima K_{max} y la frecuencia de oscilación f_0. Una vez hallado K_{max} y f_0, se reduce de nuevo la ganancia proporcional y con la siguiente tabla obtendríamos los valores para el integrador y el derivador.

Controlador	K_P	K_I	K_D
P	$0.5*K_{max}$	-	-
PI	$0.45*K_{max}$	$T_0/1.2$	-
PID	$0.6*K_{max}$	$T_0/2$	$T_0/8$

Donde T_0 es el periodo máx. de oscilación, que viene de la siguiente expresión

$$w_0 = 2\pi/T_0 \quad (frecuencia\ critica)$$

$$F_0 = 1/T_0 \quad [5]$$

Aparte del método de Zieger-Nichols existen otros muchos métodos de sintonización basado en reglas como el Cohen-Coon, Kappa-Tau, o método Lambda.

Otra forma de sintonizar un PID es por el método Heurístico, en el cual la experiencia juega un papel importante.

Uno de estos métodos es basado en ensayo y error.

- Ensayo y error

Es un método relativamente sencillo una vez que se tenga asimilado el concepto de PID y cada una de sus partes, el procedimiento se basa en ir probando diferentes valores para cada una de las ganancias, empezando desde cero en cada una de ellas.

Por lo general para empezar se ajusta principalmente la ganancia proporcional, esto sirve para incrementar el tiempo de reacción del sistema hasta encontrar el comportamiento deseado según nuestro diseño, pero un incremento excesivo provocará oscilaciones o incluso inestabilidad y un valor de la ganancia excesivamente bajo producirá una reacción demasiada lenta, tanto que sería indeseable.

Lo siguiente sería ajustar el integrador o derivador por partes.

Para el integrador, no se debe poner valores muy altos si tenemos errores muy amplios ya que este haría que los errores en régimen transitorio se amplificasen todavía más haciendo inestable el sistema, pero en cambio para el régimen permanente el integrador es muy importante para conseguir anular casi a cero el error.

En el derivador los valores pueden variar mucho dependiendo del sistema sobre el que se trabaje, pero este es necesario para aplanar posibles sobreimpulsos. Un problema muy común con el derivador es que un valor muy alto puede provocar oscilaciones en altas frecuencias e incluso debido al propio ruido.

3 CAPITULO III: Componentes.

3.1 MPU6050.

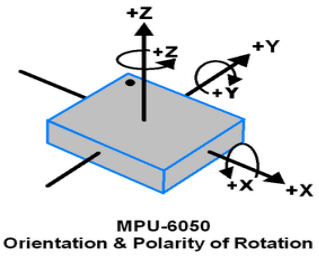


FIG. 11 SENSOR MPU6050

3.1.1 Estudio del sensor

El MPU6050 es un IMU o unidad de medida inercial el cual posee un sensor de seis ejes. Dispone en su chip integrado un giroscopio y un acelerómetro los cuales se complementan entre ellos para obtener así la velocidad, aceleración u orientación sobre el lugar en el que se encuentren, además de poseer un sensor de temperatura.

El giroscopio cuenta con un sensor de 3 ejes con tres ADC 's de 16 bits con el cual nos permite medir la velocidad angular, al igual que el acelerómetro, pero en su lugar medimos la aceleración del dispositivo sobre los 3 ejes, X Y Z.

Para su comunicación con el microcontrolador se realiza mediante la comunicación serie I2C a través de sus puestos SDA y SCL como podemos observar en el esquema de conexión a continuación. [1]

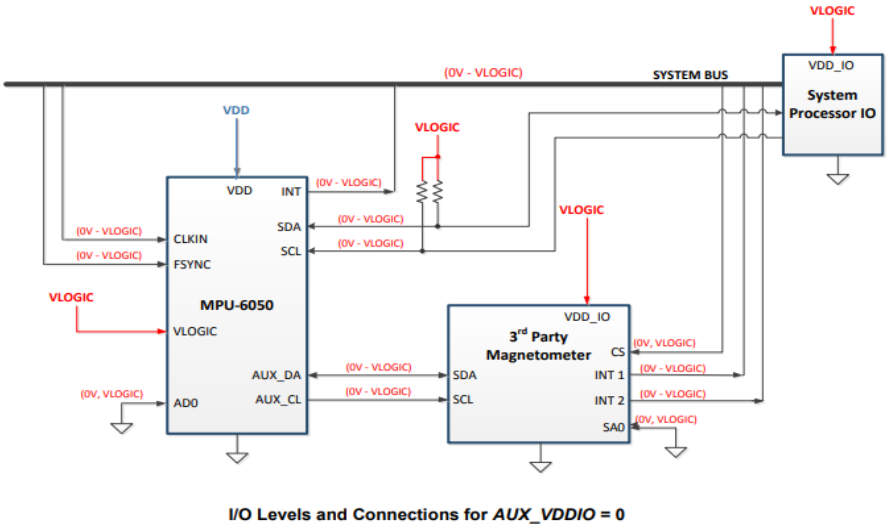


FIG. 12 ESQUEMA CONEXIONADO MPU6050

El MPU6050 es capaz de alimentarse con un voltaje que va desde los 3V hasta los 5V.

Para la correcta implementación del dispositivo, el MPU6050 cuenta con un filtro complementario para la combinación de los cálculos del giroscopio con los del acelerómetro, en el cual se basa de un filtro paso bajo y paso alto fácilmente de implementar con el microcontrolador.

Acelerómetro: el acelerómetro es un dispositivo que mide la aceleración o vibración de una estructura que tiene en su interior. Esta vibración ya sea causada por una vibración o una aceleración provoca el movimiento de una masa el cual choca con un material piezoeléctrico el cual produce una corriente proporcional a la fuerza experimentada.

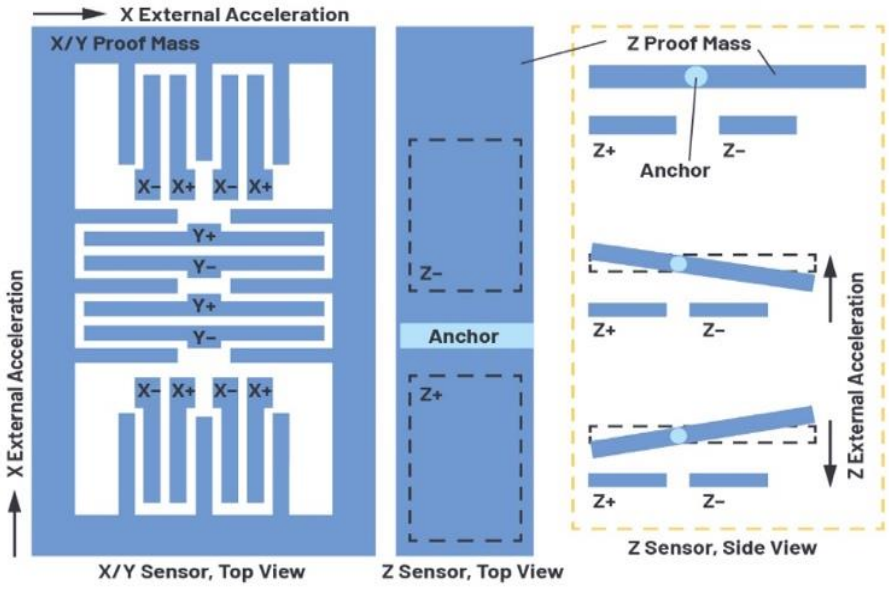


FIG. 13INTERIOR ACELERÓMETRO

Giroscopio: el giroscopio es un dispositivo que mide la velocidad angular del sistema cuando estos están rotando o la variación de orientación del sistema con respecto a un eje de referencia. Los giroscopios empleados en MEMS por sus siglas en inglés (Microeletromechanical Systems) son aquellos en los cuales su sistema mecánico físico se ha miniaturizado hasta escalas micrométricas.

Estos tipos de giroscopios se basan en el principio físico en el que, si un objeto está vibrando, este tiende a continuar vibrando en el mismo plano en el cual gira su apoyo, o también conocida como efecto de vibración de Coriolis.

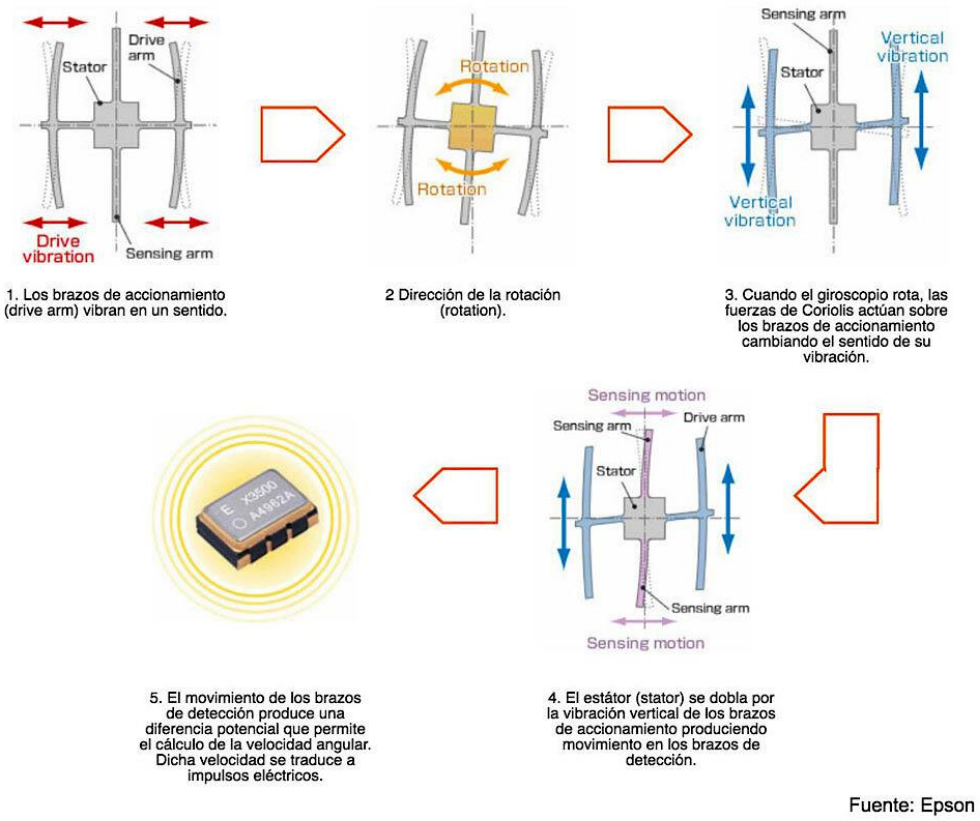


FIG. 14 ESQUEMA GIROSCOPIO

3.1.2 Implementación y cálculos.

Para hallar el ángulo de nuestro robot, es necesario hacer ciertos cálculos pues el sensor MPU6050 nos envía la información al microcontrolador con valores sin procesar.

El MPU6050 se situará en la parte superior del robot con orientación hacia el eje X, es por tanto que el ángulo de giro que nos interesa conocer es sobre el eje Y o también conocido como el pitch del robot.

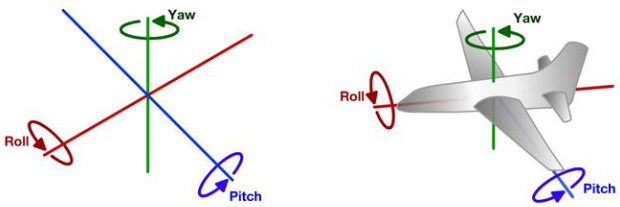


FIG. 15 PITCH ROLL YAW

Para el cálculo del ángulo del eje Y con las aceleraciones recurrimos a la trigonometría obteniendo la siguiente fórmula:

$$AngY_{accel} = \text{atan}\left(\frac{X}{\sqrt{Y^2+Z^2}}\right) \qquad [6]$$

Con esta fórmula podemos conocer el ángulo en el eje Y gracias a las aceleraciones X, Y Z.

Por otro lado, con el giróscopo se mide velocidad angular por tanto para obtener el ángulo necesitamos realizar una integración en el tiempo, tal como se indica en la siguiente fórmula.

$$AngY_{giro} = AngY_{giroinicial} + \int w(t)dt \qquad [7]$$

Donde w(t) es la velocidad angular recogida por el sensor.

Por último, como ya habíamos mencionado anteriormente, implementaremos un filtro para reducir los posibles ruidos que capte el sensor.

Dicho filtro se basa en la siguiente ecuacion:

$$AnguloY_{final} = 0.9 * AngY_{giro} + 0.1 * AngY_{accel} \qquad [8]$$

Los valores de 0.9 para el giroscopio y 0.1 para el acelerómetro son obtenidos de manera experimental en este proyecto. Si se desea que el acelerómetro tome menos importancia en el ángulo final podemos reducir el valor de 0.1 y si por el contrario deseamos que sea más sensible podemos incrementarlo. Un problema de incrementar demasiado la importancia del acelerómetro es que este es muy sensible a las vibraciones del sistema pudiendo provocar un ruido en el ángulo final.

Si observamos las siguientes gráficas observamos como cambiando los valores del filtro complementario, obtenemos un ángulo final más limpio o con mayor rapidez de reacción al cambio.

3.2 Driver DRV8825

FIG. 16 DRIVER DRV8825

Para el control de los motores paso a paso, se empleará dos drivers DRV8825.

Estos dispositivos son necesarios para dar la corriente demandada por el motor y permitir así su correcto funcionamiento. Se basan principalmente en la configuración de un puente en H.

Para el DRV8825 en concreto se emplean MOSFET's de canal N en disposición de puente en H para llegar a entregar con un potenciómetro incorporado en el propio dispositivo hasta 2.5 A al motor.

La diferencia del DRV8825 respecto a otros drivers es que permite la funcionalidad de microstepping, es decir dividir la corriente entregada al motor en incrementos más pequeños para obtener una mayor precisión para el motor paso-paso. Esta técnica nos ofrece una gran ventaja, ya que nos permite posicionar el eje del motor en un mayor rango de opciones, pues un motor stepper con una escala completa tiene una resolución por ejemplo de 1. 8º cada paso que dé, esto quiere decir que para dar una vuelta completa de 360º tendría que dar 200 pasos.

Con el microstepping estos 200 pasos se convertirían incluso en 6400 pasos para dar una vuelta completa, pues el DRV8825 permite un microstepping de 1/32, lo cual nos ofrecería posicionar el eje en un rango mayor de grados.

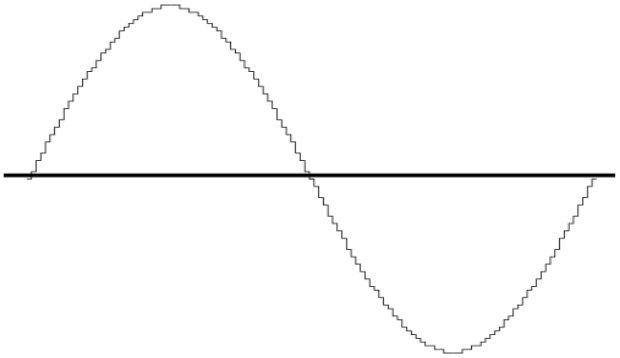


FIG. 17 ONDA CON MICROSTEPPING

El esquema de conexión del DRV8825 es muy simple como podemos observar en la siguiente imagen.

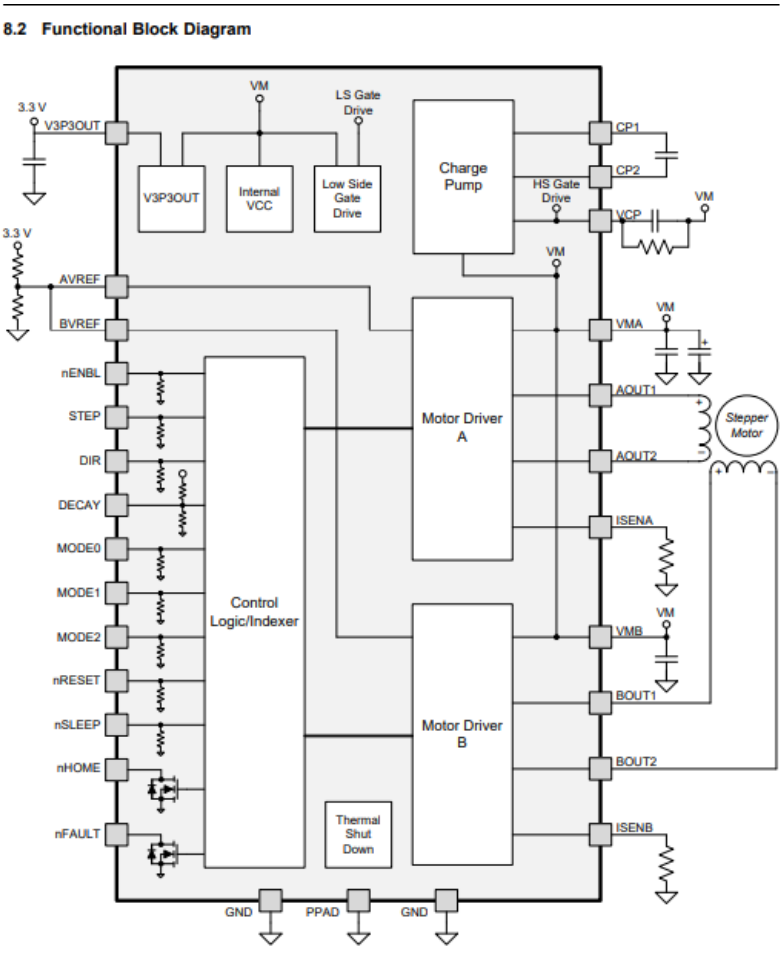


FIG. 18 ESQUEMA ELÉCTRICO DRV8825

En la imagen las patillas AOUT1, AOUT2 se conecta a uno de los dos bobinados que tiene el motor, por tanto, las patillas BOUT1 y BOUT2 se conecta al bobinado restante, alimentando dicho bobinado hasta una corriente máxima de 2.5A

En la patilla VM se conecta la alimentación en tensión el cual alimentaría al motor correspondiente. Esta patilla permite voltajes de 8.2V hasta 42V.

En este proyecto se emplea una batería el cual alimenta el sistema entorno a los 12.4V y 11.1V.

Para el control de este driver, hay que conectar a la patilla STEP y DIR a un puerto digital del microcontrolador para así poder controlar la dirección y los steps a dar del motor.

Si a la patilla DIR lo ponemos a nivel alto este hará que el motor gire en sentido de las agujas del reloj y al ponerlo en bajo girará en sentido contrario.

A la patilla STEP cada vez que le llega un pulso alto hace girar un step al motor o el número de step determinado por el modo al que este configurado.

Las patillas MODE0 MODE1 y MODE2 son las usadas para la funcionalidad del microstepping, la tabla mostrada a continuación explica que pines hay que poner a nivel alto o bajo para obtener una resolución u otra.

M0	M1	M2	Microstep Resolution
Low	Low	Low	Full step
High	Low	Low	Half 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

FIG. 19 TABLA DE MODOS DE RESOLUCIÓN

Para este proyecto he escogido una resolución de 1/4 de paso con el cual necesitaríamos con el motor elegido, 800 pasos para una vuelta completa.

3.3 Arduino nano

FIG. 20 PLACA ARDUINO NANO

En este proyecto se empleará la placa de Arduino Nano el cual está basada en el microcontrolador ATmega328p, el cual nos ofrece una frecuencia de trabajo de 16 MHz.

Se puede alimentar con 5V a través del pin 27, o con 6V hasta 20V por el pin 30, esto nos permite un amplio rango de alimentación con diferentes fuentes. También nos permite alimentarlo a través del puerto mini-USB tipo B por el cual también se comunica con el ordenador.

Posee una flash memory de 32KB el cual 2KB será ocupado por el propio bootloader.

Tiene una memoria EEPROM de 1 KB, que son un tipo de memoria ROM que puede ser programada, borrada y reprogramada eléctricamente.

Dispone también de una memoria SRAM de 2KB, el cual es una memoria volátil el cual perderá todos sus datos una vez que se corte la alimentación.

El Arduino Nano posee hasta 22 pines digitales los cuales pueden ser configurados como entradas o salidas. Además, programando pueden tener funciones diversas como la generación de señales PWM o incluso la de recepción y envío de datos por los puertos serie como el RXD y TXD.

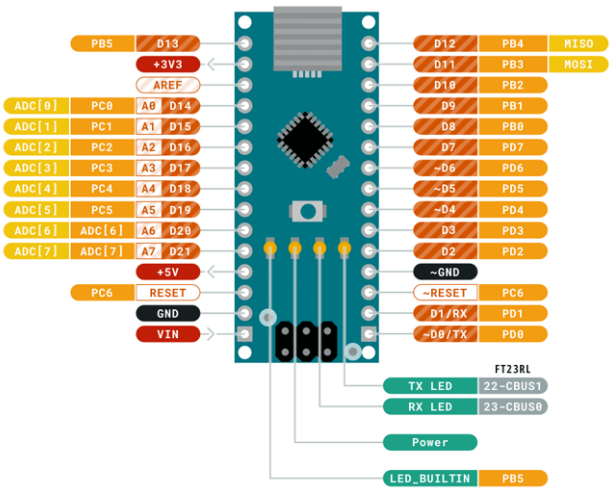


FIG. 21 PATILLAS ARDUINO NANO

En el mercado hay muchos otros microcontroladores con un amplio rango de especificaciones diferentes o muy similares a este.

El uso en específico del Arduino nano es debido que tiene un formato muy compacto con un tamaño de 18x45 mm y un peso muy ligero de tan solo 7 gramos. Además, posee todos los puertos necesarios para este proyecto, aunque con una velocidad de procesamiento alta se puede quedar un poco justa para que todas las funciones se calculen a una velocidad propicia.

Se puede alimentar externamente con un voltaje de 5V lo cual es muy conveniente pues todos los demás dispositivos electrónicos se alimentan también al mismo voltaje.

Además, al ser Arduino, la programación resultaría menos complicada ya que hay un amplio repertorio de foros, tutoriales y bibliotecas que te solucionan o facilitan la resolución del problema.

Es por todas estas cualidades que se eligió finalmente usar el Arduino Nano.

3.4 Motores Stepper o paso a paso

FIG. 22 MOTOR PASO A PASO

Los motores stepper o paso-paso son motores DC que no poseen escobillas, en cambio poseen una serie de bobinas que, alimentado con corriente continua, estas son capaces de generar el campo necesario para el giro del rotor.

3.4.1 Tipos.

Dependiendo del tipo de rotor que poseen existen tres tipos de motores:

- Imanes permanentes

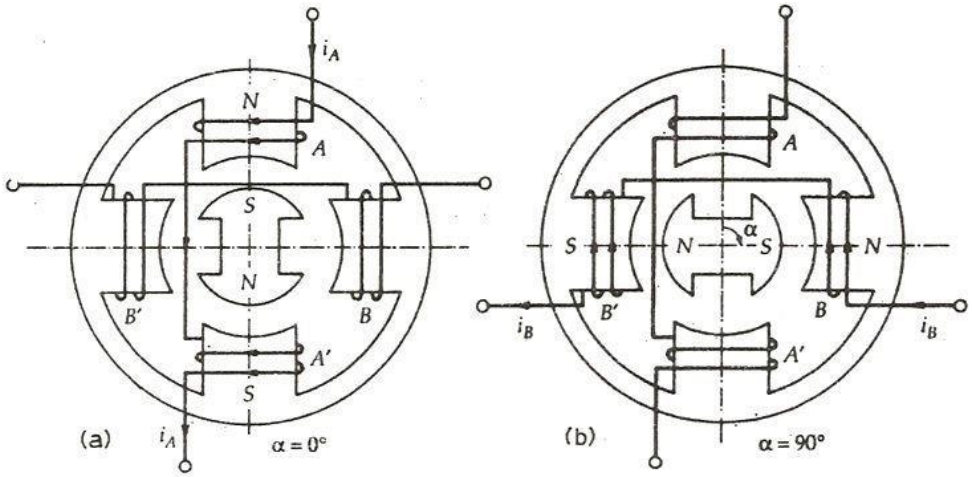


FIG. 23 MOTOR IMANES PERMANENTES

El motor de imanes permanentes se caracteriza, como su nombre indica, por tener un rotor de imanes permanentes. Estas tienen una polaridad que junto con las bobinas del estator los cuales producirán polos opuestos desencadenarán el giro del rotor.

Al ser de imanes permanentes su parada en un punto deseado es mucho mayor, disminuyendo el error de posición debido a la propia inercia, añadiendo que su torque es mucho mayor.

El principal inconveniente de estos motores es que su paso nominal es mayor provocando por tanto el no poder controlar la posición de este de forma más precisa.

- Reluctancia variable

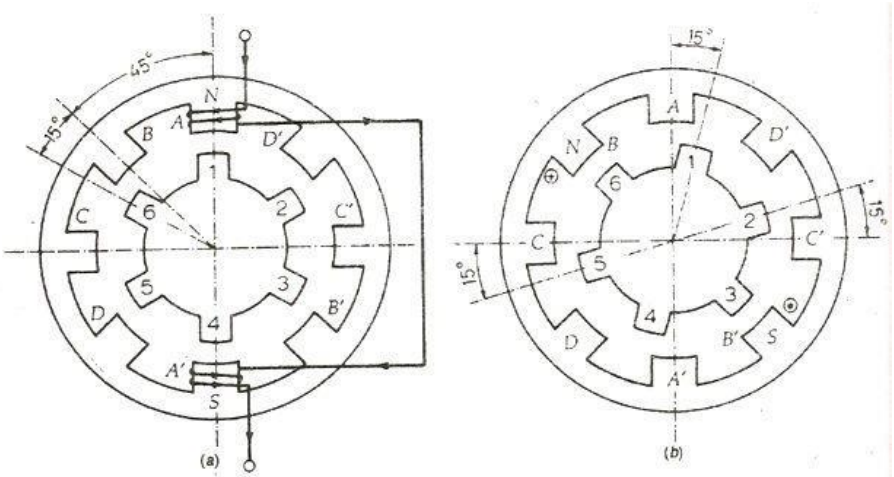


FIG. 24 MOTOR RELUCTANCIA VARIABLE

Los motores de reluctancia variable poseen un rotor de hierro dulce laminado con varios dientes y su estator se forma con polos bobinados de diferente número de dientes al rotor, el cual según el número de dientes tendrá una fase u otra.

El rotor tenderá a alinearse con los polos del estator intentando minimizar la reluctancia entre ellas, esto se da cuando los dientes del rotor encajan con los espacios de los salientes del estator.

El mayor problema de estos motores es que su torque es menor que las de imanes permanentes, pero también se consigue mayor resolución al poder funcionar con pasos más pequeños.

- Híbridos

Los motores híbridos son una mezcla de ambos motores antes descritos.

El rotor en este caso está formado por un disco cilíndrico imantado produciendo así un par de polos norte y sur, además cuenta con una serie de dientes como en el motor de reluctancia variable, solo que estos están altamente imantados por el eje central.

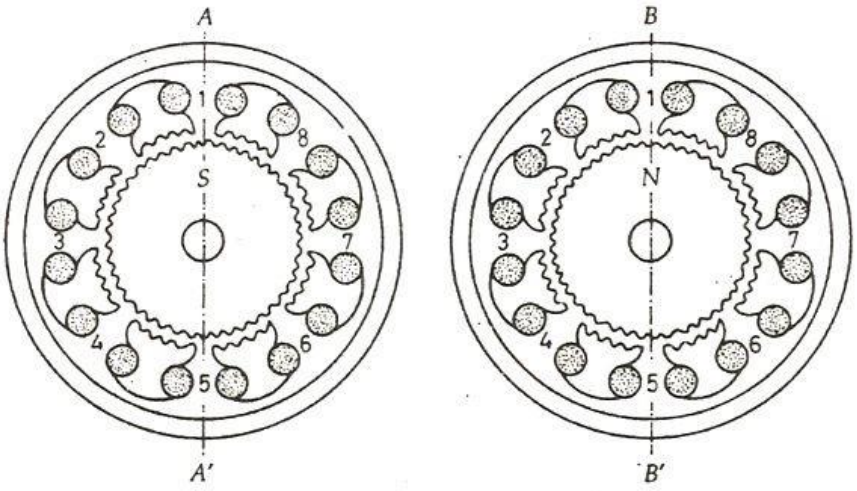


FIG. 25 MOTOR HIBRIDO

En el estator nos encontramos con un número determinado de dientes a los cuales están bobinados produciendo así un numero par de polos.

Para un motor paso a paso de 2 fases normalmente cuentan con 8 polos (como en la imagen de arriba) y en el rotor con 50 dientes, con esta información y la siguiente fórmula podremos determinar cuánto es su ángulo de paso.

$$Angulo\ de\ paso = \frac{360°}{2 * 2\ fases * 50\ dientes} = 1.8°/paso$$

Una cualidad muy importante y distinguible de estos motores, es que estos motores son muy precisos ya que cada pulso que le llegue, este rotará exactamente un ángulo determinado. Un ejemplo es el motor usado para este trabajo, el cual se desplaza 1.8° cada pulso o step que le llegue. Es decir, necesitará de 400 steps para completar una vuelta de 360°. Al poseer este tipo de característica nos permite tener un mayor control sobre la posición de este, incluso implementándolo junto a un buen driver con microstepping podemos obtener hasta resoluciones mayores. La ventaja de que sean tan precisos es que no son necesarios encoders para determinar su posición para la mayoría de las aplicaciones.

3.4.2 Excitación.

El control de estos motores es muy simple, funcionan mediante pulsos de determinado ciclo de trabajo, es decir unas veces están a nivel alto y otras a nivel bajo.

Tenemos que diferenciar entre dos modos de conexionado del motor, el conexionado unipolar y el bipolar. A partir de aquí explicaremos suponiendo que el conexionado del motor es bipolar.

Existen cuatro modos de trabajo:

- Excitación a una sola bobina

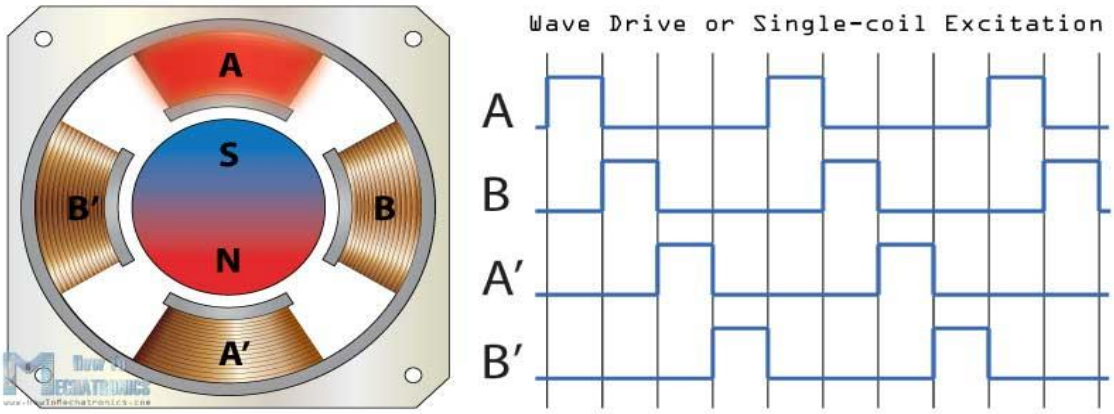


FIG. 26 EXCITACIÓN A UNA SOLA BOBINA

En primer lugar, en un motor de cuatro bobinas, podemos hacer que este de una vuelta completa en tan solo 4 steps. Esta forma de trabajo se llama excitación a una sola bobina. En cada tiempo se excita un solo bobinado produciendo así un giro completo en 4 pasos. En este modo se produce el menor número de conmutaciones posibles por los devanados, reduciendo así el tiempo en los cambios de estado lo cual permite una mayor velocidad de giro, además al estar dos bobinas. El inconveniente de este modo se presenta en los momentos de parada y arranque, pues en las paradas se pueden producir de manera muy brusca y en los arranques es posible que no sea posible vencer el par de resistencia inicial.

- Excitación paso completo.

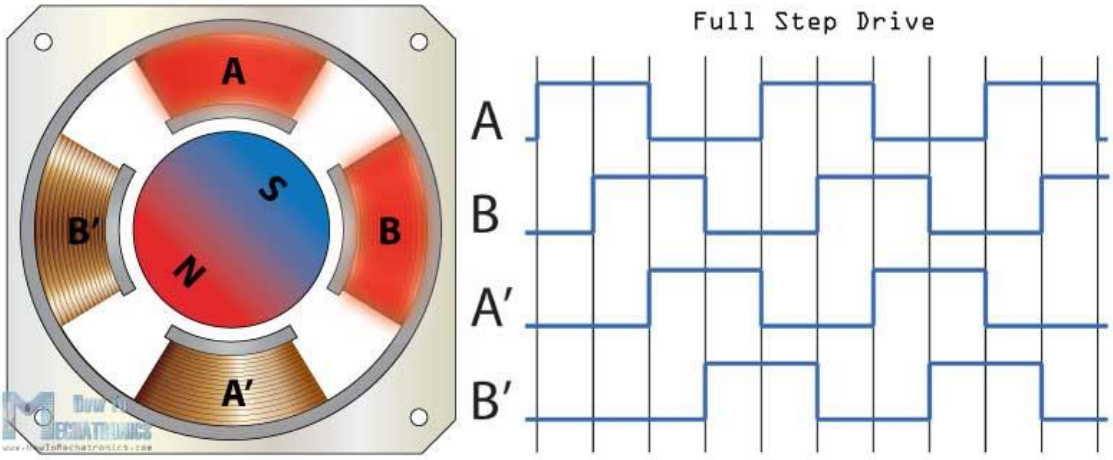


FIG. 27 EXCITACIÓN PASO COMPLETO

Este tipo de excitación es muy similar al anterior con la diferencia en el que en cada tiempo determinado tendremos 2 bobinas activas al mismo tiempo. Esto nos permitirá aumentar el torque del motor ya que estarán dos bobinas activas, pero no obtendremos una resolución mayor, pues seguiremos obteniendo un giro completo en tan solo 4 pasos.

- Excitación medio paso

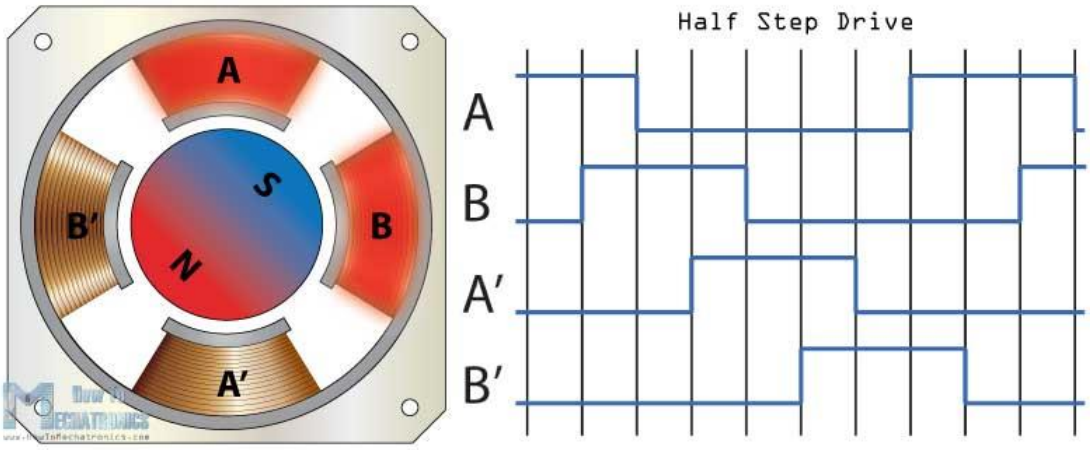


FIG. 28 EXCITACIÓN MEDIO PASO

En este modo lo que se consigue son posiciones intermedias en equilibrio entre pasos consecutivos. Esto nos permitirá una mayor resolución, exactamente el doble que las anteriores. El inconveniente de este modo es que podemos llegar a reducir la velocidad máxima de giro, pues las conmutaciones son mayores.

- Micropasos o microstep

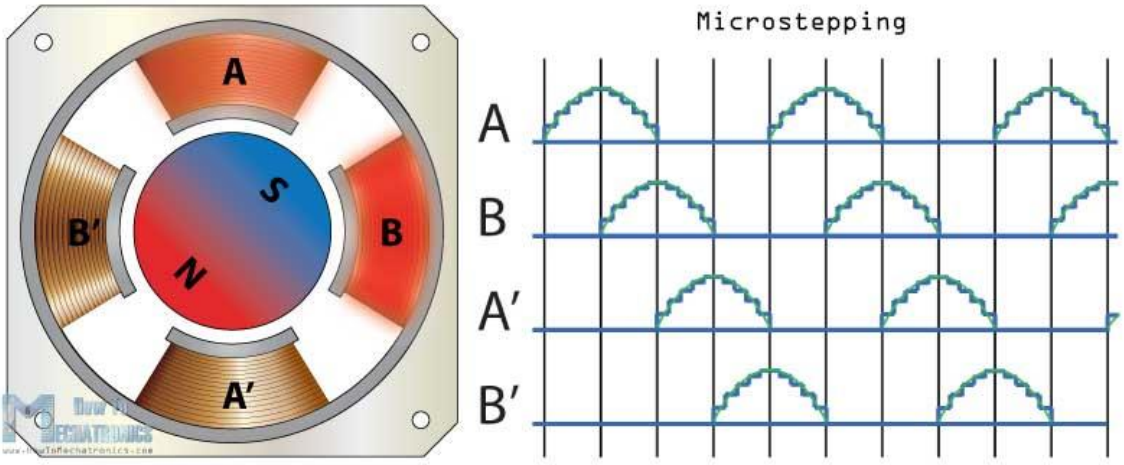


FIG. 29 EXCITACIÓN MICROSTEPPING

En el microstepping o micropasos lo que se logra es dividir en secciones más pequeñas el paso nominal, esto se consigue alimentando el motor con una señal en forma de onda escalonada. De esta forma se consigue una mayor suavidad en el movimiento de los motores ya que la excitación es de forma más escalonada, además al dividir los pasos nominales aumentamos la resolución del motor.

Un problema a tener en cuenta es que esto hace que el motor trabaje a mayor frecuencia de conmutación por tanto se puede llegar a perder velocidad de giro, para evitar este problema deberíamos usar un driver apto para frecuencias altas que sean capaces de conmutar a la velocidad exigida.

3.4.3 Nema 17

Por todas las cualidades explicadas anteriormente se decidió escoger un motor paso a paso en vez de un motor DC común.

En específico usamos un motor paso a paso NEMA 17 17HS4401. [2]

El NEMA 17 es un motor bipolar de 2 fases híbrido que tiene una corriente nominal de 1,5 A por fase, su ángulo de fase es de 1.8° y con un torque de parada de unos 420mN.m

Para un correcto funcionamiento lo conectaremos al driver DRV8825 de una manera muy sencilla el cual nos permitirá limitar la corriente que pase por el motor para evitar fallos o incluso la quema del motor. Aparte de la limitación de corriente, nos facilitará el control con el microcontrolador y la posibilidad de implementar el microstepping ya comentado.

Para el cálculo de la limitación de la corriente por el motor tenemos la siguiente formula que nos da el propio fabricante del driver DRV8825.

$$Imax = \frac{Vref}{5 * RS}$$

Si tenemos en cuenta que el motor la corriente nominal es de 1.5 A por fase y proponemos que trabaje a un 80% de su capacidad esto nos da una corriente de 1.2 A.

La resistencia típica de un DRV8825 es de 0.1Ω por tanto sustituyendo la ecuacion obtenemos lo siguiente:

$$\boldsymbol{Vref = Imax * (5 * RS) = 1.2 * (5 * 0.1) = 0.6V}$$

Por tanto, si en el potenciómetro del driver se ajusta hasta los 0.6V como máx. aseguraremos de no quemar el motor.

3.5 Fuente de alimentación.

Este bloque se encargará de proporcionar la energía necesaria para nuestro robot.

"Una batería eléctrica, acumulador eléctrico o simplemente batería o acumulador, es un dispositivo que consiste en dos o más celdas electroquímicas que pueden convertir la energía química almacenada en corriente eléctrica" [13]

Para este proyecto se ha escogido una batería LiPo recargable que ofrece 11.1V y con una capacidad de 2250mAh, esto lo convierte en una gran opción pues al ser recargable nos permite hacer funcionar al robot incontables veces.

FIG. 30 BATERÍA LIPO

Por su nomenclatura sabemos que se compone de 3 celdas gracias a las letras 3S. Esto significa que cada celda puede llegar hasta los 4.2V cuando están completamente cargadas que conectadas en serie da un total de 12.6V. En la realidad se aprecia que al cargar la batería completamente este llega a los 12.4V alimentando así el circuito del robot a un voltaje mayor, el cual no resulta ningún problema como veremos a continuación gracias al convertidor DC-DC.

La principal ventaja de una batería LiPo es que posee una gran capacidad a un peso relativamente bajo ocupando muy poco espacio, lo cual lo convierte en ideal para aplicaciones de aparatos teledirigidos como pueden ser drones o robots.

Un inconveniente de estas baterías LiPo es que pueden ser inflamables pues en su interior contienen litio el cual es altamente reactiva con el agua, el cual se puede encontrar en la humedad del propio ambiente y por ello que es conveniente tener mucho cuidado al manejar dichas baterías.

El proceso de carga se realiza con un cargador adecuado para baterías LiPo, ya que cargarlas de manera inadecuada podrían producir un mayor deterioro en las celdas o incluso provocar un incendio.

3.6 Convertidor DC-DC reductor.

Un convertidor DC-DC reductor es un dispositivo capaz de reducir la tensión a su salida con un voltaje de entrada mayor.

Su funcionamiento en resumen se basa en pasar la corriente por un dispositivo interruptor, por lo general un transistor, el cual hace que la señal se convierta en AC y posteriormente pasarlo por un filtro compuesto por inductores y condensadores que lo vuelve a convertir en DC a la tensión deseada.

Los hay de varios tipos como pueden ser de tipo Buck, Boost, Cuk, SEPIC etc.

Un convertidor DC-DC puede trabajar en dos modos, el modo continuo o CCM o en discontinuo DCM, se diferencian principalmente en el que si la corriente que pasa por la bobina llega a ser nula se dice entonces que esta trabaja en modo discontinuo. Lo cual cambiaría por completo los cálculos.

El convertidor empleado en este proyecto es un convertidor reductor regulable de la marca AZDelivery. Esto quiere decir que el voltaje a su entrada puede ir en este caso de 4-40V y a la salida podemos obtener un voltaje en el rango de 3.3V a 24V.

Esto nos ofrece una gran ventaja pues podemos coger el voltaje de 12.4V de la batería y convertirla a los 5V necesarios para la alimentación de los demás dispositivos como puede ser el microcontrolador, los drivers o el IMU. Cuenta con un regulador en el cual se nos permite seleccionar el voltaje deseado y visualizarlo a través de un display de 8 segmentos que tiene incorporado el propio dispositivo.

FIG. 31 CONVERTIDO REDUCTOR DC-DC

3.7 Estructura.

La estructura del robot es bastante simple, está formado por 3 planchas de metacrilato transparentes de 20x10cm a los cuales se les ha perforado en sus respectivas esquinas unos agujeros utilizando un taladro de 6 cm de diámetro. A estos agujeros se les ha introducido unas varillas con roscas de M6 y de 200mm de longitud. A cada plancha de metacrilato se les ha sujetado a las varillas con tuercas de M6 por los dos lados apretadas correctamente para evitar el desplazamiento de estas. Hay 3 niveles de altura en el robot y la disposición de dichas alturas de las planchas, tomando como referencia cero la altura de los motores, son de 13.5 cm y 16 cm de altura.

En el nivel inferior se sitúa las sujeciones de los motores y la batería, los cuales son los que más masa tienen bajando así el centro de gravedad.

En el nivel secundario se encuentran el convertidor DC-DC y el microcontrolador y en el nivel superior se encuentra la placa protoboard, al que van conectadas los drivers para los motores y el MPU6050.

La sujeción de los motores se ha realizado a través de unos perfiles metálicos ideales para los NEMA-17 de 51x51mm y unos tornillos de M3. La fijación de los perfiles a la plancha de metacrilato se realiza con unos tornillos de M1 y sus respectivas tuercas.

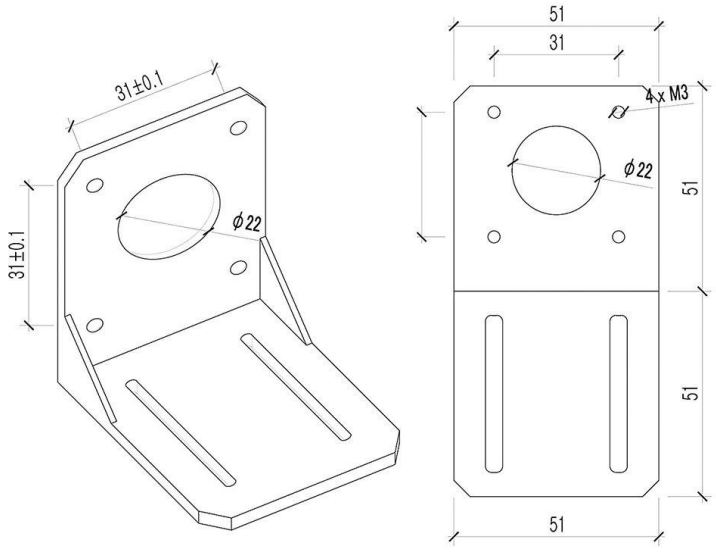


FIG. 32 PERFIL SUJECIÓN MOTOR

Todos los componentes salvo los motores se han sujetado con cinta de doble cara al ser la opción más sencilla y práctica.

La estructura del robot completo quedaría como la imagen mostrada a continuación usando AUTOCAD para su modelado.

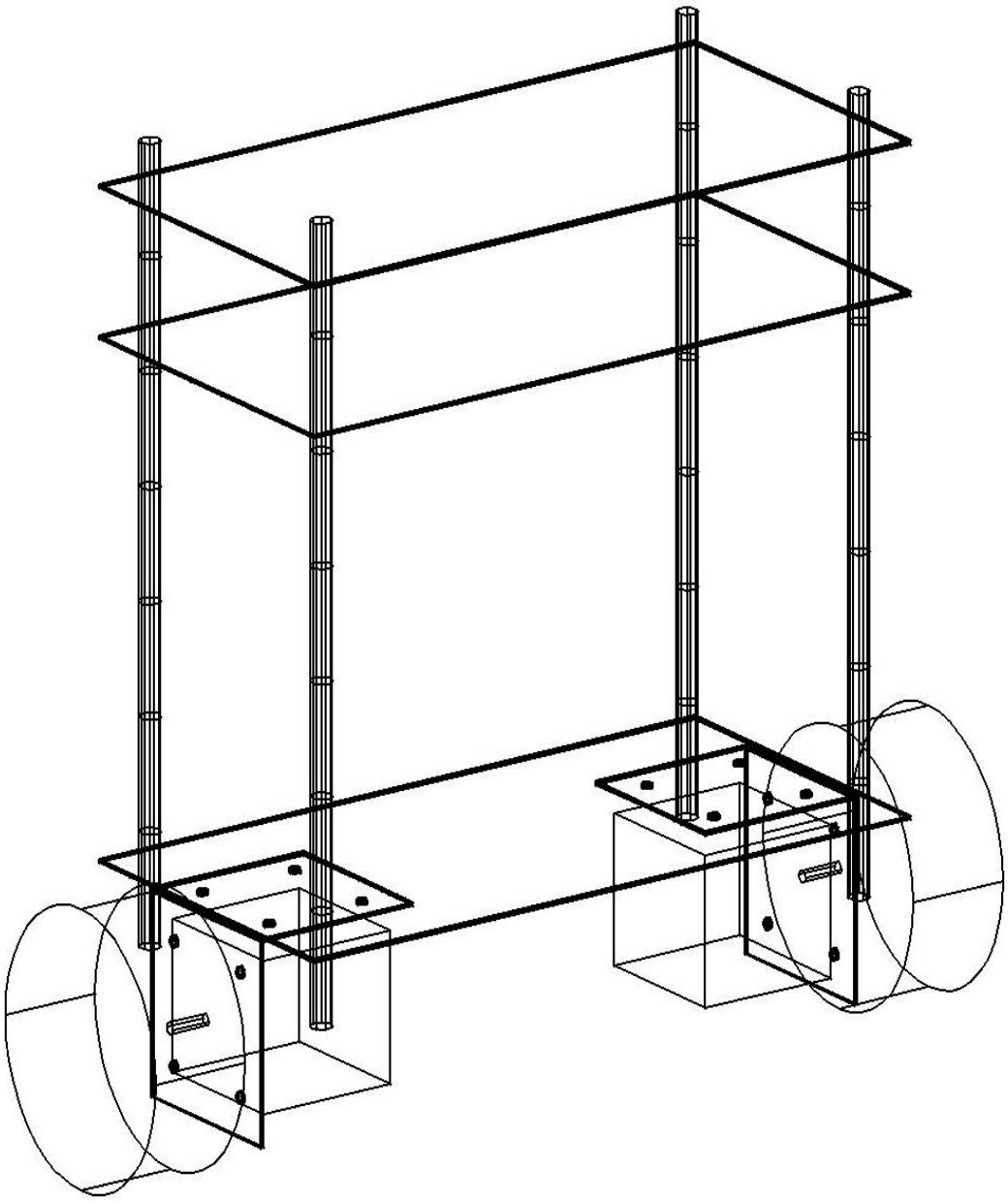


FIG. 33 MODELO 3D ROBOT

Y finalmente el robot final una vez montado con todos sus componentes y las conexiones correspondientes quedaría como en la imagen mostrada a continuación.

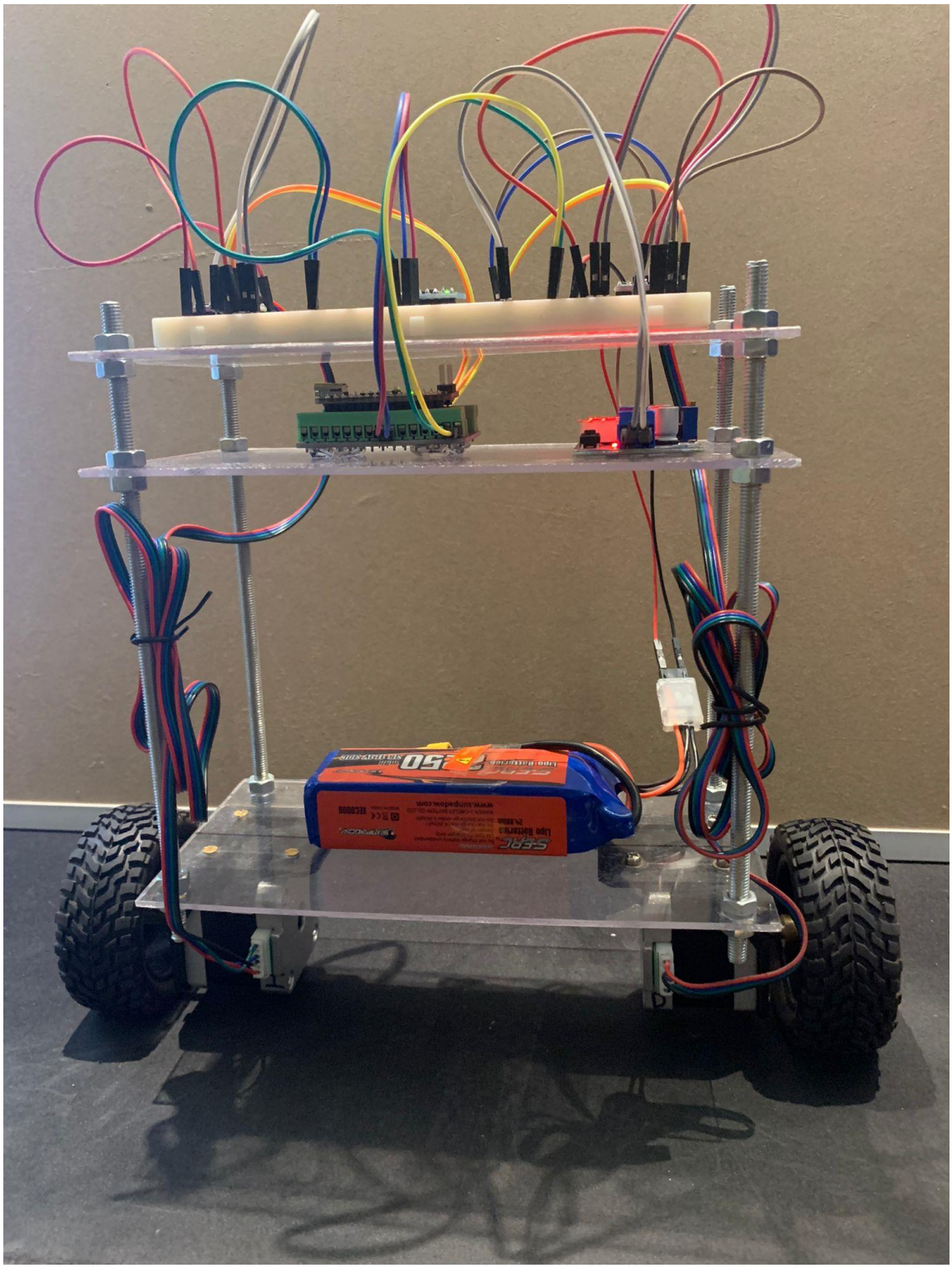


FIG. 34 ROBOT FINAL

4 CAPITULO IV: Conexionado.

El conexionado se realizará a través de cables jumper macho de diferentes colores para facilitar la identificación de cada puerto.

Siguiendo el esquema de conexionado realizado con el programa EAGLE de Autodesk que se muestra a continuación, se procede a realizar las conexiones pertinentes.

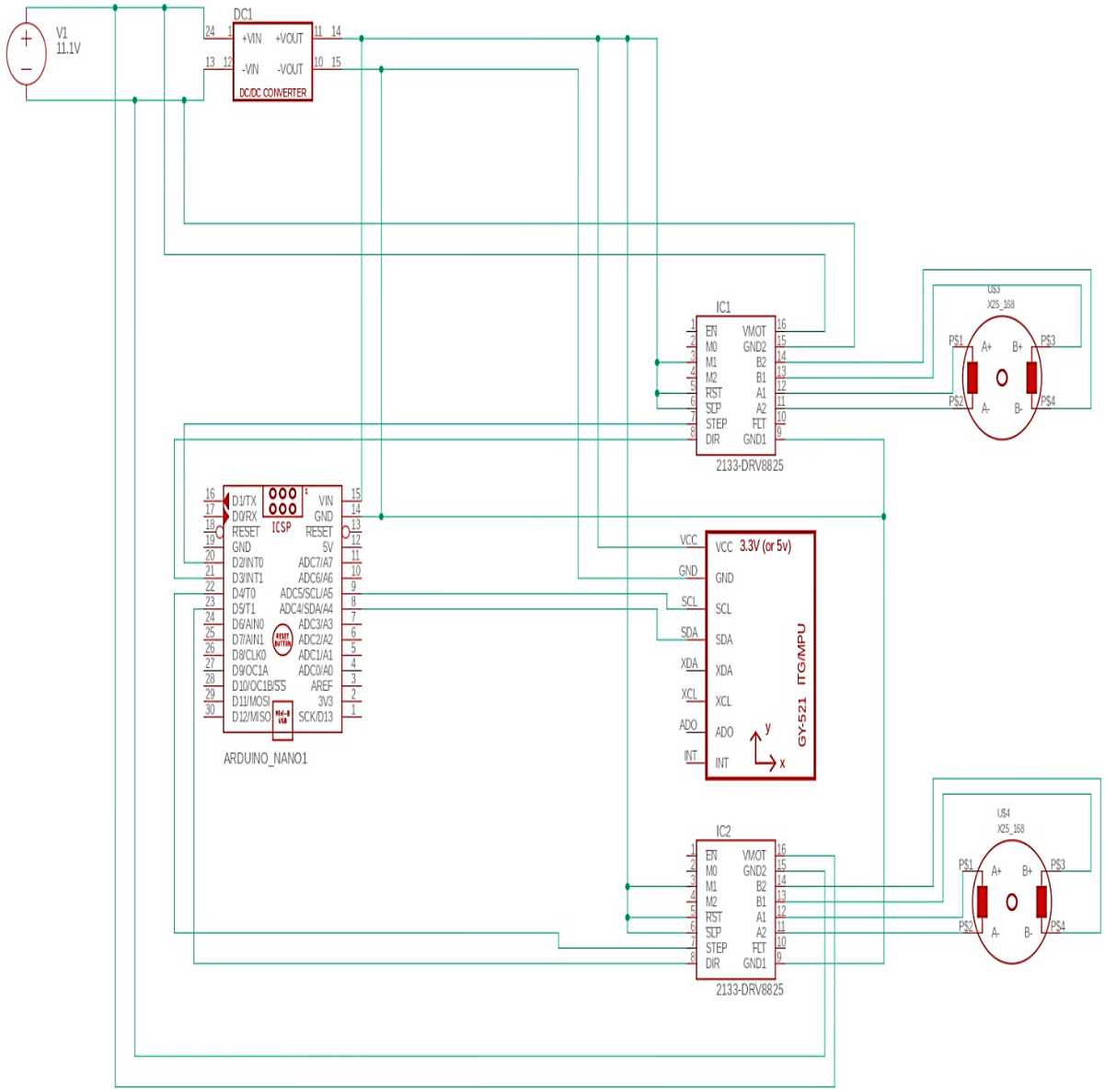


FIG. 35 CONEXIONADO DEL SISTEMA

4.1 Alimentación del circuito.

En la placa protoboard al que se conectará principalmente los componentes del sensor MPU6050 y los dos drivers de los motores, se dispone de dos ranuras a los laterales con el signo + / – ideales para alimentar el circuito.

Usaremos ambos lados, en el cual como se aprecia en la imagen por el lado marcado con rotulador negro conectaremos a 11.1V y el otro lado de la placa alimentaremos a 5V usando el convertidor reductor.

Toda la regleta de 11.1V será alimentado por la batería LiPo, el cual se ha usado cables jumper de color rojo y negro para poder diferenciar correctamente la polaridad y evitar incidencias.

Esto nos facilitará y dejará más limpio el conexionado y sin cables entrecruzándose.

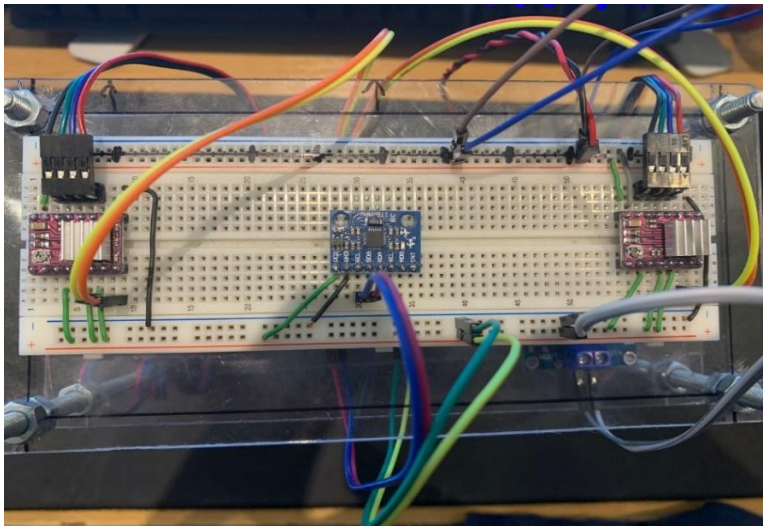


FIG. 36 PLACA PROTOBOARD

A la regleta de 11.1V se conectará el Arduino Nano la cual se alimentará por el pin 30. Junto al Arduino alimentaremos también a 11.1V el convertidor reductor DC-DC necesario para obtener los 5V del circuito.

También se alimentará los motores a 11.1 V conectando los pines del VMOT y GND tal como se muestra en la imagen de abajo.

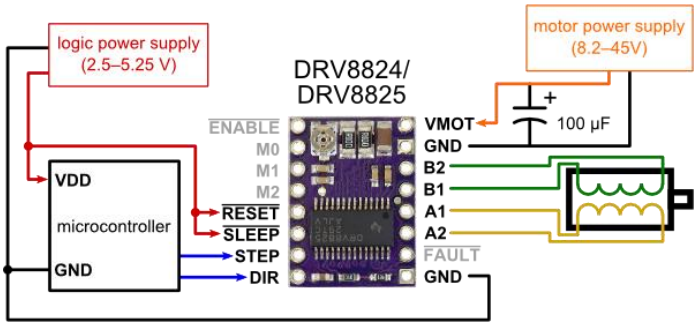


FIG. 37 ESQUEMA DDRV8825

El resto de los componentes como el sensor MPU6050 y el driver DRV8825 serán alimentados a 5V por sus respectivos pines Vcc y GND.

4.2 Circuito de control y comunicación.

El control del driver será llevado a cabo por el microcontrolador a través de las patillas STEP Y DIR.

Para el driver del motor derecho se conectará la patilla STEP al pin D4 y la patilla DIR al pin D5 del Arduino. Y para el driver del motor izquierdo se conectará el STEP al pin D2 y DIR al pin D3.

Para el MPU6050 realizaremos la comunicación I2C a través de sus patillas SDA y SCL que conectaremos al Arduino por los pines A4 y A5 respectivamente.

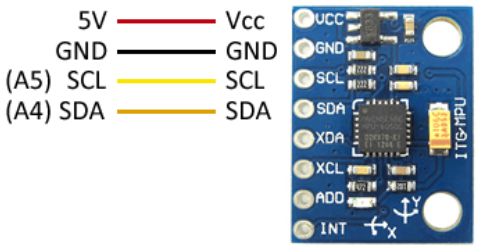


FIG. 38 CONEXIÓN MPU6050

4.3 Precauciones

Para el conexionado de los diferentes elementos hay que tener mucho cuidado a la hora de conectar las diferentes patillas pues una equivocación podría provocar un cortocircuito en el componente averiándolo y dejando totalmente inservible el dispositivo.

Antes de conectar la batería siempre será recomendable asegurarnos de que todo el cableado este correctamente fijo y revisar que ninguno esté conectado erróneamente para evitar fallos catastróficos como sería un cortocircuito en el circuito de la batería y provocar un incendio.

5 CAPITULO V: Software

En este apartado se explicará el código implementado en el robot para su funcionamiento.

Como ya se mencionó anteriormente se utilizará el Arduino Nano para realizar este proyecto y es por ello por lo que necesitaremos usar el programa de Arduino IDE.

Una cualidad principal para la programación en Arduino es que, al ser un software gratuito y muy popular en este sector, se puede recurrir a numerosos foros y bibliografías que puedan solucionar los posibles errores que surjan a la hora de programar. Además, cuenta con un gran número de librerías que permite su uso en el código de manera muy sencilla.

5.1 Entorno de desarrollo

En la imagen de abajo se muestra un ejemplo de cómo se ve el entorno de programación de Arduino. Su uso principal es muy sencillo, como se irá explicando a continuación.

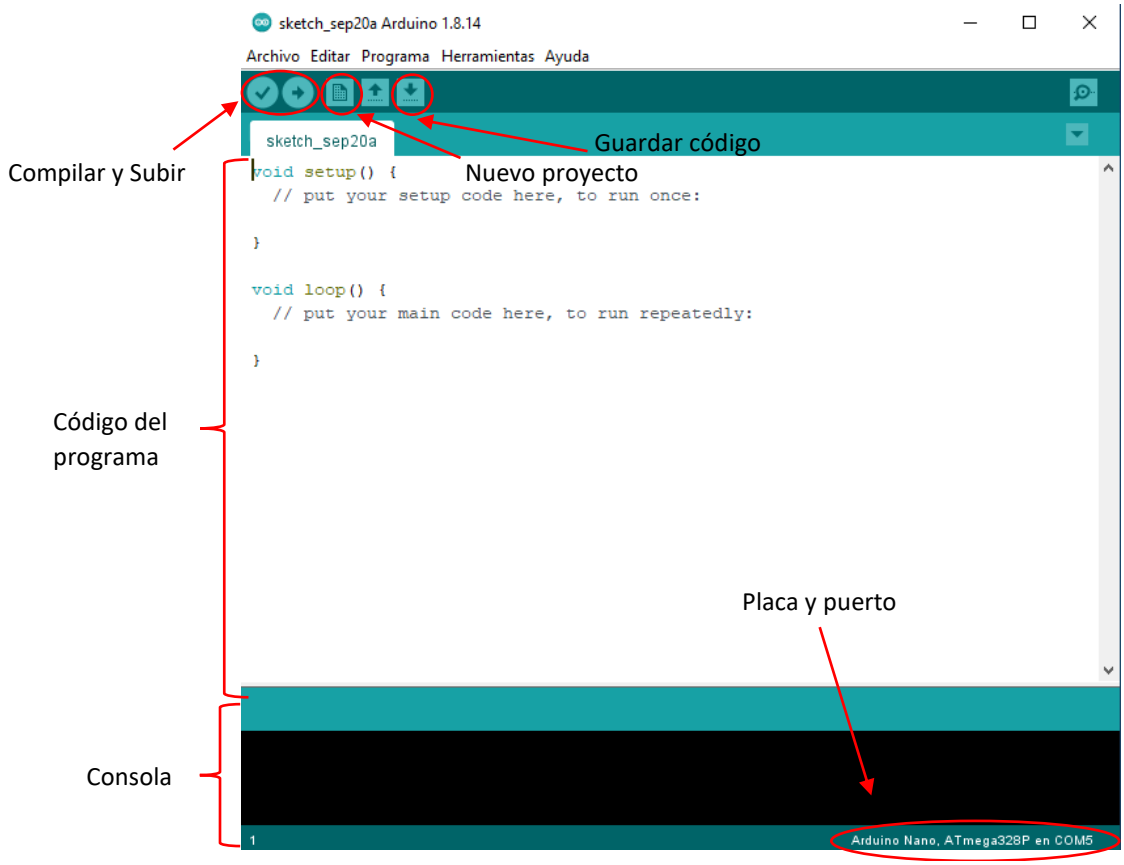


FIG. 39 ARDUINO IDE

Para empezar, hay que asegurar que la placa esté correctamente conectado al PC y comprobar que puerto utiliza. Para ello hay que entrar en el administrador de dispositivos y en la sección de puertos (COM y LPT) verificar que puerto se emplea.

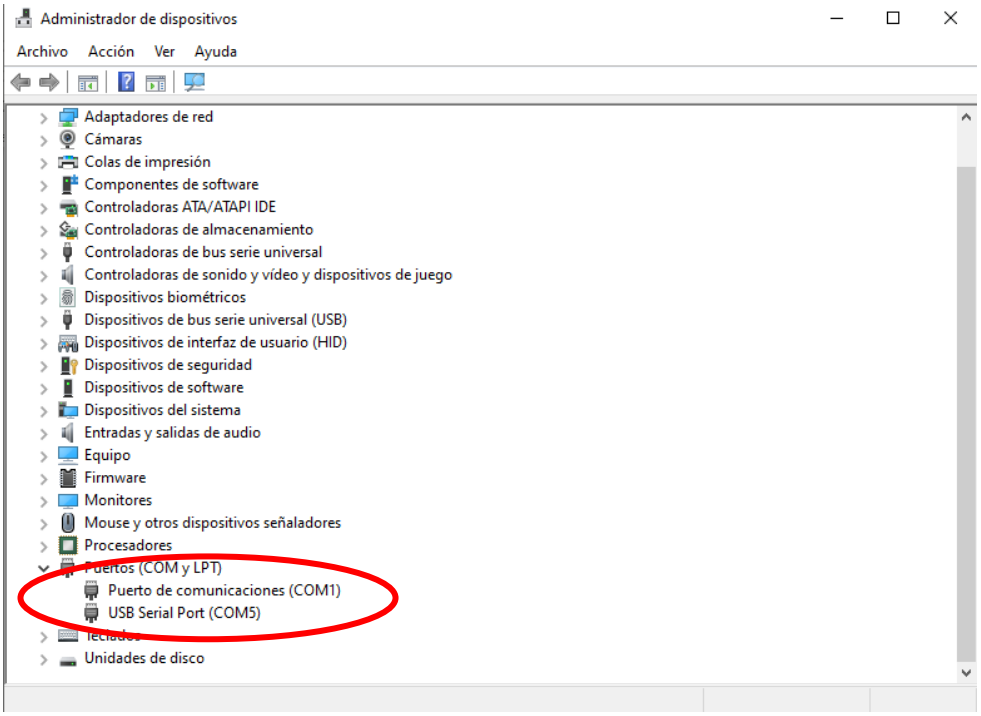


FIG. 40 ADMINISTRADOR DE DISPOSITIVOS

Una vez comprobado el puerto de la placa, el código del programa se compila comprobando que no haya ningún error en las funciones.

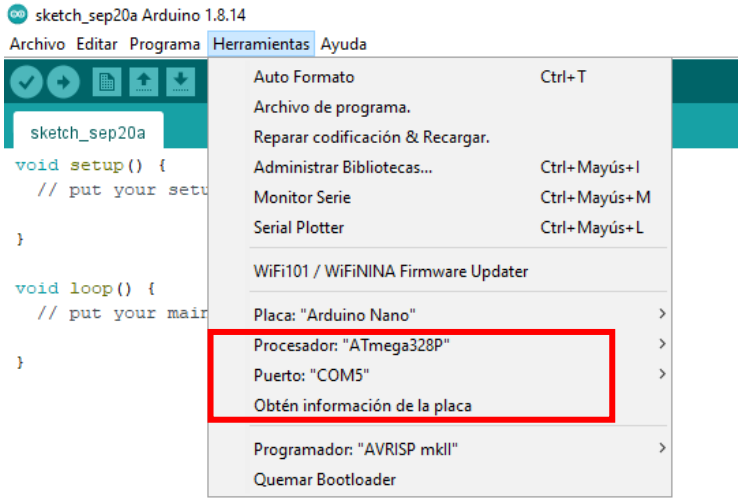


FIG. 41 PESTAÑA HERRAMIENTAS

Antes de subir el código hay que comprobar en la pestaña de herramientas que la placa, el procesador empleado y el puerto coincidan los datos anteriormente descritos. Finalmente, una vez se comprueba todo se sube el programa compilado a la placa

5.2 Código ROBOT

A continuación, se explicará paso a paso el funcionamiento del código implementado para este proyecto.

Empezando por la configuración inicial o la función void setup()

5.2.1 Función Setup()

Dentro de esta función se inicia la comunicación I2C el cual se configurará a 400 KHz escribiendo en el registro TWBR y se inicia tambien la comunicación serial port a 9600 baudios.

Para el control de los steppers se ha usado los drivers DDRV 8825 que se ha comentado anteriormente que van conectados a los puertos D2, D3, D4 y D5. Dichos puertos se configuran como pines de salida.

```
46 void setup() {
47
48   Serial.begin(9600);
49   Wire.begin();
50   TWBR=12;
51
52   //__________configuracion como salida de los pines de los Driver DDRV8825____
53   pinMode(dirPin1, OUTPUT);
54   pinMode(stepPin1, OUTPUT);
55   pinMode(dirPin2, OUTPUT);
56   pinMode(stepPin2, OUTPUT);
```

FIG. 42 CÓDIGO 1

El control de los motores se realizará por medio de una señal PWM que se configurará con el timer2 de 8 bits del Arduino.

Dicho Timer interrumpirá cada 65us generando un pulso en los motores, para que funcione de esta manera se debe configurar los siguientes registros.

```
58 //____configuracion del timer2
59
60   TCCR2A = 0;
61   TCCR2B = 0;
62   TIMSK2 |= (1 << OCIE2A);
63   TCCR2B |= (1 << CS21);
64   OCR2A = 129;
65   TCCR2A |= (1 << WGM21);
```

FIG. 43 CÓDIGO 2

Primero hay que asegurar que los registros del timer se encuentren a cero, igualando a 0 los registros TCCR2A y TCCR2B limpiando así dichos registros.

Activamos el modo "match compare" del timer escribiendo en el registro TIMSK2.

Con el registro TCCR2B se selecciona el preescaler del temporizador que en el caso de este proyecto se ha puesto a 8, como se puede observar en la tabla al poner un 1 en el CS21.

Table 17-9. Clock Select Bit Description

CS22	CS21	CS20	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	clk_{T2S}/(no prescaling)
0	1	0	clk_{T2S}/8 (from prescaler)
0	1	1	clk_{T2S}/32 (from prescaler)
1	0	0	clk_{T2S}/64 (from prescaler)
1	0	1	clk_{T2S}/128 (from prescaler)
1	1	0	clk_{T2S}/256 (from prescaler)
1	1	1	clk_{T2S}/1024 (from prescaler)

FIG. 44 TABLA PRESCALER

El siguiente es el valor al que llegará el contador e interrumpirá. Este valor se calcula con la frecuencia del reloj del procesador, el prescaler y cada cuanto se desea interrumpir.

$$ticks = \frac{65us}{\frac{1s}{(\frac{16Mhz}{8})}} - 1 = 129\ ticks$$

Donde 16 MHz es la velocidad del procesador, 8 el prescaler y 65us el tiempo entre cada interrupción. Por tanto, en el registro OCR2A se pone a 129.

Una vez se produce la interrupción se debe resetear de nuevo el timer para que funcione continua e indefinidamente poniendo el registro TCCR2A un 1 en el bit WGM21.

A continuación, se configura el MPU6050, para ello se hace uso de la librería <MPU6050.h> ya incluido en el desarrollador de Arduino.

```
68   //_____________________configuracion del MPU6050_____
69   mpu.initialize();
70   Wire.beginTransmission(dir_MPU);
71   Wire.write(0x6B); // PWR_MGMT_1 registro;
72   Wire.write(0x00);
73   Wire.endTransmission();
74   //Configurar el giro a +/-250 grados por segundo
75   Wire.beginTransmission(dir_MPU);
76   Wire.write(0x1B);
77   Wire.write(0x00);
78   Wire.endTransmission();
79   //Configurar el acelecometro a +/- 2g
80   Wire.beginTransmission(dir_MPU);
81   Wire.write(0x1C);
82   Wire.write(0x10);
83   Wire.endTransmission();
84   //Activamos un filtro del propio sensor
85   Wire.beginTransmission(dir_MPU);
86   Wire.write(0x1A);
87   Wire.write(0x03);
88   Wire.endTransmission();
89
90     mpu.setXAccelOffset(-971);
91     mpu.setYAccelOffset(679);
92     mpu.setZAccelOffset(933);
93     mpu.setXGyroOffset(7);
94     mpu.setYGyroOffset(47);
95     mpu.setZGyroOffset(9);
96 }
```

FIG. 45 CÓDIGO 3

Primero iniciamos la función mpu.initialize() el cual inicia la comunicación con el sensor MPU6050.

A continuación, con la función Wire.write() se escribe la dirección deseada que se quiera modificar, en este caso empezando por la dirección 0x6B.

- Registro 0x6B

Dicha dirección sirve para configurar el modo de funcionamiento y que reloj ha de usar, si escribimos un 0 reseteamos todo el registro usando además el reloj interno de 8MHz.

4.28 Register 107 – Power Management 1
PWR_MGMT_1

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
6B	107	DEVICE_RESET	SLEEP	CYCLE	-	TEMP_DIS	CLKSEL[2:0]		

CLKSEL	Clock Source
0	Internal 8MHz oscillator
1	PLL with X axis gyroscope reference
2	PLL with Y axis gyroscope reference
3	PLL with Z axis gyroscope reference
4	PLL with external 32.768kHz reference
5	PLL with external 19.2MHz reference
6	Reserved
7	Stops the clock and keeps the timing generator in reset

FIG. 46 REGISTRO 0X6B

- Registro 0x1B

Para la dirección 0x1B se configura el giroscopio y que escala se desea tener, en este caso se usará una resolución de ± 250 °/s

4.4 Register 27 – Gyroscope Configuration
GYRO_CONFIG

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1B	27	XG_ST	YG_ST	ZG_ST	FS_SEL[1:0]		-	-	-

FS_SEL	Full Scale Range
0	± 250 °/s
1	± 500 °/s
2	± 1000 °/s
3	± 2000 °/s

FIG. 47 REGISTRO 0X1B

- Registro 0x1C

Con el registro 0x1C nos permite el acceso a la configuración del acelerómetro, el cual se ajustará con una sensibilidad de ±4g donde 1g es igual a la aceleración de la gravedad o 9.8 m/s^2.

4.5 Register 28 – Accelerometer Configuration
ACCEL_CONFIG

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1C	28	XA_ST	YA_ST	ZA_ST	AFS_SEL[1:0]		-		

AFS_SEL	Full Scale Range
0	± 2*g*
1	± 4*g*
2	± 8*g*
3	± 16*g*

FIG. 48 REGISTRO 0X1C

- Registro 0x1A

Por último, entrando en el registro 0x1A se puede configurar el filtro paso digital del propio sensor. Dicho filtro de paso bajo será configurado a 43Hz aprox.

El uso de este filtro provocará un retardo de 4.9 ms, pero será útil para limpiar posibles ruidos del sistema.

DLPF_CFG	Accelerometer (F_s = 1kHz)		Gyroscope		
	Bandwidth (Hz)	Delay (ms)	Bandwidth (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	RESERVED		RESERVED		8

FIG. 49 DLPF

- Errores de offset

Como cualquier otro sensor el MPU6050 posee ciertos errores de offset que han de ser corregidos evitando un error aun mayor a la salida del ángulo que deseamos medir.

Es por esto por lo que con la función mpu.set____Offset() es posible eliminar dicho valor. Para ello primero se debe calcular el offset del sensor por medio de un sketch externo.

Una vez finalizado dicho sketch, este imprime por pantalla los valores de offset que se deben anotar para finalmente ser corregidos.

En este caso el MPU6050 empleado tenían los siguiente errores de offset:

Xacelerometro	-975	XGiroscopio	9
Yacelerometro	635	YGiroscopio	45
Zacelerometro	938	ZGiroscopio	8

5.2.2 Función loop()

La función loop se ejecutará en el procesador del microcontrolador indefinidamente o también conocido como bucle del inglés "loop".

Aquí se realizará las acciones necesarias para el funcionamiento correcto y continuado del robot.

- **Bloque adquisición de datos**

En este bloque se obtendrán los datos recibidos por el MPU6050 y el uso de dichos datos para el cálculo del ángulo deseado del pitch.

```
98  void loop() {
99
100   mpu.getAcceleration(&ax, &ay, &az);
101   mpu.getRotation(&gx, &gy, &gz);
102
103   GyX= gx/131;
104   GyY= gy/131;
105   GyZ= gz/131;
106
107   float acc_ang_y = atan((ax/8192.0) / sqrt(pow((ay/8192.0), 2)+ pow((az/8192.0), 2)))*RAD_TO_DEG;
108
109   Tiempo_pasado = Tiempo_actual;
110   Tiempo_actual = millis();
111   Tiempo_transcurrido= (millis() - Tiempo_pasado)/ 1000; //para conseguirlo en segundos.
112
113   gyro_ang_y = gyro_ang_y + GyY*Tiempo_transcurrido; //los valores estan en grados/s por tanto mul
114
115   pitch = 0.98*gyro_ang_y + 0.02*acc_ang_y;
116
117 //  Serial.print(acc_ang_y);
118 //  Serial.print(" ");
119 //  Serial.print(gyro_ang_y);
120 //  Serial.print(" ");
121 //  Serial.print("\tpitch: ");
122 //  Serial.println(pitch);
123 //  Serial.print(" ");
```

FIG. 50 CÓDIGO 4

Empezando por la adquisición de datos que envía el sensor de movimiento se recurre a las funciones :

mpu.getAcceleration(&ax, &ay, &az);

mpu.getRotation(&gx, &gy, &gz);

Estas funciones permiten obtener los valores del acelerómetro y giroscopio y guardarlas en sus respectivas variables.

Los valores de las variables obtenidas deben de ser reescaladas, ya que el sensor nos da valores brutos las cuales tienen que ser recalculadas para su compresión. Según el datasheet del fabricante para los valores del giroscopio, teniendo una escala de ±250°/s los valores obtenidos tienen que ser divididos entre 131, como podemos observar en la siguiente imagen.

FS_SEL	Full Scale Range	LSB Sensitivity
0	± 250 °/s	131 LSB/°/s
1	± 500 °/s	65.5 LSB/°/s
2	± 1000 °/s	32.8 LSB/°/s
3	± 2000 °/s	16.4 LSB/°/s

FIG. 51 TABLA GIROSCOPIO SENSIBILIDAD

Al igual que en el giroscopio en el acelerómetro también se debe dividir los valores recibidos. En este caso entre 8192 para una resolución de ±4g como se indica en la siguiente tabla.

AFS_SEL	Full Scale Range	LSB Sensitivity
0	±2*g*	16384 LSB/*g*
1	±4*g*	8192 LSB/*g*
2	±8*g*	4096 LSB/*g*
3	±16*g*	2048 LSB/*g*

FIG. 52 TABLA ACELERÓMETRO SENSIBILIDAD

Una vez obtenidos los valores del acelerómetro y giroscopio se procede al cálculo del ángulo. Como ya se comentó anteriormente en el apartado del MPU6050 se utilizará la trigonometría para el cálculo del ángulo por medio de las aceleraciones con el siguiente algoritmo.

```
107  float acc_ang_y = atan((ax/8192.0) / sqrt(pow((ay/8192.0), 2)+ pow((az/8192.0), 2)))*RAD_TO_DEG;
```

FIG. 53 CALCULO ANGULO-ACEL

Para el cálculo del ángulo por medio del giroscopio es necesario medir el tiempo que pasa en cada muestra y realizar la integración. Dando como resultado el siguiente algoritmo.

```
109  Tiempo_pasado = Tiempo_actual;
110  Tiempo_actual = millis();
111  Tiempo_transcurrido= (millis() - Tiempo_pasado)/ 1000; ,
112
113  gyro_ang_y = gyro_ang_y + GyY*Tiempo_transcurrido; //los
114
```

FIG. 54 CALCULO ANGULO-GYRO

Finalmente se implementa el filtro complementario para juntar los valores medidos por el acelerómetro y giroscopio con el fin de obtener una señal final bien definida y con mayor resistencia al ruido.

```
115   pitch = 0.98*gyro_ang_y + 0.02*acc_ang_y;
```

FIG. 55 FILTRO COMPLEMENTARIO

- **Bloque Controlador PID**

En este bloque se realiza el código necesario para la implementación del algoritmo del controlador PID usado para este proyecto.

Este código se encarga de combinar las tres partes del control en una sola salida que será empleada más adelante para la salida a los motores.

```
125   ////////////////////////calculo para el PID/////////
126
127   error = pitch - angulo_deseado
128   pid_p = kp*error;
129   pid_i= pid_i + (ki*error);
130   if (pid_i>400)pid_i=400;
131   else if(pid_i<-400)pid_i=-400;
132
133   pid_d= kd*((error-error_ant));
134
135   PID_output= pid_p + pid_i + pid_d;
136
137   if (PID_output > 400){
138     PID_output=400;
139   }
140   if(PID_output< -400){
141     PID_output=-400;
142   }
143     error_ant = error;
144
145   if(PID_output <5 && PID_output >-5){
146     PID_output=0;
147   }
148   if(pitch>=45 || pitch<=-45){
149     PID_output=0;
150   }
151
152 //
153   pid_izq=PID_output;
154   pid_der=PID_output;
```

Como ya se explicó anteriormente, el funcionamiento de un controlador PID se basa en corregir el error recibido a la entrada. Dicho error en el caso de este proyecto será la diferencia entre el ángulo real medido y el ángulo deseado que en este caso será de 0 grados.

En el código se observa como la variable "PID_output" es la suma de los tres componentes del PID con sus respectivas ganancias cuyos valores han sido obtenidos a base de ensayo y error:

Donde toman los valores Kp= 37 ; Ki=2 y Kd=10.

Además, se ha implementado en el código una restricción para evitar problemas con el WindUp.

El WindUp es un problema que aparece cuando surgen un error en un tiempo prolongado, provocando que la parte integral se dispare a valores demasiado altos cuyos valores no son comprensibles en la realidad para el actuador, es por eso por lo que se implementa un control Anti WindUp poniendo como límite un valor de 400 cuentas.

También se ha limitado la salida del PID pues valores más altos de 400 y -400 provocan que los motores giren erraticamente, pues como ya se explicó antes el rango que tiene nuestros steppers con el microstepping implementado es tan solo de 800 pulsos.

Para acabar se han añadido otras restricciones para aumentar la eficacia del robot y una de ellas es bloquear los motores cuando el sistema esté muy próximo al equilibrio.

Y otra es parar completamente los motores cuando se experimente un ángulo de inclinación mayor de 45° tanto positivo como negativo significando que el robot se ha volcado por completo.

Finalmente, la salida del PID se copiará en las variables "pid_izq" y "pid_der" para su uso en los motores.

- **Bloque Pulso de Motores**

Una vez conseguido los valores necesarios para la creación de los pulsos, se debe linealizar el comportamiento de los motores stepper ya que estos no tienen un comportamiento lineal.

Con el siguiente algoritmo obtenido a base de varias pruebas y modificaciones con el Excel se consigue linealizar el comportamiento de dichos motores.

```
if (pid_izq > 0)pid_izq = 405 - (1 / (pid_izq + 9)) * 5500;
else if (pid_izq < 0)pid_izq = -405 - (1 / (pid_izq - 9)) * 5500;

if (pid_der > 0)pid_der = 405 - (1 / (pid_der + 9)) * 5500;
else if (pid_der < 0)pid_der = -405 - (1 / (pid_der - 9)) * 5500;
```

FIG. 56 LINEALIZACIÓN STEPPERS

El siguiente algoritmo mostrado a continuación sirve para conseguir el complementario del valor que se desea, es decir cuando estamos en un ángulo muy lejano al valor deseado el valor del PID se dispara que alcanzará como máximo un valor de ±400. Cuando este sea el caso "motor_izq" y "motor_der" tendrá como valor 0 o muy próximo a 0, el valor de estas variables se guardará en las variables "aceleracion_motor_izq" y "aceleracion_motor_der" respectivamente, que servirán para el contador de memoria de la rutina de interrupción.

```
164   //motor izquierdo
165   if(pid_izq>0){
166     motor_izq = 400-pid_izq;
167   }
168   else if(pid_izq < 0){
169     motor_izq= -400-pid_izq;
170   }
171   else motor_izq=0;
172
173   //motor derecho
174   if(pid_der>0){
175     motor_der= 400-pid_der;
176   }
177   else if (pid_der<0){
178     motor_der= -400 - pid_der;
179   }
180   else motor_der=0;
181
182   aceleracion_motor_izq = motor_izq;
183   aceleracion_motor_der = motor_der;
184
185 }
```

FIG. 57 VALOR ACELERACION

- **Rutina de Interrupción**

Finalmente, dentro de la rutina de interrupción que se activa cada 65us, se muestra el siguiente algoritmo que genera los pulsos necesarios para generar una PWM variable a los valores del PID, tanto para el motor izquierdo como para el derecho.

Este código se comporta de la siguiente manera, cada vez que el contador sea igual a 1 este generará un nivel alto a la patilla del step del driver. La segunda vez que interrumpa el contador pasará a ser 2, ya que siempre que entremos el contador se incrementará en una unidad.

El contador al llegar a 2 generará un nivel bajo sobre la patilla deseada, bajando así el pulso de nivel alto. El resto de los tiempos será decidido por el valor de "aceleracion_motor" es decir del valor que recibamos del PID. Si obtenemos una aceleración de bajo valor por ejemplo 4, el contador tardará solo 4+1 cuentas de 65us en volver a poner el pin 3 a nivel alto y en la siguiente interrupción volver a bajarlo.

Si por ejemplo tenemos un valor de "aceleracion_motor_" por encima cerca de los 400, esto quiere decir que nuestro PID está cerca del valor 0, diciendo así que estamos muy próximos al ángulo deseado.

Cuando esto ocurre, el motor tardará 400+1 cuentas de 65us en volver a poner el pin 3 a nivel alto y resetear el contador. Ya que no es necesario tener un PWM con tal alta frecuencia, sino más bien cuanto más tiempo se esté a nivel bajo hará que el motor se quede anclado en una posición sin girar, durante 401*65us= 0,026s o 26ms.

De esta forma se consigue una PWM dependiente del controlador PID para ambos motores.

```
187 ISR(TIMER2_COMPA_vect) {
188   //Pulsaciones motor izquierdo
189   contador_motor_izq ++;                                  //Aumentamos el contador siempre que se interrumpa
190   if (contador_motor_izq > mem_contador_motor_izq) {      //Si es mayor que la memoria_contador
191     contador_motor_izq = 0;                               //Reseteamos el contador
192     mem_contador_motor_izq = aceleracion_motor_izq;             //Cargamos en la memoria del contador el valor del motor izq
193     if (mem_contador_motor_izq < 0) {                     //Si la memoria del contador es negativa
194       PORTD &= 0b11110111;                                      //Ponemos la salida 3 a nivel bajo, para cambiar el sentido de giro
195       mem_contador_motor_izq *= -1;                       //Invertimos el signo de la memoria del contador
196     }
197    else PORTD |= 0b00001000;                                  //Ponemos la salida 3 a nivel alto
198   }
199   else if (contador_motor_izq == 1)PORTD |= 0b00000100;     //Salida 2 a nivel alto para generar un pulso
200   else if (contador_motor_izq == 2)PORTD &= 0b11111011;     //Salida 2 a nivel bajo para bajar el pulso tras los 65us
201
202   //Pulsaciones motor derecho
203   contador_motor_der ++;
204   if (contador_motor_der > mem_contador_motor_der) {
205     contador_motor_der = 0;
206     mem_contador_motor_der = aceleracion_motor_der;
207     if (mem_contador_motor_der < 0) {
208       PORTD |= 0b00100000;                                      //Ponemos la salida 5 a nivel bajo, para cambiar el sentido de giro
209       mem_contador_motor_der *= -1;
210     }
211     else PORTD &= 0b11011111;                                   //Ponemos la salida 5 a nivel alto
212   }
213   else if (contador_motor_der == 1)PORTD |= 0b00010000;     //Salida 4 a nivel alto para generar un pulso
214   else if (contador_motor_der == 2)PORTD &= 0b11101111;     //Salida 4 a nivel bajo para bajar el pulso tras los 65us
215 }
```

FIG. 58 RUTINA INTERRUPCIÓN

6 CAPITULO VI: Presupuesto

En este apartado se mostrará los costes de cada componente necesarios para la realización de este proyecto incluyendo los componentes que han tenido que ser reemplazados por fallos catastróficos.

Componente	Precio por unidad	Cantidad	Precio
Driver DDRV 8825	1,98 €	2	3,96 €
Sensor MPU6050	2,63 €	1	2,63 €
Arduino Nano	24,20 €	1	24,20 €
Cables Jumper Varios	4,99 €	1	4,99 €
Batería LiPo 2250mAh	19,07 €	1	19,07 €
Convertidor Reductor DC-DC	5,99 €	1	5,99 €
Motores Stepper Nema 17	10,96 €	2	21,92 €
Cargador Batería LiPo	17,99 €	1	17,99 €
Ruedas 75mm RC	3,20 €	2	6,40 €
Soporte Motor Nema 17	3,12 €	2	6,24 €
Acoplador hexagonal de eje	1,14 €	2	2,28 €
Cinta doble cara	6,99 €	1	6,99 €
Metacrilato 2mm 150x50cm	15,76 €	1	15,76 €
Varilla roscada M6 200mm	0,75 €	1	0,75 €
Placa Protoboard	4,90 €	1	4,90 €
Tuercas M6	1,89 €	1	1,89 €
TOTAL			145,96 €

Como podemos observar en la tabla de arriba el total de los componentes del propio robot suman un total de 145,96€.

El precio calculado más arriba no incluye los componentes extras usados y la licencia de programas usados el cual aumentarían el precio real.

En la siguiente tabla se añadirán los componentes extras usados y reemplazados.

Componente	Precio por unidad	Cantidad	Precio
Driver DDRV 8825	1,98 €	4	7,92 €
Sensor MPU6050	2,63 €	3	7,89 €
Arduino Nano	24,20 €	1	24,20 €
Soporte Motor Nema 17	3,12 €	2	6,24 €
Acoplador hexagonal de eje	1,14 €	2	2,28 €
TOTAL			48,53 €

Se adquirió un extra de componentes debido a errores en el conexionado que llevo al cortocircuito de algunos componentes como fue con el Arduino Nano y a dos drivers DDRV 8825. El resto de los componentes es debido a que venían en conjunto en cantidades de 4, llegando a usar un menor número de unidades en el robot final.

Si se tuviese que incluir las licencias de los programas utilizados para este proyecto tendríamos la siguiente tabla

Componente	Precio por unidad	Cantidad	Precio
Licencia AutoCAD 1 año	2.123,95 €	1	2.123,95 €
Licencia Fusión 360 1 año	365,00 €	1	365,00 €
Licencia Microsoft Office	99,00 €	1	99,00 €
TOTAL			2.587,95 €

Como observamos las licencias de los programas aumentaría el coste enormemente pero este precio no será incluido en precio final total del proyecto ya que gracias a la universidad de Alcalá de Henares los estudiantes poseemos licencia gratis para los programas de AutoCAD, Fusion360 y Microsoft Office.

Finalmente, el precio total del robot seria de **194,49€**

$$\boldsymbol{Total = 145,96€ + 48,53 = 194,49€}$$

7 Conclusiones

Una vez finalizado el proyecto podemos observar cómo aplicando leyes de control se logra mantener el sistema estable en una posición vertical y que este imita el comportamiento de un péndulo invertido.

Al realizar este proyecto se adquieren mayores nociones teóricas y de manera experimental el concepto de controlador PID y de los valores que este nos devuelve. También gracias a este proyecto se aprende como sintonizarlo de manera experimental en un sistema real.

Se puede observar como el sistema nunca llega a alcanzar el error ideal, es decir 0 pues en la realidad nunca se llegará a alcanzar dicho ángulo, pero si muy próximo a ello. También pasado un largo periodo el sistema no permanece en una posición fija y este empieza a ladearse sobre uno de sus lados, esto es debido a que el error de offset del sensor MPU6050 no es del todo cero incluso habiéndose calibrado con antelación.

Añadir un detalle muy curioso experimentado en este proyecto, es la extremada sensibilidad que poseen los sensores de movimiento y la cantidad de ruido que pueden recibir por tanto la implementación del filtro complementario es muy importante a la hora de trabajar con los datos que se recibe.

En el conexionado se ha llegado a la conclusión de la precaución que hay que tomar cuando se conectan ciertos componentes pues por un descuido al inicio del proyecto se conectó a 11.1V la patilla de 5V del Arduino, provocando así su rotura.

Para realizar este proyecto se ha tenido que adquirir un conocimiento previo en el campo de ingeniería de control y también en programación de microcontroladores.

Para trabajos futuros se puede implementar en el robot mejoras como el control de manera remota con una aplicación del móvil vía bluetooth o con un joystick, a la vez que también se pueda ajustar las ganancias del controlador y el ángulo de inclinación deseado de manera remota e instantánea para que sea capaz de adaptarse sobre el terreno sobre el que se encuentre.

8 Referencias y bibliografía

[1] "Datasheet MPU6050" consultado en:

https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf

[2] "Datasheet NEMA17" consultado en:
https://datasheetspdf.com/pdf/1260602/Schneider/NEMA17/1

[3] Control of Stepping motors, a tutorial. Douglas W. Jones
http://homepage.divms.uiowa.edu/~jones/step/

[4] "Datasheet driver DRV8825", consultado en:
https://www.ti.com/lit/ds/symlink/drv8825.pdf?ts=1631719013005&ref_url=https%253A%252F%252Fwww.google.com%252F

[5] Luis Geovanny Triviño Macias "modelado, simulación y control de un péndulo invertido"

[6] Aracil, J. and Gordillo, F., 2005, "El Péndulo Invertido. Un Desafío Para el Control no Lineal", Revista Interamericana de Automática e Informática Industrial

[7] Ooi, R. 2003 "Balancing a Two-Wheeled Autonomous Robot". University of Western Australia.

[8] Grasser, F. D'Arrigo, A. Colombi, S. and Rufer,A 2002 ; "Joe: A Mobile, Inverted Pendulum",, Swiss Federal Insitute

[9] Librería Arduino disponible en : https://www.arduino.cc/

[10] "Stepper Motor Study" Techno Instruments disponible en:
http://www.ee.iitb.ac.in/~ccgroup/old/Lab_pages/experiment_files/manual_stepper.motor.study.pdf

[11] "Diseño y control de un sistema electromecánico inestable de dos grados de libertad: aplicación al caso de un péndulo invertido" Eugenio Molis Merino, Universidad politécnica de Cataluña.

[12] "Stepper Motor Model for Dynamic Simulation" Alexandru Morar disponible en:

https://ie.utcluj.ro/files/acta/2003/Number%202/Paper08_Morar.pdf

[13] "definición Bateria" RAE

9 Listado de componentes

- 1 unidad placa Arduino nano, ATmega328
- 2 unidades de drv8825
- 1 unidad de MPU6050
- 2 motores paso-paso Nema 17.
- 2 soportes metálicos para los motores.
- Varios cables jumper macho-macho.
- 1 unidad de placa protoboard
- 1 convertidor DC-DC ajustable.
- 4 varillas roscadas de 20cm M6.
- 20 tuercas M6.
- 1 plancha de metacrilato 150x50 cm.
- 1 batería LiPo 2250 mAh
- 1 cargador de batería LiPo

Anexos

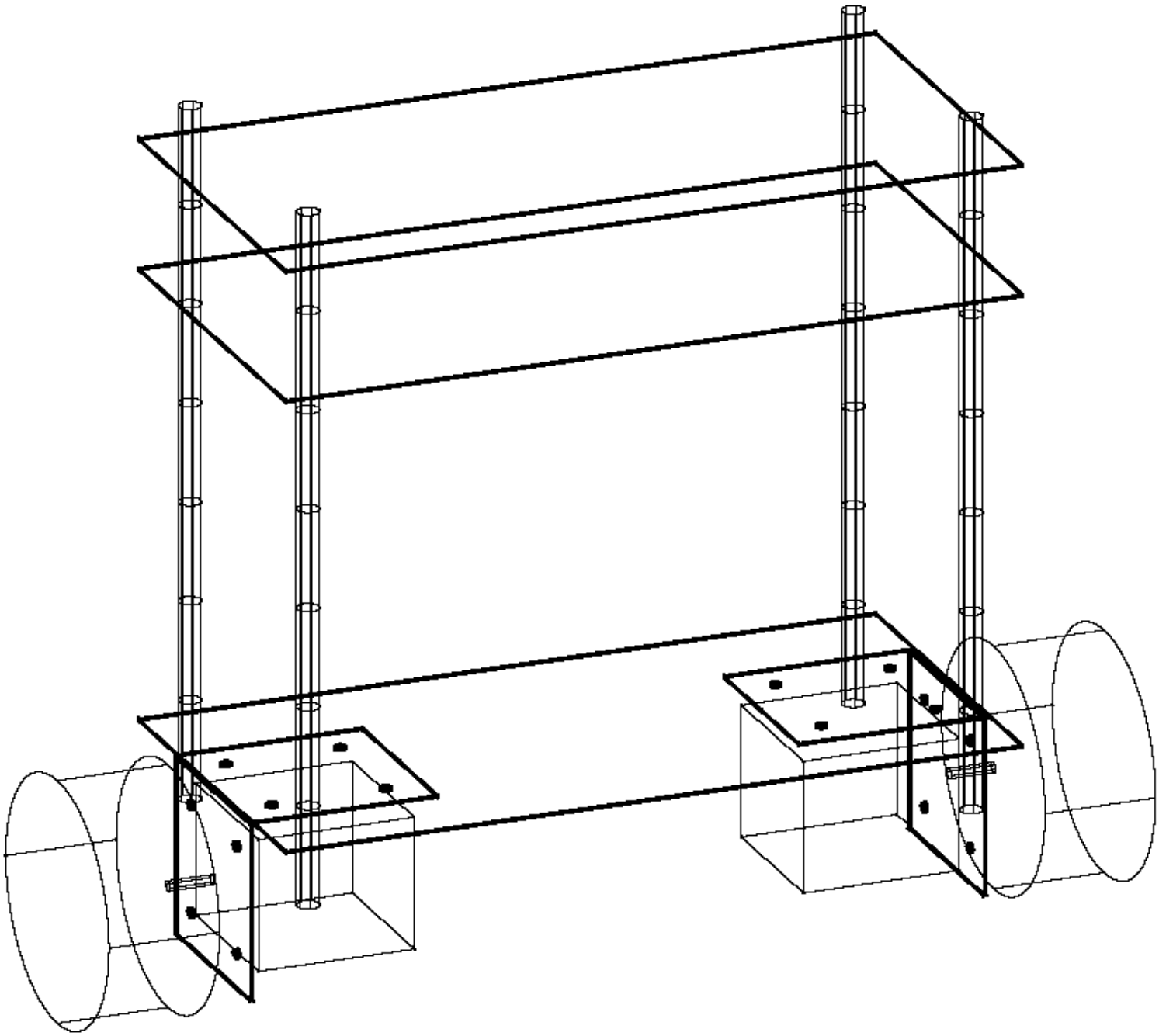
```
#include <Wire.h>
#include "MPU6050.h"

unsigned long loop_timer;

int dir_MPU = 0x68;
MPU6050 mpu(dir_MPU);

float Tiempo_transcurrido;
float Tiempo_pasado;
float Tiempo_actual;

float gyro_ang_y, gyro_ang_y_anterior;
float pitch, pitch_ant;
//variables para la lectura del MPU
int ax, ay, az, gx, gy, gz;
float AcX, AcY, AcZ, GyX, GyY, GyZ;

//variables para el PID

float kp=37; //37 funciona muy muy bien
float ki=2; //5 bien
float kd=10; //10 bastante correcto

float autobalanceo_pid=0;
float angulo_deseado=0;
float error, error_ant;
float PID_output;
float pid_p=0;
float pid_i=0;
float pid_d=0;

float pid_izq, pid_der, motor_izq, motor_der, velocidad_izq,
velocidad_der;

float aceleracion_motor_izq, contador_motor_izq,
mem_contador_motor_izq;
float aceleracion_motor_der, contador_motor_der,
mem_contador_motor_der;

//variables motores
int dirPin1=3;
int stepPin1=2;
int dirPin2=5;
int stepPin2=4;


void setup() {

  Serial.begin(9600);
// Serial port a 9600 kbps
  Wire.begin();
// Iniciamos la comunicación I2C
  TWBR=12;
// Velocidad del I2C a 400Khz   SCL frequency = CPU clock freq / 16 +
2(TWBR)*(preescaler)

  //___________configuracion como salida de los pines de los Driver
DDRV8825_________________
```

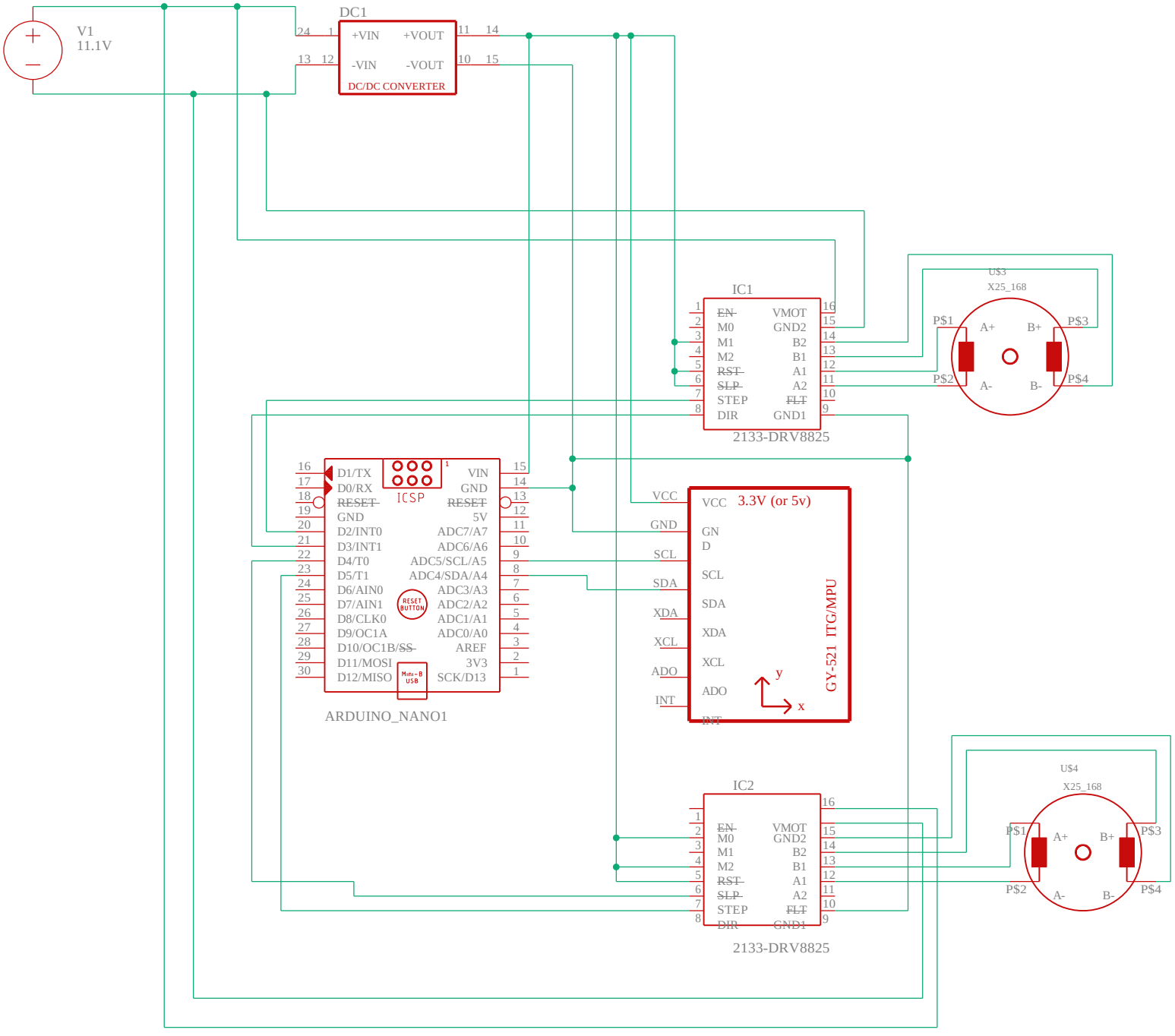
```
  pinMode(dirPin1, OUTPUT);
//puerto D2 como salida para los drivers del motor
  pinMode(stepPin1, OUTPUT);
//puerto D3 como salida para los drivers del motor
  pinMode(dirPin2, OUTPUT);
//puerto D4 como salida para los drivers del motor
  pinMode(stepPin2, OUTPUT);
//puerto D5 como salida para los drivers del motor

 //____configuración del timer2 de 8 bits para controlar los motores
interrumpe cada 65us_____________

  TCCR2A = 0;
//Ponemos TCCR2A register a 0 para limpiar los registros
  TCCR2B = 0;
//Ponemos TCCR2B register a 0 para limpiar los registros
  TIMSK2 |= (1 << OCIE2A);
//La interrupción del Timer/Counter2 compare match A activa.
  TCCR2B |= (1 << CS21);
//Ponemos el CS21 bit para poner el preescaler a 8 en el registro
TCCR2B.
  OCR2A = 129;
//El registro comparador se pone a 129 interrumpiendo cada 65us =>
65us/(1s/(16Mhz/8))-1
  TCCR2A |= (1 << WGM21);
//Reseteamos el timer 2 en modo compare


  //____________________configuración del
MPU6050_____________________________//
  mpu.initialize();
  Wire.beginTransmission(dir_MPU);
  Wire.write(0x6B); // PWR_MGMT_1 registro;
  Wire.write(0x00);
  Wire.endTransmission();
  //Configurar el giro a +/-250 grados por segundo
  Wire.beginTransmission(dir_MPU);
  Wire.write(0x1B);
//GYRO_CONFIG registro
  Wire.write(0x00);
//250 grados por segundo full scale
  Wire.endTransmission();
  //Configurar el acelerómetro a +/- 2g
  Wire.beginTransmission(dir_MPU);
  Wire.write(0x1C);
//registro ACCEL_CONFIG
  Wire.write(0x10);
//bits a 00001000 para conseguir +/- 4g
  Wire.endTransmission();
  //Activamos un filtro del propio sensor
  Wire.beginTransmission(dir_MPU);
  Wire.write(0x1A);
//registro CONFIG
  Wire.write(0x03);
//Filtro paso bajo a ~43Hz
  Wire.endTransmission();

    mpu.setXAccelOffset(-975); //-971
    mpu.setYAccelOffset(635);  //679
    mpu.setZAccelOffset(938);  //933
    mpu.setXGyroOffset(9);    //7
```

```
    mpu.setYGyroOffset(45);    //47
    mpu.setZGyroOffset(8);    //9
}

void loop() {

  mpu.getAcceleration(&ax, &ay, &az);
  mpu.getRotation(&gx, &gy, &gz);

  GyX= (gx)/131;
  GyY= (gy)/131;
  GyZ= (gz)/131;

  float acc_ang_y = atan((ax/8192.0) / sqrt(pow((ay/8192.0), 2)+
pow((az/8192.0), 2)))*RAD_TO_DEG; //hay que dividir los valores de la
aceleracion puros entre 16384 segun el data sheet

  Tiempo_pasado = Tiempo_actual;
  Tiempo_actual = millis();
  Tiempo_transcurrido= (millis() - Tiempo_pasado)/ 1000; //para
conseguirlo en segundos.

  gyro_ang_y = gyro_ang_y + (GyY*Tiempo_transcurrido); //los valores
estan en grados/s por tanto multiplicaremos por el tiempo transcurrido
y obtendremos los grados

  pitch = 0.98*gyro_ang_y + 0.02*acc_ang_y; //0.98 + 0.02

//  Serial.print(acc_ang_y);
//  Serial.print(" ");
//  Serial.print(gyro_ang_y);
//  Serial.print(" ");
//  Serial.print("\tpitch: ");
//  Serial.println(pitch);
//  Serial.print(" ");

  ////////calculo para el PID//////////

  error = pitch - angulo_deseado;
  pid_p = kp*error;
  pid_i= pid_i + (ki*error);
  if (pid_i>400)pid_i=400;
  else if(pid_i<-400)pid_i=-400;

  pid_d= kd*((error-error_ant));

  PID_output= pid_p + pid_i + pid_d;

  if (PID_output > 400){
    PID_output=400;
  }
  if(PID_output< -400){
    PID_output=-400;
  }
    error_ant = error;

  if(PID_output <5 && PID_output >-5){
    PID_output=0;
  }
  if(pitch>=45 || pitch<=-45){
    PID_output=0;
```

```
  }

//
  pid_izq=PID_output;
  pid_der=PID_output;


//_______CALCULO PULSO DE MOTOR_____________//
  if (pid_izq > 0)pid_izq = 405 - (1 / (pid_izq + 9)) * 5500;
  else if (pid_izq < 0)pid_izq = -405 - (1 / (pid_izq - 9)) * 5500;

  if (pid_der > 0)pid_der = 405 - (1 / (pid_der + 9)) * 5500;
  else if (pid_der < 0)pid_der = -405 - (1 / (pid_der - 9)) * 5500;

  //motor izquierdo
  if(pid_izq>0){
    motor_izq = 400-pid_izq;
  }
  else if(pid_izq < 0){
    motor_izq= -400-pid_izq;
  }
  else motor_izq=0;

  //motor derecho
  if(pid_der>0){
    motor_der= 400-pid_der;
  }
  else if (pid_der<0){
    motor_der= -400 - pid_der;
  }
  else motor_der=0;

  aceleracion_motor_izq = motor_izq;
  aceleracion_motor_der = motor_der;

}

ISR(TIMER2_COMPA_vect) {
  //Pulsaciones motor izquierdo
  contador_motor_izq ++;                  //Aumentamos el contador
siempre que se interrumpa
  if (contador_motor_izq > mem_contador_motor_izq) {    //Si es mayor
que la memoria_contador
    contador_motor_izq = 0;                              //Reseteamos
el contador
    mem_contador_motor_izq = aceleracion_motor_izq;//Cargamos en la
memoria el valor del motor izq
    if (mem_contador_motor_izq < 0) {//Si la memoria del contador es
negativa
      PORTD &= 0b11110111; //Ponemos la salida 3 a nivel bajo, para
cambiar el sentido de giro
      mem_contador_motor_izq *= -1;  //Invertimos el signo de la
memoria del contador
    }
   else PORTD |= 0b00001000; //Ponemos la salida 3 a nivel alto
  }
  else if (contador_motor_izq == 1)PORTD |= 0b00000100; //Salida2 a
nivel alto generarando un pulso
  else if (contador_motor_izq == 2)PORTD &= 0b11111011; //Salida2 a
nivel bajo para bajar el pulso tras los 65us
```

```
  //Pulsaciones motor derecho
  contador_motor_der ++;
  if (contador_motor_der > mem_contador_motor_der) {
    contador_motor_der = 0;
    mem_contador_motor_der = aceleracion_motor_der;
    if (mem_contador_motor_der < 0) {
      PORTD |= 0b00100000;
//Ponemos la salida 5 a nivel bajo, para cambiar el sentido de giro
      mem_contador_motor_der *= -1;
    }
    else PORTD &= 0b11011111;
//Ponemos la salida 5 a nivel alto
  }
  else if (contador_motor_der == 1)PORTD |= 0b00010000;
//Salida 4 a nivel alto para generar un pulso
  else if (contador_motor_der == 2)PORTD &= 0b11101111;
//Salida 4 a nivel bajo para bajar el pulso tras los 65us
}
```




V1
11.1V
DC1
+VIN
+VOUT
-VIN
-VOUT
DC/DC CONVERTER
IC1
EN
M0
M1
M2
RST
SLP
STEP
DIR
VMOT
GND2
B2
B1
A1
A2
FLT
GND1
2133-DRV8825
U$3
X25_168
P$1
P$2
P$3
P$4
A+
A-
B+
B-
ARDUINO_NANO1
D1/TX
D0/RX
RESET
GND
D2/INT0
D3/INT1
D4/T0
D5/T1
D6/AIN0
D7/AIN1
D8/CLK0
D9/OC1A
D10/OC1B/SS
D11/MOSI
D12/MISO
ICSP
RESET BUTTON
Mini-B USB
VIN
GND
RESET
5V
ADC7/A7
ADC6/A6
ADC5/SCL/A5
ADC4/SDA/A4
ADC3/A3
ADC2/A2
ADC1/A1
ADC0/A0
AREF
3V3
SCK/D13
GY-521 ITG/MPU
VCC
3.3V (or 5v)
GN
D
SCL
SDA
XDA
XCL
ADO
INT
y
x
IC2
U$4

ESCUELA POLITECNICA SUPERIOR

Universidad de Alcalá