Universidad de Oviedo

ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN

INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

AREA DE TECNOLOGÍA ELECTRÓNICA

TRABAJO FIN DE GRADO
BANCO EDUCACIONAL DE SISTEMAS ELECTRÓNICOS PARA EL ACCIONAMIENTO DE MOTORES ELÉCTRICOS BASADO EN ESP32

D. Mallada Fernández, Daniel
TUTOR/ES:
D. Calleja Rodriguez, Antonio Javier
D. Rico Secades, Manuel

FECHA: Noviembre 2022

Índice de contenido

Índice de figuras

Índice de tablas

1. Introducción

Hoy en día una de las principales ramas de estudio de la ingeniería y donde mayor trabajo hay, es en la tecnologías IoT. Estas tecnologías combinan el uso de la conexión a Internet con distintos periféricos y dispositivos que con capaces de conectarse a la red.

Por otra parte, tenemos los motores eléctricos, estos pueden ser de varios tamaños, potencias, velocidades, etc. Por tanto tenemos una gran infinidad de ellos, podemos abarcar de motores de grandes velocidades, que son aplicados en el mundo de los drones, motores de posición, estos compensan su baja velocidad, con un gran control de la posición de los mismos.

El Internet de las cosas (IoT) ha revolucionado la forma en que interactuamos con los objetos y dispositivos de nuestro entorno. Esta tecnología ha llegado también a los motores, brindando nuevas oportunidades de control y optimización. El uso de IoT en los motores permite monitorear y analizar en tiempo real el rendimiento y estado de estos componentes, facilitando el mantenimiento preventivo y la detección temprana de averías. Además, el IoT en motores posibilita la implementación de sistemas de automatización y gestión inteligente, mejorando la eficiencia energética y reduciendo costos operativos. En este sentido, el uso de IoT en motores representa un paso adelante en la industria, proporcionando un mayor control y eficiencia en el funcionamiento de los motores.

La aplicación de IoT en el control de motores ha revolucionado la industria en los últimos años. Gracias a la conectividad y los avances tecnológicos, ahora es posible monitorear, controlar y optimizar el rendimiento de los motores de manera remota y en tiempo real.

Una de las prnicipales ventajas de la aplicación de IoT en el control de motores es la capacidad de recopilar datos y analizarlos para obtener información valiosa sobre su funcionamiento. Sensores instalados en los motores pueden medir variables como la temperatura, la presión y la velocidad, entre otras. Estos datos se transmiten a través de una red inalámbrica a una plataforma de IoT, donde se analizan y se generan informes detallados.

Con esta información, los ingenieros pueden detectar posibles problemas o fallas en los motores y tomar medidas preventivas antes de que ocurra un problema mayor. Esto mejora significativamente la eficiencia operativa y reduce los tiempos de inactividad no planificados, lo que se traduce en ahorros de costos para las empresas.

Otra aplicación destacada es la capacidad de controlar múltiples motores desde una sola plataforma. Esto es especialmente útil en grandes instalaciones industriales donde se pueden

tener cientos o incluso miles de motores en funcionamiento. A través de una interfaz centralizada, los operadores pueden supervisar y controlar cada motor de forma individual, lo que facilita la gestión y el mantenimiento de la planta.

Además, la aplicación de IoT en el control de motores permite la optimización del rendimiento. Los motores pueden ser ajustados de forma remota para maximizar su eficiencia y adaptarse a las necesidades específicas de la operación. Esto no solo reduce el consumo energético, sino que también prolonga la vida útil de los motores y reduce las emisiones de carbono.

En resumen, la aplicación de IoT en el control de motores ha revolucionado la forma en que se gestionan y utilizan los motores en todo tipo de industrias. Gracias a la conectividad y la capacidad de análisis de datos en tiempo real, las empresas pueden mejorar la eficiencia operativa, reducir costos y optimizar el rendimiento de sus motores.

Los profesores y directores del TFG, me propusieron este trabajo para que en un futuro sirviese de apoyo a las clases de accionamientos electrónicos y asi familiarizar a los alumnos con un nuevo hardware, muy extendido, asi como el lenguaje de programación como el Python.

El uso del ESP32 como control de motores y Python como lenguaje de programación ofrece una serie de ventajas y bondades que benefician a los proyectos y aplicaciones en los que se utilizan. A continuación, se detallan algunas de estas bondades:

Versatilidad: El ESP32 es un microcontrolador con capacidades de conectividad inalámbrica, lo que permite controlar los motores de forma remota a través de Wi-Fi o Bluetooth. Esto lo hace ideal para aplicaciones domóticas, robótica o automatización industrial.

Potencia de procesamiento: El ESP32 cuenta con un procesador de doble núcleo y una velocidad de reloj de hasta 240 MHz, lo que permite realizar cálculos complejos y procesar datos en tiempo real. Esto resulta especialmente útil en sistemas de control de motores que requieren de una alta capacidad de procesamiento.

Fácil programación: Python es un lenguaje de programación de alto nivel y fácil aprendizaje, lo que lo convierte en una opción ideal para aquellos que se están iniciando en la programación. Su sintaxis clara y legible facilita la escritura y comprensión del código, lo cual agiliza el desarrollo de aplicaciones de control de motores.

Enorme comunidad de desarrolladores: Python cuenta con una amplia comunidad de desarrolladores que comparten sus conocimientos y contribuyen a la creación de librerías y

frameworks para el control de motores. Esto facilita enormemente el desarrollo de proyectos, ya que se pueden encontrar soluciones a problemas comunes y aprovechar el trabajo de otros desarrolladores

Librerías y módulos específicos: Tanto el ESP32 como Python cuentan con una gran cantidad de librerías y módulos específicos para el control de motores. Estas librerías permiten controlar fácilmente la velocidad, dirección y posición de los motores, así como acceder a diferentes configuraciones y parámetros. Esto agiliza el desarrollo de proyectos y reduce la cantidad de código necesario.

Integración con otros dispositivos y plataformas: El ESP32 y Python presentan una excelente compatibilidad con otros dispositivos y plataformas. Esto permite integrar el control de motores con otros sistemas y tecnologías, como IoT, sensores, bases de datos, servicios en la nube, entre otros. De esta manera, se pueden crear soluciones más completas y escalables.

En resumen, usar el ESP32 como control de motores y Python como lenguaje de programación ofrece versatilidad, potencia de procesamiento, facilidad de programación, gran comunidad de desarrolladores, librerías específicas y amplia integración con otros dispositivos y plataformas. Estas bondades hacen de esta combinación una elección ideal para proyectos de control de motores, proporcionando un alto grado de flexibilidad y eficiencia en su desarrollo.

Entonces, que pasaría si juntamos los motores eléctricos, con las tecnologías IoT, los microcontroladores de bajo consumo ESP32, y Python, tenemos como resultado una nueva manera de ver el mundo de los accionamientos eléctricos, las conexiones a distancia y el mundo de los nuevos microcontroladores.

La automatización de los brazos robóticos, el control y monitorización de las plantas, el vuelo de un dron, todas estas ideas combinan todo lo que buscamos, por lo que en este trabajo vamos a buscar el facilitar el entendimiento y combinación de todas estas tecnologías, con la fabricación de una placa que permita la aplicación de estas.

2. Objetivos del Trabajo

El objetivo de este trabajo es el desarrollo de un banco educacional para el trabajo, simulación, control, asimilación del funcionamiento y comprensión de los motores eléctricos, partiendo del uso del ESP32, como microcontrolador para realizar el control que además permita usar los distintos lenguajes de programación que admita nuestro MCU, Arduino, C, Espresiff ...

La idea de este trabajo nació de la idea de implementar y documentar el MicroPython como nuevo lenguaje de programación para los microcontroladores, combinándolo con las nuevas tecnologías IoT, y el uso de los motores eléctricos que se estudian en la asignatura de *Accionamientos Electrónicos*.

Por parte de los profesores y tutores del TFG, les parece buena idea poder implementar este firmware en sus correspondientes asignaturas, debido a que piensan que la curva de dificultad de este lenguaje se reduciría de gran manera con respecto a otros lenguajes, debido a que estamos tratando con un lenguaje base que es el Python, entonces se podría realizar muchas mas tareas, aumentar el aprendizaje y hacerlo en un tiempo mas corto, además de ser pioneros en la implementación de este firmware en toda la universidad.

Una vez ya visto el nacimiento de esta idea, los objetivos a cumplir, serán la compresión del funcionamiento de los distintos motores, el motor Brushless, el motor Paso a Paso y el motor de Continua. En este trabajo se analizará el funcionamiento de los mismos, se analizará en el laboratorio y se realizará un programa de control para los mismos.

Una vez realizado toda la documentación sobre los motores, instalación del firmware y realización de los programas de control, procederemos al desarrollo de una placa de control, única, para todos los motores vistos, de esta manera podremos ensayar los distintos motores, sin necesidad de protoboards, ni de cableado externo, solo necesitaremos nuestro MCU, y los motores.

Además buscamos que el banco incorpore mas opciones como, pulsadores, leds y un potenciómetro, para poder aprovechar al máximo el potencial de la placa.

Recalcamos que nuestro fin solo es educacional, con el fin de que poder promover el uso de este firmware, y permitir al usuario un aprendizaje sencillo y didáctico de los accionamientos eléctricos.

Una vez analizado los objetivos a conseguir y el propósito del trabajo, abordaremos el

trabajo desde el porque hemos elegido el microcontrolador, como instalar el firmware y el software necesario. Seguido de esto, pondremos a disposición de todos un breve tutorial sobre el uso de las configuraciones mas simples del MicroPython, como viene a ser el puerto ADC o el PWM.

Seguido de esto analizaremos los distintos motores, y realizaremos un breve programa de control para cada uno de ellos.

Por último realizaremos y diseñaremos la correspondiente PCB, en la que aplicaremos todos estos conceptos anteriores.

3. ESP32

El ESP32 es un microcontrolador de bajo costo y bajo consumo de energía que pertenece a la serie de microcontroladores ESP8266 de Espressif Systems. Tiene un procesador de doble núcleo a 32 bits, con una frecuencia de reloj de hasta 240 MHz.

El ESP32 cuenta con una memoria flash integrada de hasta 16 MB y una memoria SRAM de hasta 520 KB. Además, ofrece conectividad Wi-Fi y Bluetooth de doble modo (clásico y BLE), lo que lo convierte en una opción ideal para aplicaciones de IoT (Internet de las cosas) ([1]).

Algunas de las características generales del ESP32 son:

- **Procesador dual-core**: Cuenta con dos núcleos de procesamiento, lo que permite realizar múltiples tareas de manera más eficiente.
- **Conectividad Wi-Fi**: Permite la conexión a redes inalámbricas para acceder a internet y realizar comunicación con otros dispositivos.
- **Conectividad Bluetooth**: Además de Wi-Fi, el ESP32 también proporciona conectividad Bluetooth, lo que amplía las posibilidades de comunicación con otros dispositivos.
- **Baja potencia**: El ESP32 está diseñado para ser eficiente en términos de consumo de energía, permitiendo una mayor duración de la batería en aplicaciones móviles o con alimentación limitada.
- **Amplia memoria**: La memoria flash integrada del ESP32 permite almacenar programas y datos, mientras que la memoria SRAM proporciona espacio para almacenar variables y ejecutar programas.
- **Diversidad de puertos y GPIO**: El ESP32 ofrece una variedad de puertos y pines GPIO, lo que facilita la conexión de sensores, actuadores y otros periféricos.
- **Soporte de desarrollo**: El ESP32 es compatible con el entorno de desarrollo Arduino, lo que facilita su programación y desarrollo de aplicaciones.

En resumen, el ESP32 es un microcontrolador versátil y potente que combina conectividad Wi-Fi y Bluetooth, junto con capacidades de procesamiento de doble núcleo y una amplia memoria, lo que lo convierte en una opción popular para proyectos de IoT y aplicaciones inalámbricas.

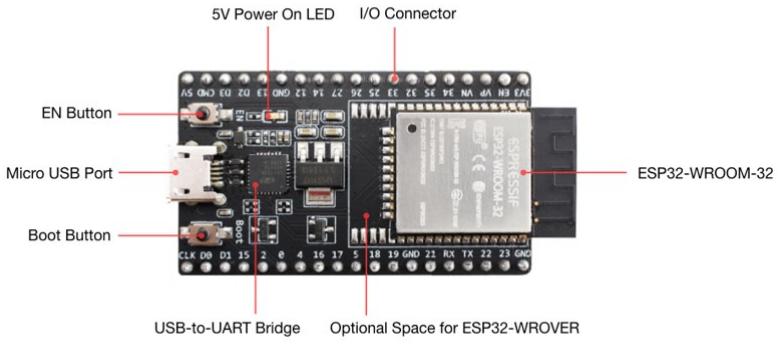

Figura 3.1: ESP32 (ESPRESIFF)

En este caso, trabajaremos con el modelo de **ESP32-DevKitC-32-UE**, es el que se muestra en la figura 3.1. Este será nuestro referente para la realización de las prácticas y el diseño de la placa PCB, de la que se realizará mas adelante.

3.1.- Estructura del ESP32

En esta parte del capítulo, vamos a analizar el diagrama de bloques del MCU en cuestión. En este caso recordar que estamos tratando con el modelo específico de ESP32, el **ESP32-WROOM-32E**.

En el diagrama de bloques del ESP32 (Figura 3.2), se nos muestra las conexiones de nuestro chip, este se encuentra conectado a un cristal de cuarzo de 40 MHz, el cristal será el encargado de generador de frecuencias alta, para lograr el funcionamiento de la memoria interna interna del MCU a altas velocidades. A este se nos une el **RF Matching**, también denominada red de adaptación, esta nos permita adaptar las radio frecuencia que entren por la antena interna o externa del microcontrolador.

Como ultimo bloque disponemos de la unidad **QSPI Flash**, es una unidad flash externa que nos permite ampliar la memoria interna del microcontrolador y realizar tareas que necesiten un mayor tamaño de memoria, por tanto es una buena opción para realizar aplicaciones que necesiten de una alta capacidad de memoria.

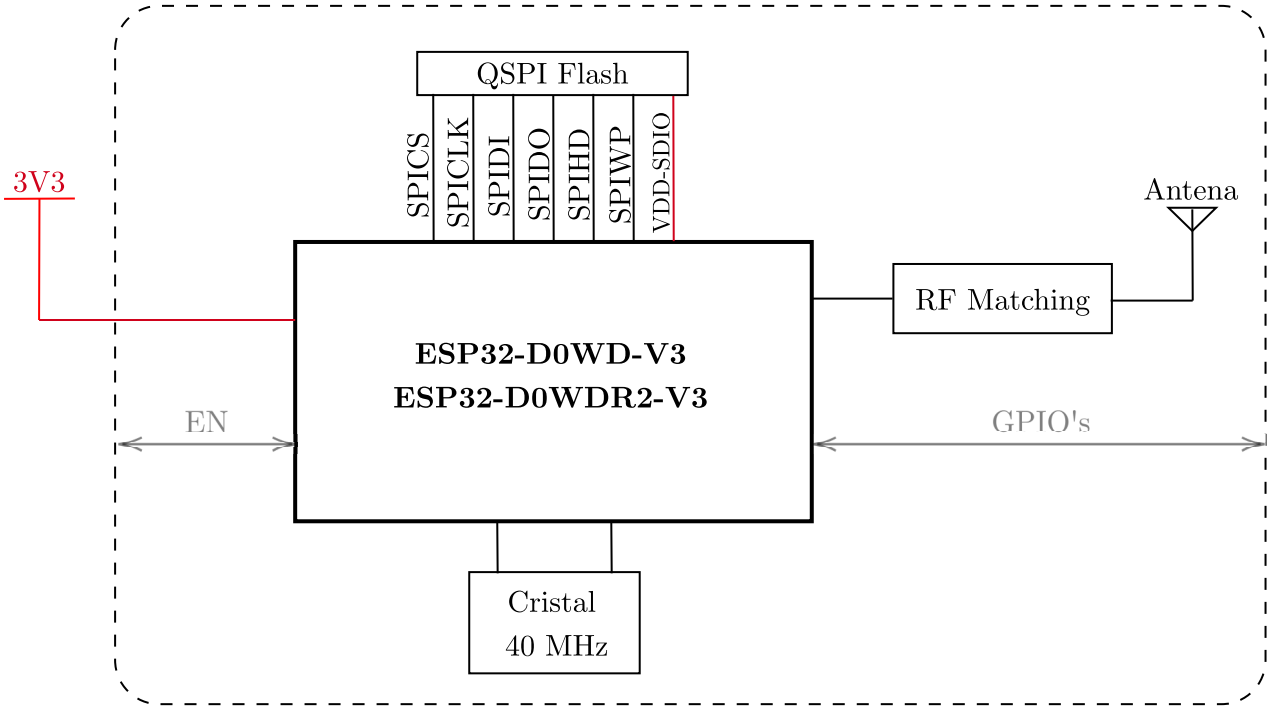

Figura 3.2: Diagrama de bloques del ESP32

3.2.- Características Eléctricas

Para poder trabajar con el ESP32, vamos a analizar las características eléctricas del mismo, aquellas que nos puedas limitar en algún momento, y pueda impedir hacer la tarea tal como nosotros queremos.

Todos estos datos, son sacados del datasheet del fabricante (Datasheet ESP32).

Debemos tener en cuenta que a diferencia los MCU de microchip, con los que tanto hemos trabajado, las entradas del ESP32, están limitadas a 3.6 Voltios, por lo que debemos de tener cuidado con esto.

La temperatura máxima de trabajo, nos asegura que puede ir desde -40 ºC hasta los 105 ºC.

En cuanto la lógica digital de entrada, tenemos que es:

$$V_{entrada}(V) = \begin{cases} 1, \text{ si } V > 0{,}75 \cdot VDD \\ 0, \text{ si } V < 0{,}25 \cdot VDD \end{cases}$$

Vemos que la lógica del mismo viene dada en función de la tensión VDD, que viene a

ser la tensión de alimentación del mismo, el fabricante recomienda que dicha tensión vaya desde 3 Voltios hasta 3.6.

En cuanto a la la lógica de salida del mismo, tenemos que:

$$V_{salida}(V) = \begin{cases} 1, \text{ si } V > 0{,}8 \cdot VDD \\ 0, \text{ si } V < 0{,}1 \cdot VDD \end{cases}$$

En este apartado, hemos querido hacer un breve resumen sobre las preocupaciones del ESP32, y cuales podrían suponer un problema o llegar a estropear el MCU.

3.3.- Pines del ESP32

Tal como se ha comentado a principios del capítulo, tenemos un microcontrolador de 39 pines, estos tienen funcionen específicas y complementarias, por lo que disponemos de una infinidad de combinaciones y posibilidades, para el uso de estos.

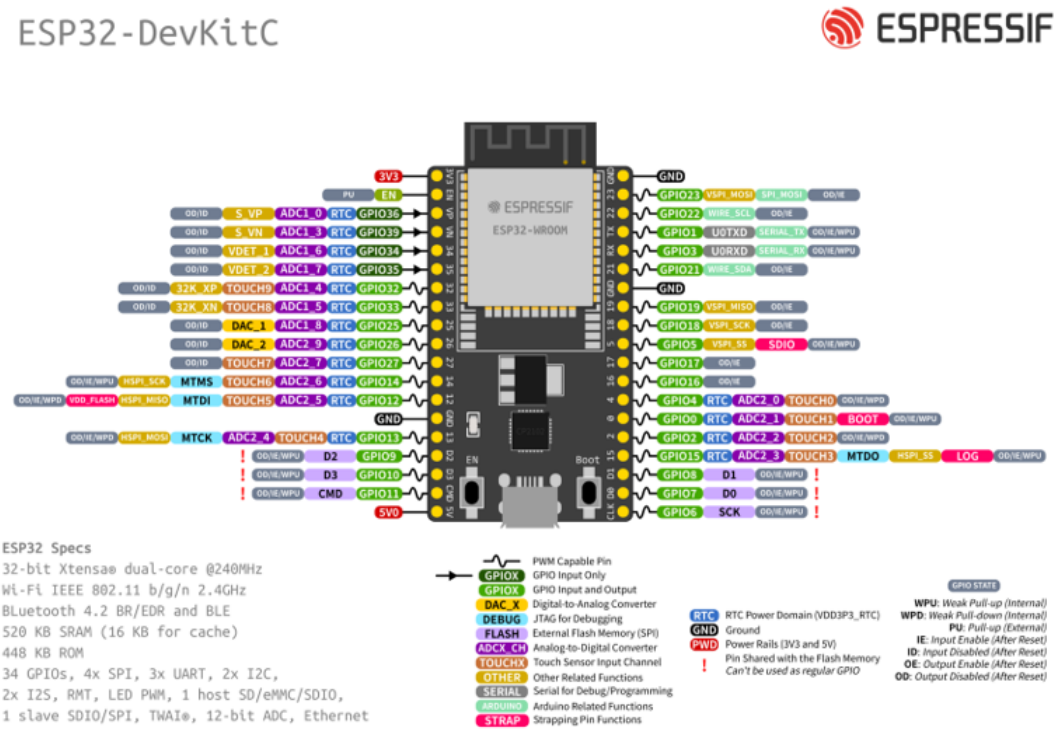

Figura 3.3: Pin OUT del ESP32 (ESPRESSIF)

En la figura 3.3, se nos muestra el esquema de los pines del ESP32, junto al tipo de Pin, y las funcionalidades de estos mismo, como salidas PWM, el tipo de salida, si pueden funcionar como salidas o entradas analógicas, etc. En este caso al disponer de mas de 28

puertos PWM, no vamos a tener problemas algunos para la distribución de los mismos, al igual que el tipo, debido a que todos pueden funcionar como puertos de entrada o salida, a excepción de los puertos GPIO36, GPIO39, GPIO34 y GPIO35.

El motivo por el que hemos elegido este microcontrolador ha sido por el gran número de pines, además de su múltiple numero de pines que se pueden utilizar como puertos PWM.

En la figura 3.3, se nos muestra la función que pueden tener todos los pines del ESP32, que hemos elegido. Al disponer de este microcontrolador dispondremos de una gran libertad a la hora de diseñar la correspondiente PCB, en función a nuestro MCU.

4. Introducción e instalación del software de MicroPython

En este capítulo abordaremos el software necesario y los conceptos básicos para empezar poder hacer uso del software, de manera óptima, dicho esto, comenzaremos con la instalación.

4.1.- Instalación de software

Para este apartado, se aborda la instalación del software que se utilizará en la programación del micro que vamos a utilizar.

A continuación, se detallara paso a paso toda la instalación del software que se utilizará en el desarrollo del trabajo.

Como primer paso deberemos instalar el **Visual Studio Code**, este es un entorno de programación que permite que soporta múltiples lenguajes de programación, la instalación se puede hacer desde el siguiente link `https://code.visualstudio.com/`, una vez nos dirigimos al link, entraremos en la página principal de **VS Code**, en la pagina principal, 4.1, nos dirigiremos a la opción de descarga, una vez descargado el *.exe* lo instalaremos de forma normal, aceptando los uso y condiciones de aplicación, una vez lo tengamos instalado saltaremos al siguiente paso.

Una vez instalado, abriremos la aplicación de VS Code, la pantalla de inicio se mostrará como en la imagen 4.2, la versión de por si no trae ningún lenguaje de programación instalado, eso lo tendremos que hacer nosotros. En nuestro caso solo necesitamos instalar *Python*, el lenguaje de interacción entre el ESP32 y el microcontrolador, para instalarlo solo debemos ir a la opción de *extensiones*, es la quinta opción, de la columna de la izquierda, empezando desde arriba.

Una vez nos dirigimos a las extensiones y buscaremos *Python*, en ellas encontraremos varias opciones, instalaremos la primera opción que nos sale, para instalarlo solo debemos darle al botón de instalar, tal como se ve en la imagen 4.3.

Una vez ya tengamos la una aplicación para interactuar con el microcontrolador y desarrollar nuestros programas, pero solo tendemos la aplicación, nos falta por instalar un medio para comunicarnos con el microcontrolador, el firmware, en este caso tendremos que instalar

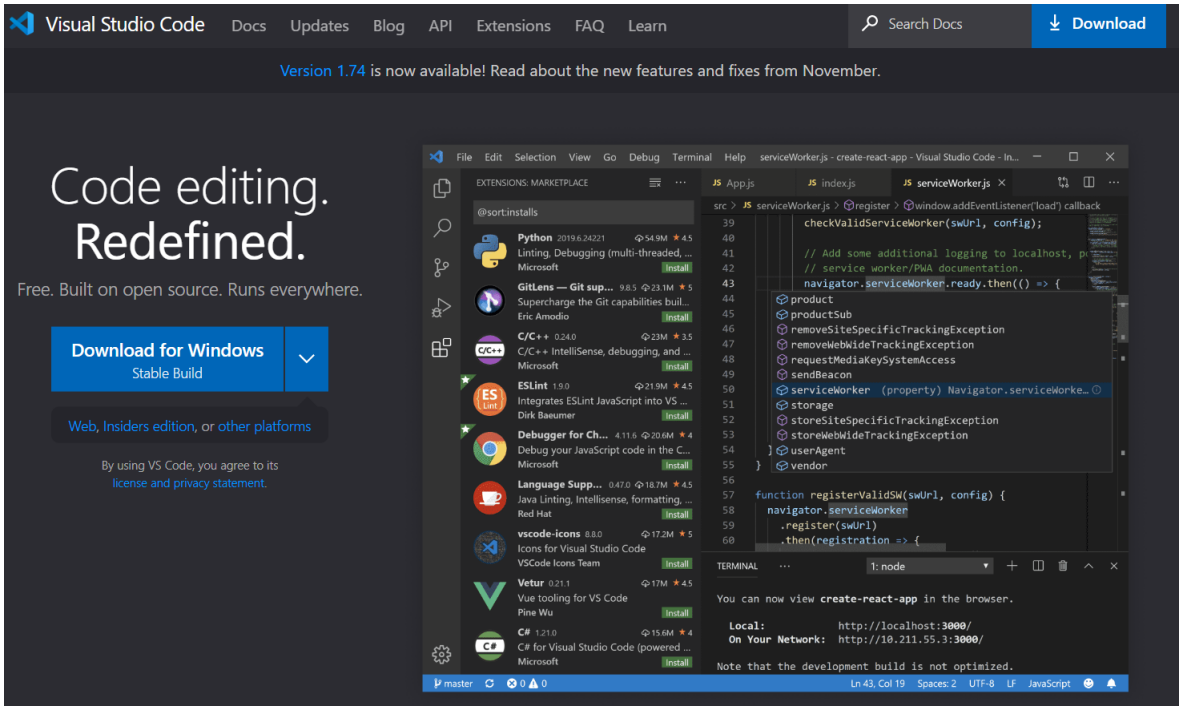

Figura 4.1: Instalación Visual Studio Code

un firmware especifico para nuestro microcontrolador. Para ello nos dirigiremos a la pagina de MicroPython `https://micropython.org/`. Como se observa en la imagen 4.4, hay una opción arriba a la derecha, *DOWNLOAD*, dicha opción nos direcciona a todos los microcontroladores compatibles con el firmware de MicroPython, entre ellos se encuentra el que necesitamos.

Una vez hallamos encontrado el microcontrolador que estamos usando, en nuestro es el ESP32-WROOM-32, para ello deberemos, descargar la primera opción, se muestra en la imagen 4.5, en este caso es muy aconsejable descargar la primera opción que se muestra en la imagen, eso se debe a que es justo la versión con la que vamos a trabajar.

Una vez tengamos descargado el firmware correspondiente, nos dirigiremos a la consola de comandos en nuestro ordenador, es muy fácil acceder a ella, solo debemos escribir *símbolo del sistema* en la barra de búsqueda de windows. Una vez la tengamos abierta se nos mostrará de esta manera, 4.6.

Para comenzar a instalar el firmware en nuestro microcontrolador debemos instalar *Python*, en el ordenador, para esto debemos ir a la página de *Python*, e instalarlo, la página de *Python*, `https://www.python.org/downloads/`, 4.7, una vez lo descarguemos, debemos instalarlo, y en un momento de la instalación nos dejará marcar una opción que indica *Añadirlo al Path*, la marcamos y continuamos con la instalación.

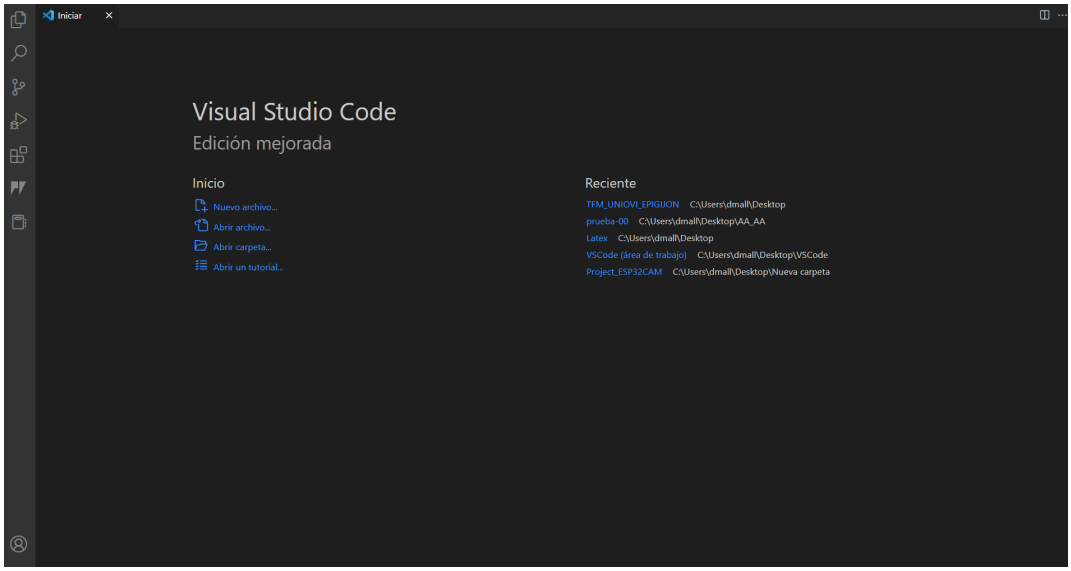

Figura 4.2: Pantalla de Inicio de VS Code

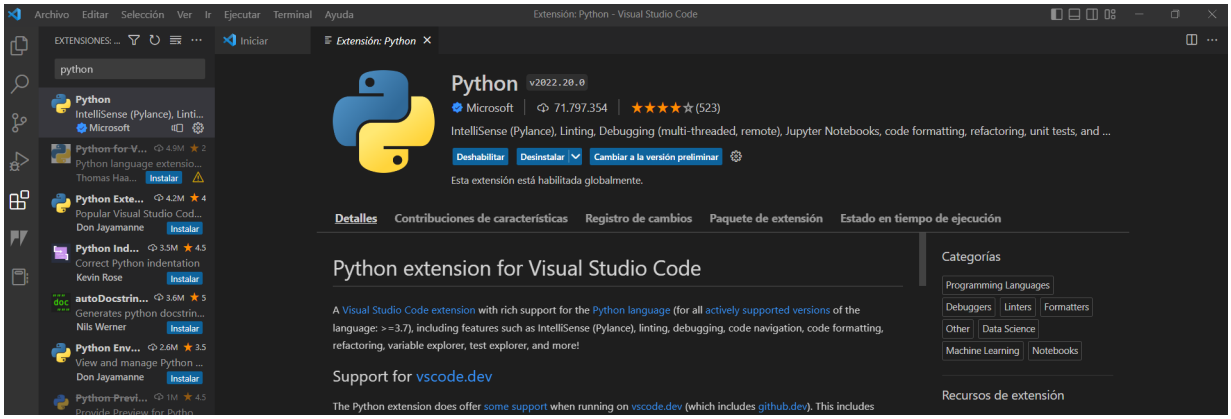

Figura 4.3: Instalación Python en VS Code

Una vez instalado debemos irnos a la consola del ordenador y debemos instalar la herramienta *esptool*, para ello debemos escribir la siguiente instrucción en la consola, *pip install esptool*, tal como se muestra en la imagen 4.8.

Como siguiente paso debemos conectar el ESP32 al ordenador, una vez realizado este paso debemos ubicar el puerto en el que se conecta el microcontrolador, se recomienda utilizar siempre el mismo puerto. Si no sabes como localizar el puerto al que esta conectado, es muy sencillo, solo debes escribir en la barra de búsquedas *Administrador de dispositivos*, una vez se nos habrá la pantalla, solo debemos ir a puertos y buscar cuál es el puerto al que está conectado el microcontrolador. (Imagen 4.9)

En nuestro caso usaremos el puerto *COM7*, en otro caso solo deberemos cambiar el nombre del puerto para seguir los siguientes pasos.

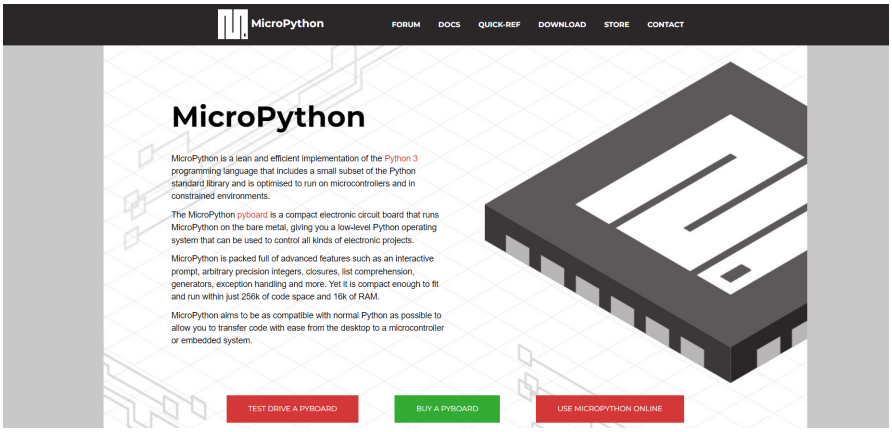

Figura 4.4: Instalación Python en VS Code

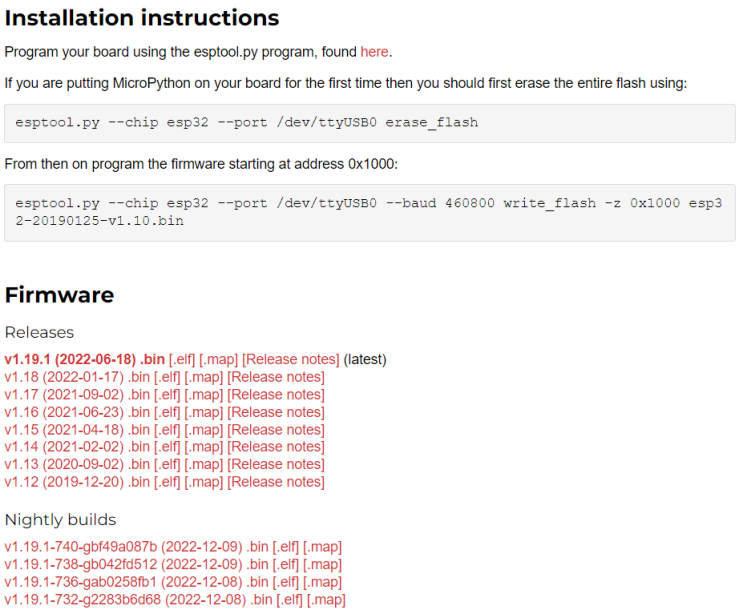

Figura 4.5: Descarga de firmware de MicroPython

Siguiendo con la instalación, nos volveremos a dirigir a la terminal de comandos y escribir el siguiente comando, que se muestra en la imagen 4.10. Una vez terminado la limpieza de la memoria del microcontrolador, pasamos al último, que es introducir el firmware en el microcontrolador, este paso solo debemos escribir en la ventana de comandos escribir la siguiente línea de código, que se muestra en la imagen 4.11, es importante de que en caso de que nos de un error de instalación es debido a que la memoria del microcontrolador ya este ocupada, es por ello que se recomienda pulsar el botón de *reset* del microcontrolador mientras se ejecuta esta instrucción.

En la imagen 4.11 se nos muestra la línea de código que debemos proceder a escribir en la ventana de comandos, esta línea de código es sencilla de entender, básicamente de estamos diciendo que debemos escribir en la memoria del microcontrolador el firmware

Figura 4.6: Pantalla Inicio Terminal

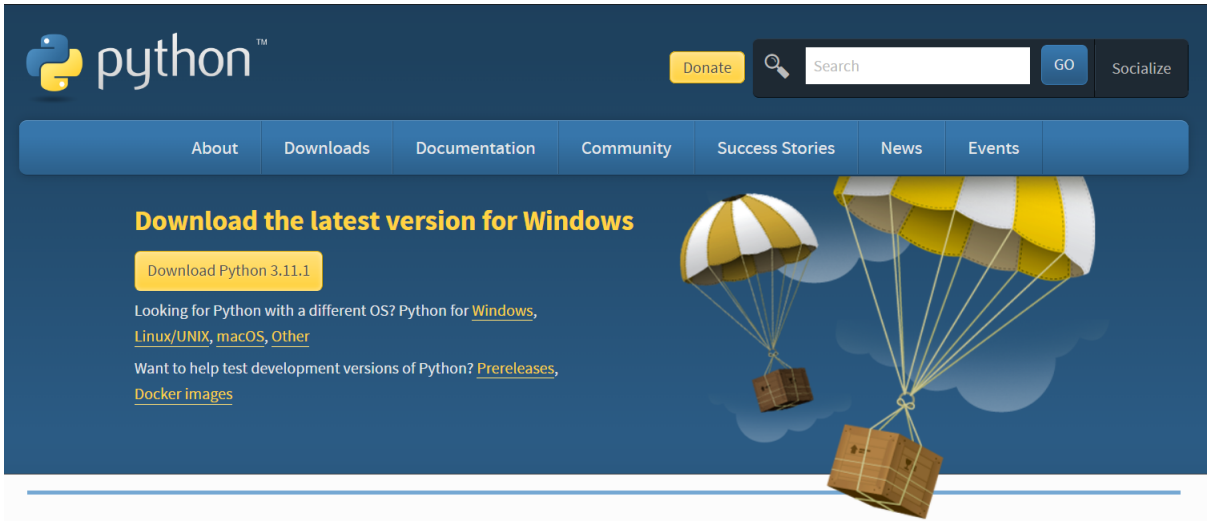

Figura 4.7: Pantalla de Descarga de Python

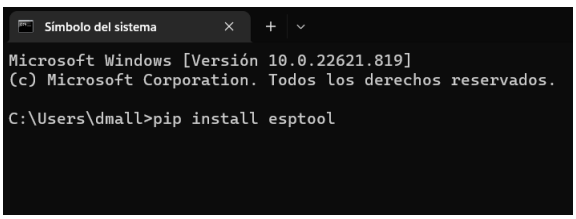

Figura 4.8: Instalación de la herramienta *esptool*

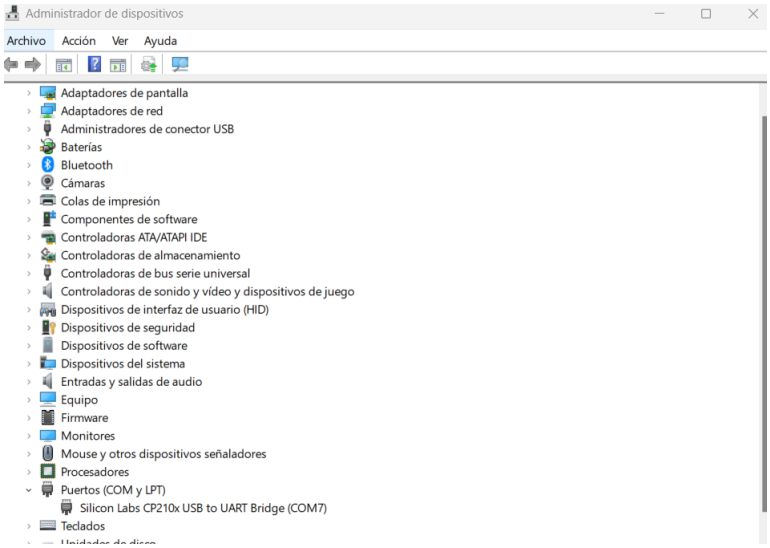

Figura 4.9: Pantalla de Administrador de dispositivos

Figura 4.10: instrucción para limpiar la memoria del microcontrolador

correspondiente que hemos instalado a una velocidad de transmisión (*baudios*) de 460800, este firmware lo escribimos en la posición *0x1000*, que se corresponde a la posición de memoria de programa de nuestro microcontrolador.

4.2.- Introducción al Visual Studio Code

Con la instalación finalizada, podemos pasar a explicar un poco el funcionamiento de como pasar nuestros programas a nuestro ESP32 (o en caso de utilizar otro microcontrolador, el que se utilice).

Antes de empezar a adentrarnos un poco en el Visual Studio Code, queremos recordar que esta no es la única manera de trabajar con MicroPython, siempre podemos utilizar otro IDEs, por ejemplo VS Code tiene *Pymakr*, que es un IDE propio que permite trabajar con este, o incluso se puede instalar un programa específico para estas tares como por ejemplo el *Thonny*.

Dejando ya de lado este pequeño corte, vamos a proceder a explicar como podemos

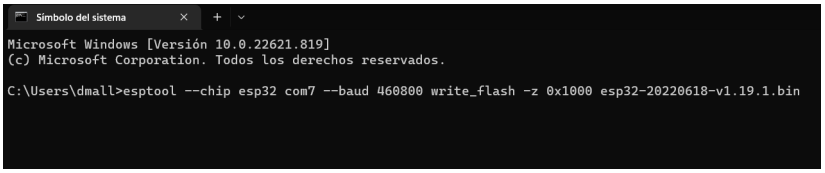

Figura 4.11: instrucción para Instalar el firmware

introducir nuestros programas en el Microcontrolador. Para ello vamos a abrir el VS Code, y abrir una carpeta donde vayamos a guardar nuestros programas.

Una vez hecho este paso vamos a escribir nuestro código correspondiente y abriremos la terminal, (se puede abrir con el atajo *Ctrl + J* o desde la propia ventana de VS Code, entonces nos quedará la ventana de tal manera como se muestra en la imagen 4.12.

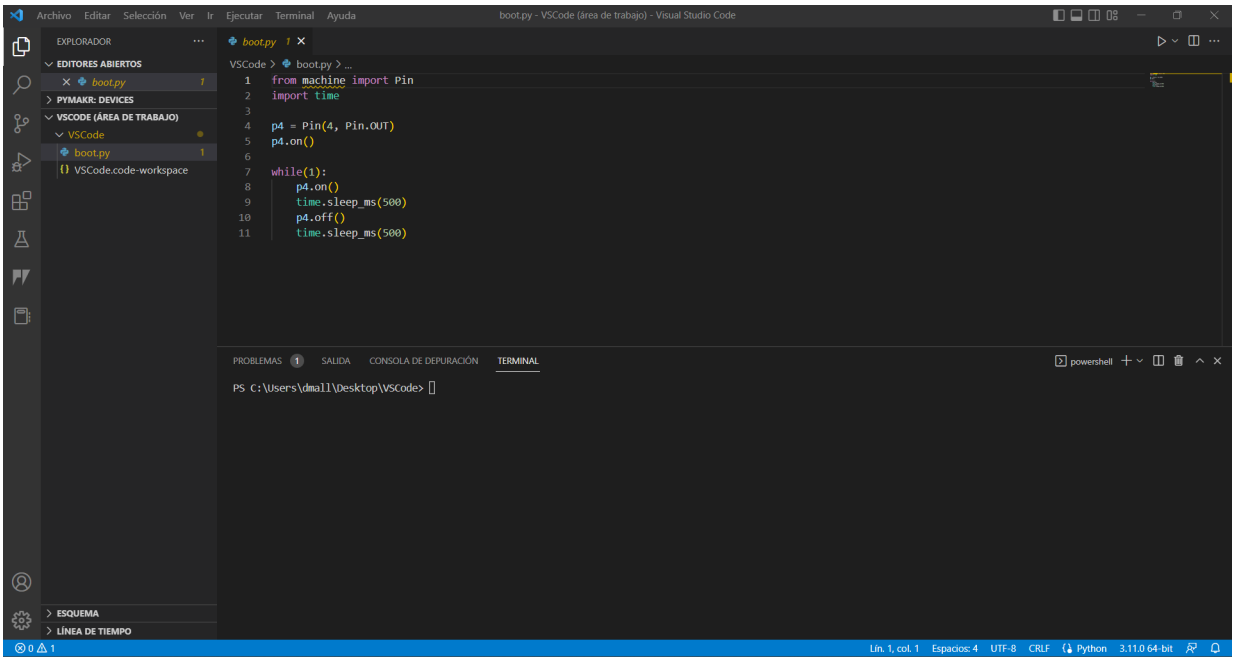

Figura 4.12: Pantalla Principal

Una vez tengamos acomodada la pantalla principal a nuestro gusto, pasaremos al siguiente paso. Este paso siguiente no está incluido en la instalación debido a que en algunos casos el propio *Python* nos lo incluye, en caso contrario deberemos proceder a realizar el siguiente paso.

Vamos a irnos a la terminal y escribir la siguiente línea de código,*pip3 install adafruit-ampy*, en la terminal de Visual Studio Code, tal como se muestra en la imagen 4.13.

Esta instalación nos permitirá pasarlo los programas, arrancarlos, eliminarlos... de nuestro microcontrolador. Una vez lo tengamos instalado, vamos a ejecutar en la terminal de VS Code, la instrucción *ampy*.

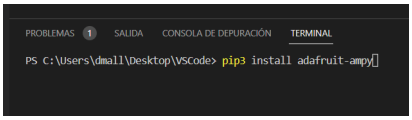

Figura 4.13: Instalación de *adafruit-ampy*

Esta instrucción nos dará una serie de instrucciones que podremos ejecutar en la terminal.

1. Options:
   a) -p, –port PORT *Name of serial port for conected board [requiered]*
   b) -b, –baud BAUD *Baud rate for the serial connection. (default 115200)*
   c) –help *Show this message and exit*
2. Commands:
   a) get, *Retrieve a file from the board*
   b) ls, *List contents of a directory on the board*
   c) put, *Put a file on the board*
   d) rm, *Remove a file from the board*
   e) run, *Run a scripts and print its output*

A través de estas instrucciones podremos trabajar con el microcontrolador, simplemente debemos ejecutar las instrucciones en la terminal y está te indicará cuando este cargado, si hay algún problema... Por ejemplo, tomando como ejemplo un programa cualquiera que tengamos, como el que se muestra en la figura 4.14, podemos observar que en la terminal estamos ejecutando la instrucción *ampy –port com7 put boot.py*, con esta instrucción estamos diciendo que en el puerto al que esta conectado nuestro microcontrolador queremos colocarle el programa *boot.py*, y en caso de querer arrancar nuestro programa solo debemos cambiar la orden *put* por *run*.

Cuando arrancamos nuestro programa, en caso de querer interrumpirlo solo debemos pulsar en la terminal *Ctrl + C* o *Ctrl + Z*.

4.3.- Thonny

Otra alternativa para cargar programas de Python, es el IDE de **Thonny**. El IDE de Thonny es el mas sencillo de manejar. En este caso, nos ofrece opciones de librerías propias, como por ejemplo, librerías de MQTT, librerías de manejo I2C ...

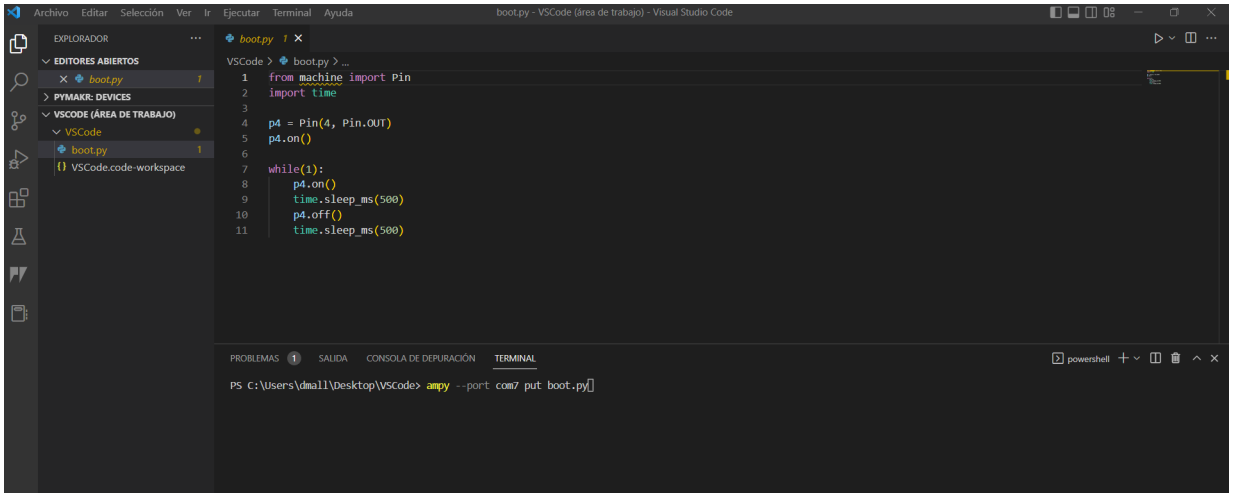

Figura 4.14: Programa de Ejemplo

4.3.1.- Instalación de Thonny

La instalación sencilla, nos dirigimos a la página de Thonny (`https://thonny.org/`), lo descargamos y ejecutamos.

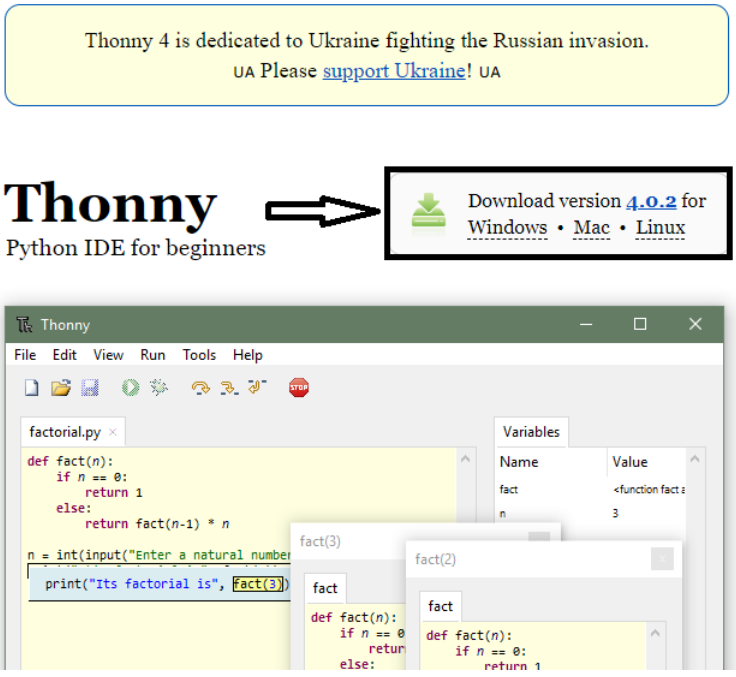

Figura 4.15: Página principal de *Thonny*

Una vez tengamos instalado el programa, ejecutaremos el programa, y abriremos la aplicación.

Seleccionaremos el idioma que se desee, y la opción Standard, en caso de disponer de

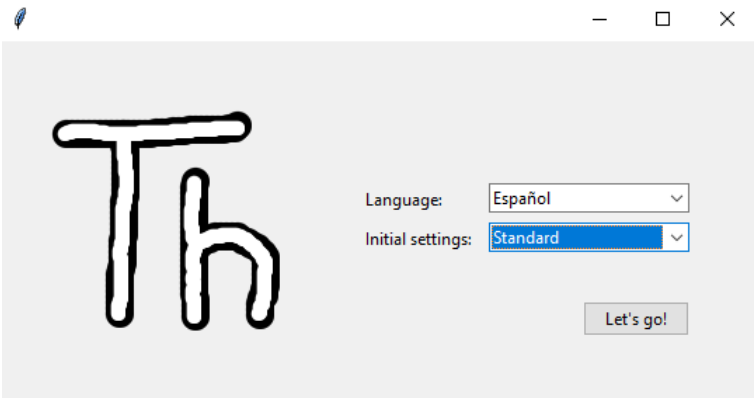

Figura 4.16: Configuración de Thonny

una Raspberry Pi, se escogerá dicha opción, en vez de la *Standard*.

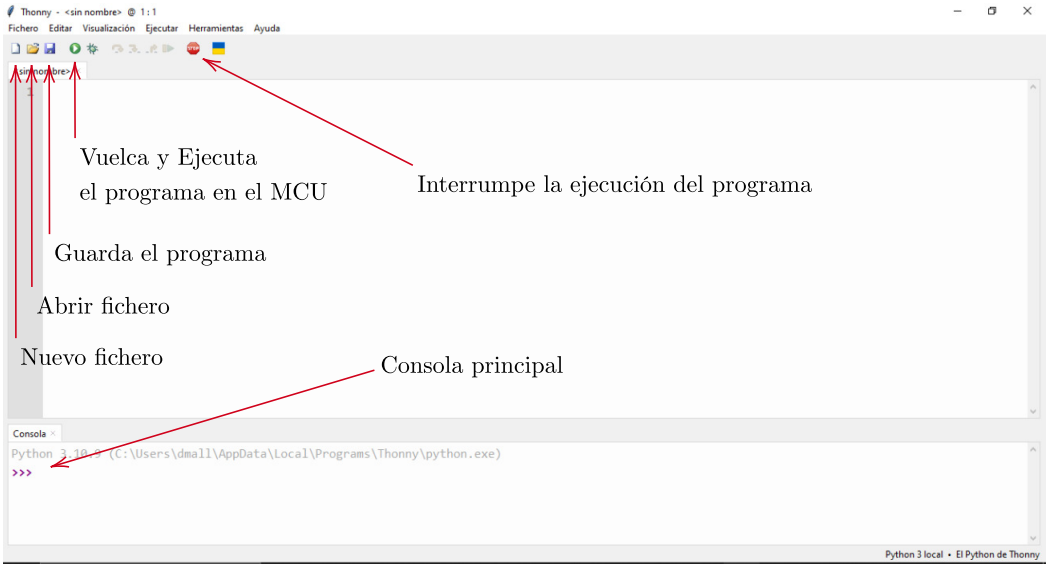

Figura 4.17: Pantalla principal de Thonny

En al figura 4.17 se nos muestra la pantalla principal con las opciones básicas para el manejo de los programas de MicroPython, más adelante nos adentraremos en otras opciones del programa.

4.4.- Instalación y configuración de la aplicación IoT MQTT Panel

Para realizar la comunicación MQTT, que veremos en el capítulo 7, vamos a instalar la aplicación **IoT MQTT Panel**, esta disponible solo para dispositivos Android, aunque se puede utilizar cualquier aplicación semejante a esta, que permita comunicarse con el protocolo MQTT.

Figura 4.18: Creación de una conexión MQTT con la aplicación IoT MQTT Panel

Partiendo de la figura 4.18, podemos ver que el uso de la aplicación en sencillo, en este caso, partiremos con la aplicación ya instalada, crearemos una nueva conexión configurándolo con los siguientes parámetros:

- *Connection name*: En este apartado introduciremos el nombre de la conexión que queramos realizar.
- *Client ID*: Indicamos un nombre de conexión para el usuario
- *Broker Web/Ip address*: Le damos la dirección IP de nuestro servidor, en este caso sera la dirección 156.35.154.170.
- *Port number*: El puerto de entrada puede ser el **1883**, para entrar por el puerto sin seguridad TLS, o por el puerto **8883**, para entrar con seguridad TLS.
- *Network protocol*: Seleccionamos la **TCP-SSL**, en este caso indicaremos el protocolo de comunicación a utilizar, en este caso, nos quedaremos con la seguridad TCP-SSL.

En el resto de las opciones no cambiaremos nada, lo dejaremos tal como estan.

5. Introduccion al Micropython

Para poder comenzar a realizar nuestros programas de control, vamos a realizar una breve introducción al *Micropython*, al final deriva en todo del Python, por lo que las nociones básicas de manejo de lista, variables, diccionarios, funciones, ... será igual que el Python, por lo que solo vamos adentrarnos en las clases[1] ([2]).

5.1.- Introducción y configuración de pines

Para el funcionamiento básico de cualquier programa deberemos disponer de la librería **machine**, esta nos dará las clases necesarias para lograr el funcionamiento de estos programas ([3]).

Como primer paso, vamos a ver como se configuran los pines en el ESP32. En este primer programa vamos a configurar el pin 3 como entrada.

1. Primero vamos a declarar la librería **machine**:

```
from machine import Pin
```

2. Seguido de esto vamos a configurar el pin a usar:

```
led = Pin(3, Pin.OUT)
```

3. Ahora vamos a ver como encender y apagar el led:

```
led.on()    # Encender el led
led.off()   # Apagar el led
```

4. Otra manera podría ser:

```
led.value(1)    # Encender el led
led.value(0)    # Apagar el led
```

5. El programa final quedaría:

```
from machine import Pin
led = Pin(3, Pin.OUT)
    # Como encender y apagar el led
```

[1] Una clase es una plantilla con la que se determina el funcionamiento de un determinado objeto, esta viene determinada con un modelo predefinido

```
led.on()
led.off()
    # Como cambiar el valor de salida del Pin
led.value(1)
```

Vemos que la clase usada en cuestión es la clase **Pin**. Aparte de para esta sencilla función, tiene configuraciones mas complejas. Si nos adentramos en la biblioteca **machine**, vemos que la forma de la clase es:

```
class machin.Pin(id,mode=-1,pull=-1,*,value=None,drive=0,alt=-1)
```

1. **id**: El pin a configurar
2. **mode**: La configuración del pin puede ser:
   - **Pin.IN**: Pin configurado como entrada
   - **Pin.OUT**: Pin configurado como salida
   - **Pin.OPEN_DRAIN**: Pin configurado para salida de drenaje abierto[2]
   - **Pin.ALT**: El pin está configurado para realizar la función alternativa de este pin, como por ejemplo la comunicación serie.
   - **Pin.ALT_OPEN_DRAIN**:
   - **Pin.ANALOG**: Configuración como pin analógico.
3. **pull**: cambiamos la resistencia del estado lógico:
   - **None**: No hay resistencia
   - **Pin.PULL_UP**: Resistencia de *pull up*[3]
   - **Pin.PULL_DOWN**: Habilitado la resistencia de *pull down*[4]
4. **value**: Solo válido para los modos **Pin.OUT** y **Pin.OPEN_DRAIN**, este permite especificar el valor de salida.
5. **drive**: Especifica la potencia de salida del pin.
6. **alt**: Especifica la función alternativa del pin.

[2]La salida del drenaje abierto funciona de la siguiente manera: si el valor de salida se estable en 0, el pin esta activo en un nivel bajo, en caso de que la salida es 1, el pin esta en un estado de alta impedancia

[3]Esta resistencia hace que cuando no hay lectura en la entrada la caída de tensión es de 5 Voltios, y en caso contrario, es de 0 Voltios

[4]Esta resistencia permite que cuando este en reposo la caída de tensión es de 0 Voltios, en caso de que haya alguna lectura, la caída de tensión es de 5 VOltios

5.1.1.- Programa propuesto

En este caso vamos a diseñar un programa que encienda y apagué el led cada 0.5 segundos.

```
from machine import Pin
    # Importamos la clase Pin para configurar los puertos
from time import sleep
    # Importar la funcion sleep de la libreria time, para dormir el MCU

led_Azul = Pin(3, Pin.OUT)
    # Configuramos el Pin 3 como salida

while True:
    led_Azul.on()
    sleep(0.5)
    led_Azul.off()
    sleep(0.5)
```

Code Listing 5.1: Programa 1: Encender y apagar un led

5.2.- ADC (Conversor analógico digital)

En la figura 3.1, se observan las posibilidades de configuración de los pines del ESP32, entre ellos vemos que existen mucho pines que nos permiten la conversión ADC. En algunos casos, vamos a tener que leer valores analógicos, como por ejemplo un potenciómetro, este realiza la lectura de una tensión diferencial.

1. Importamos el modulo **ADC**:

```
from machine import ADC
```

2. Configuramos el pin correspondiente:

```
adc = ADC(12)
    # El pin 12 es un puerto ADC
```

3. Leemos el valor de lectura del Pin:

```
val = adc.read_u16()
    # Lectura del conversor
```

En este caso la variable *val*, devuelve el valor de conversión del pin 12, este lo devuelve en un rango de 0-65536, el objeto creado con la extensión **adc.read_u16**, leer un valor analógico de rango de 2^{16}

4. El programa final sería:

```
from machine import ADC
adc = ADC(12)
    # El pin 12 es un puerto ADC
val = adc.read_u16()
    # Lectura del conversor
```

La clase *ADC*, es mas simple de configurar que la clase Pin, no tiene tantas opciones, solo depende de dos entradas:

```
class ADC(pin,*,atten)
```

En este caso tenemos:

1. **pin**: Se configura el pin que queramos que realice la lectura.
2. **atten**: Con este permitimos atenuar la tensión de referencia del conversor, es decir, cambiar los rangos de lectura, con el fin de mejorar la precisión:
   - **ADC.ATTN_0DB**: No existe ningún tipo de atenuación (100mV - 950mV).
   - **ADC.ATTN_2_5DB**: Aplica una atenuación de 2.5dB (100mV - 1250mV).
   - **ADC.ATTN_6DB**: Aplica una atenuación de 6dB (150mV - 1750mV).
   - **ADC.ATTN_11DB**: Aplica una atenuación de 11dB (150mv - 3300mV).
3. **width**: Esta instrucción nos permitirá modificar la precisión de medida, en un rango de bits:
   - **WIDTH_9BIT**: 9 bits.
   - **WIDTH_10BIT**: 10 bits.
   - **WIDTH_11BIT**: 11 bits.
   - **WIDTH_12BIT**: 12 bits.

5.2.1.- Programa propuesto

En este caso vamos a realizar la lectura de unos de los puertos de nuestro MCU, dicha tensión irá alimentada hasta 3.3 Voltios, y queremos leer dicha información con una precisión de 2^{12} bits:

```
from machine import ADC, Pin
    # Importamos la clase Pin y la clase ADC, esta # ultima para
    configura la conversion ADC
from time import sleep
    # Importamos la funcion sleep de la libreria # time, para dormir el
    MCU

adc34 = ADC(34)
    # El puerto 34, leera el valor analogico
adc34.atten(ATTN_11DB)
    # Lo configuramos con una atenuacion de 11 dB, esto nos permitira
    leer valores de hasta 3.3 VOltios
adc34.width(WIDTH_12BIT)
    # La resolucion de la lectura sera de hasta 4096 bit

while True:
    valor = adc34.read()
        # Nos guarda el valor leido por el conversor
    valor_16 = adc34.read_u16()
        # Nos devuelve el valor en un rango de 65535 bits
    sleep(0.2)
        # Leeremos el valor cada 0.2 segundos
```

Code Listing 5.2: Programa 2 : Leer una entrada analógica

5.3.- PWM (modulación de ancho de pulso)

Para manejar cualquier motor, es muy importante el módulo **PWM**. Este nos permite modelar el ancho de pulso de una señal, es decir, podemos regular el tiempo *ON/OFF* (estado alto a, estado bajo 0) de una señal de periodo T.

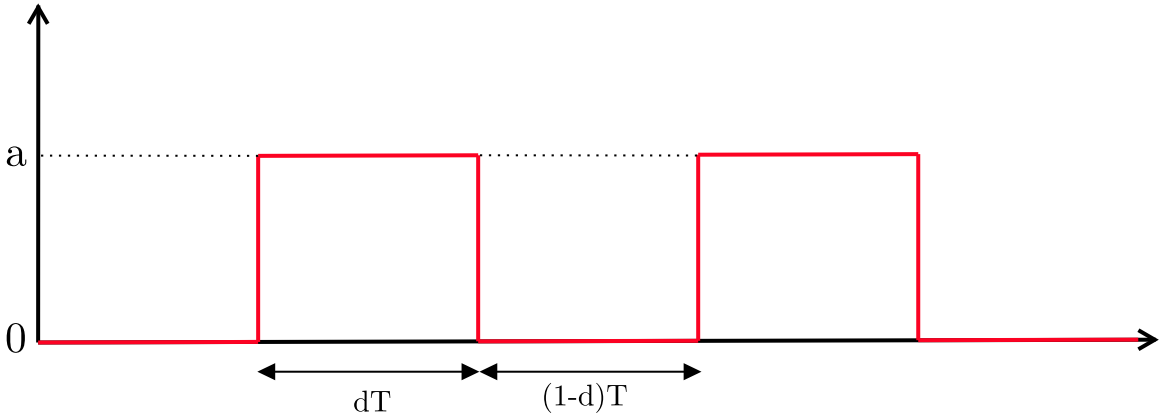

Figura 5.1: Modulación de ancho de pulso

Con la señal PWM, podemos regular el valor medio de la tensión aplicada al motor, y por lo tanto su velocidad la velocidad del motor. El parámetro a regular es el ciclo de trabajo o *duty*. El duty, como es sabido, es el tiempo que esta en estado alto la señal (a), frente al periodo completo de la señal (T).

Más adelante haremos un estudio sobre la dinámica de un circuito de potencia para el control del motor, en el estudiaremos también el ajuste de la velocidad del motor, tensión de salida, ...

En este caso vamos a comenzar con un breve programa:

1. Incluiremos las clases necesarias para este programa:

   ```
   from machine import Pin, PWM
   ```

   En este caso hemos incluido dos clases, la clase *Pin*, vista anteriormente, y la clase *PWM*.

2. Crearemos un objeto PWM de un pin:

   ```
   pwm = PWM(Pin(0), freq = 200, duty = 512)
   ```

   En este caso hemos creado un objeto *pwm*, en el configuramos el *pin 0*, como salida PWM, con una frecuencia de 200 Hz ($T = \frac{1}{200}$), y un *duty* de 512, sobre un total de 1023, estamos con una $d = 0{,}5$.

3. Como resultado final, tenemos:

```
from machine import Pin, PWM
pwm = PWM(Pin(0), freq = 200, duty = 512)
```

La clase PWM, tiene la siguiente construcción:

```
class PWM(dest, *, freq, duty_u16, duty_ns)
```

Los parámetros son:

a) **dest**: Pin de salida PWM, no todos los puertos permiten una salida PWM.

b) **freq**: Ajustamos la frecuencia de los pulsos a mandar (figura 5.1)

c) **duty_u16**: Ajustamos el *duty* en un rango de 0 a 2^{16} .

d) **duty_ns**: Configuramos el ancho del pulso en un rango de nanosegundos.

5.3.1.- Programa propuesto

En este caso realizaremos un programa en el que leeremos la lectura de un potenciómetro y la salida será el ancho del PWM, es decir, en función de la lectura del conversor, modificaremos el ancho del pulso PWM.

```
from machine import ADC, Pin, PWM
    # Importamos la clase Pin, la clase ADC y la clase PWM, esta para configurar la salida los pulsos PWM
from time import sleep
    # Importamos la funcion sleep de la libreria # time, para dormir el MCU

adc34 = ADC(34)
    # El puerto 34, leera el valor analogico
adc34.atten(ATTN_11DB)
    # Lo configuramos con una atenuacion de 11 dB, esto nos permitira leer valores de hasta 3.3 VOltios
adc34.width(WIDTH_10BIT)
    # La resolucion de la lectura sera de hasta 1023 bit

pwm = PWM(Pin(0))
    # Condifuramos el pin 0 como salida PWM
pwm.freq(200)
    # Lo configuramos a una frecuencia de 200 Hz
pwm.duty(0)
    # Le ponemos como duty inicial 0

```

```
while True:
    valor = adc34.read()
        # Leemos el valor del conversor
    pwm.duty(valor)
        # Aplicamos dicho valor al duty
    sleep(0.2)
        # Repetimos la secuencia cada 0.2 segundos
```

Code Listing 5.3: Programa 3 : Modificar el pulso PWM

5.4.- Interrupcion externa

Al igual que todos los microcontroladores, una de las características mas importantes que tiene son las *interrupciones*, estas nos permitirán realizar cualquier cambio en nuestra secuencia de bucle infinito, a través de una acción externa o interna, como por ejemplo cambiar el sentido de giro de un motor, encender y apagar un led, etc.

El método de configuración de estas interrupciones externas, se realiza de la siguiente manera:

1. Definimos la entrada donde se leerá la interrupción:

```
pin = Pin(34, Pin.IN)
```

2. Definimos el **IRQ**, con este definiremos el método de disparo de la interrupción.

   - **Trigger**: Configuraremos el flanco de disparo de la interrupción.
     - **IRQ_FALLING**: Disparo por flanco de bajada.
     - **IRQ_RISING**: Disparo por flanco de subida
   - **handler**: En esta definiremos la función interrupción[5].
   - **priority**: Define el nivel de prioridad de las interrupciones.

[5]Función encargada de realizar el programa de interrupción

5.4.1.- Programa propuesto

En este caso realizaremos un programa sencillo, vamos a encender y apagar un led mediante un pulsador externo, este será el encargado de ejecutar la interrupción.

```
from machine import Pin
    # De la libreria machine importaremos solamente la clase Pin
from time import sleep
    # De la clase time importaremos la funcion sleep

led = Pin(0, Pin.OUT)

# Funcion interrupcion
def funcion_interrupcion():
    sleep(0.02)
        # Evita el antirebote
    if (led.value() == 1):
        led.off

    else:
        led.on

boton = Pin(4, Pin.IN)
    # El boton estara conectado al puerto 0
boton.irq(trigger = IRQ_FALLING, handler = funcion_interrupcion)
    # Configuramos la interrupcion para que salte por flanco de bajada, y
     ejecute la funcion correspondiente

while True:
    # Vacio
```

Code Listing 5.4: Programa 4 : Apagar y encender un led mediante interrupción externa

5.5.- Interrupción por timer

Como hemos visto en la sección 5.4, podemos disponer de interrupciones externas, y configurarlas de varias maneras, aunque también disponemos de interrupciones internas, en este caso, la interrupción por timer. Esta interrupción consiste en que se ejecute una función interrupción, según un periodo, configurable.

Para la configuración de esta interrupción vamos a disponer de la librería **Timer**, esta se encuentra dentro de la librería **machine**.

Por lo que comenzaríamos de la siguiente manera:

1. Importamos el modulo Timer:

```
from machine import Timer
```

2. Para realizar una interrupción, necesitaremos de una función de interrupción que ejecute la tarea correspondiente a esta:

```
def mycallback(tim):
    """
    Operaciones que queremos que se ejecuten
    """
```

3. Una vez definida la función interrupción, vamos a configurar esta interrupción, de la siguiente manera:

```
tim = Timer(0)
tim.init(mode = Timer.PERIODIC, freq = 1000, callback = mycallback)
```

   En este caso, creamos el objeto *tim*, al cual se le asigna la clase *Timer*, dentro de esta se le puede asignar 0 o 1, que son los timers que contiene nuestro ESP32. Para este caso la configuración a realizar, es con una frecuencia de 1KHz, que la interrupción se ejecute de manera periódica, y que responda con la función *mycallback()*

Una vez ya visto un ejemplo, de como sería la secuencia a seguir para configurarlo, vamos a disponer a analizar y a mostrar las posibles configuraciones de este:

- **mode**: Le pasaremos el modo en el que se activa la interrupción
  - **Timer.ONE_SHOT**: El temporizador se ejecuta una vez hasta que expira el período configurado del canal
  - **Timer.PERIODIC**: El temporizador se ejecuta de manera periódica, con la frecuencia o tiempo que se la ha sido determinado.
- **freq**: Le pasaremos, en Hz, la frecuencia del temporizador.
- **period**: Le pasaremos, en ms, cuanto tiempo debe haber de temporización
- **callback**: Le asignaremos la función de temporización

5.5.1.- Programa propuesto

Vamos a realizar un programa que muestre por pantalla la lectura de un potenciómetro, con una frecuencia de 100 Hz:

```
from machine import ADC, Pin, Timer
    # Importamos los modulos ADC, Pin y Timer para la realizacion del programa

adc34 = ADC(34)
    # Asignamos el pin 34, como puerto de lectura del potenciometro
adc34.atten(ATTN_11DB)
    # Configuramos la conversion A/D, con una atenuacion de 11 dB

tim = Timer(0)
    # Vamos a crear un objeto, con la clase Timer

def temporizacion(tim):
    global valor
    valor = adc34.read()
    print(valor)
    # Creamos una funcion, que nos lea el valor del potenciometro cada vez que se
    llame a la funcion

tim.init(mode = Timer.PERIODIC, freq = 100, callback = temporizacion)

while True:
    """
    Vacio
    """
```

Code Listing 5.5: Programa 5: Lectura de un potenciómetro cada 100 Hz

5.6.- Conexión Wifi

Tal y como hemos descrito en el capítulo 3, tiene un módulo Wifi, que nos permite la conexión vía Internet, lo cual nos permitirá conectar con otros dispositivos, que es uno de los valores de este proyecto.

Para ello debemos disponer de la librería **network**, es una librería sencilla de manejar, que permitirá al MCU conectarse a internet. Para ello vamos a seguir los siguiente pasos:

1. Como primer paso importaremos la librería en cuestión:

```
import network
import time
```

2. Como segundo paso, debemos guardar los valores de *SSID* y la *PASSWORD*, estas son el nombre y la contraseña de la red Wifi, a la que nos conectaremos:

```
ssid = 'red_wifi'
password = '123456'
```

3. En el tercer paso, ya empezaremos a depender de las funciones propias de la librería *network*:

```
red = network.WLAN(network.STA_IF)
red.active(True)
red.connect(ssid, password)
```

   La variable *red*, será la que controlará la conexión de con la red Wifi. En la primera línea le indicaremos que la conexión con la red es **STA_IF**, esto indica que la interfaz de conexión con la red Wifi, es de tipo *STATION*. En la siguiente línea, le diremos que se active el modulo Wifi, y la acompañaremos con la siguiente línea, la cuál nos indica que nos conectaremos a la red que le indicamos previamente, con su respectiva contraseña.

4. Como último paso, le añadiremos un bucle de conexión:

```
while not red.isconnected():
    pass
```

   Aquí indicamos que mientras no se conecte, lo siga intentando, hasta que consiga lograrlo, aunque este bucle, se puede sustituir por un delay de valor un poco fuerte, como de medio segundo o un segundo.

6. Motores Eléctricos

En este capítulo vamos a describir el funcionamiento, de forma breve, el funcionamiento de los motores eléctricos utilizados en este trabajo.

Los motores eléctricos son dispositivos que convierten la energía eléctrica en energía mecánica para producir movimiento. Existen varios tipos de motores eléctricos, siendo los más comunes los motores de corriente continua (CC), brushless y paso a paso.

Los motores de corriente continua funcionan mediante la fuerza electromagnética generada por la interacción entre un imán permanente y un imán rotativo llamado rotor. Estos motores son los que tienen un control mucho más sencillo.

Los motores Brushless, también conocidos como motores sin escobillas, utilizan imanes permanentes en el rotor y bobinas en el estator para generar movimiento. La principal ventaja que presentan estos motores es que no tienen escobillas, como su propio nombre indica, y por lo tanto no tienen ningún elemento de rozamiento. Estos motores son más eficientes y duraderos que los motores de corriente continua, y se utilizan comúnmente en aplicaciones que requieren alta velocidad y capacidad de respuesta.

Por último, los motores paso a paso son motores eléctricos que convierten señales electrónicas en movimientos discretos del rotor. Estos motores funcionan mediante la activación secuencial de bobinas electromagnéticas en el estator, lo que permite un control preciso del posicionamiento del rotor. Estos motores son utilizados en máquinas de impresión, robótica y otros sistemas que requieren un control preciso de la posición.

En resumen, los motores eléctricos de corriente continua, Brushless y paso a paso son componentes esenciales en una amplia gama de aplicaciones industriales y comerciales. Cada tipo de motor tiene características y ventajas específicas, lo que los hace adecuados para diferentes usos y necesidades.

6.1.- Motor DC

El motor de continua, como hemos visto anteriormente, es un motor que su principal característica es la facilidad en el control.

Para poder realizar un control de un Motor DC vamos a analizar como es el funcionamiento del mismo, y el como modelarlo, para su posterior simulación, diseño de PCB, y de

su respectivo programa de control.

Antes de comenzar a entrar en los ensayos del motor y el modelado, vamos a introducir el propio motor ([4]).

Figura 6.1: Motor DC (MAXON Motores)

En la figura 6.1 se muestra un típico motor de continua con escobillas, de la marca MAXON, tenemos que nuestro motor esta compuesto, de dos partes fundamentales, el **rotor** y el **estator**.

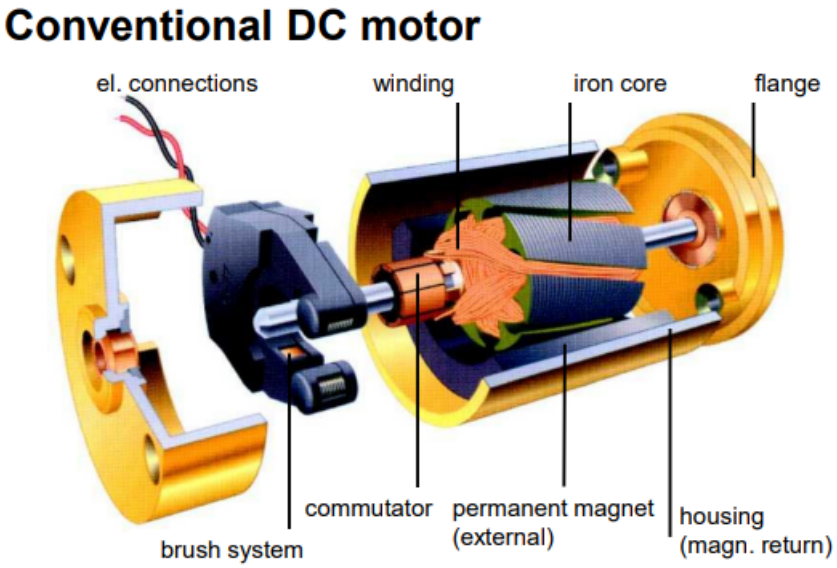

Figura 6.2: Sección de un motor DC con escobillas (MAXON Motores)

Como se observa en la figura 6.2 (despiece del motor), la parte móvil del motor es el rotor, y la parte fija el estator. El principio de funcionamiento por el que se basan, establece que cuando circula corriente eléctrica a través de los devanados del estator, genera un campo

magnético que tiende a desplazar el rotor en sentido horario o antihorario, dependiendo de la dirección del flujo magnético.

6.1.1.- Ecuaciones del Motor DC

El circuito equivalente del motor DC, sería el mostrado en la figura 6.3, en esta se muestra como la tensión **Va**, es la tensión de entrada del motor.

El motor como en cualquier caso estará sujeto a una carga que deberá mover. En función a la carga que mueva, veremos como la corriente consumida cambia, y como evolucionará de manera distinta estando en vació, sin carga, y como necesitará de mas corriente para mover una carga.

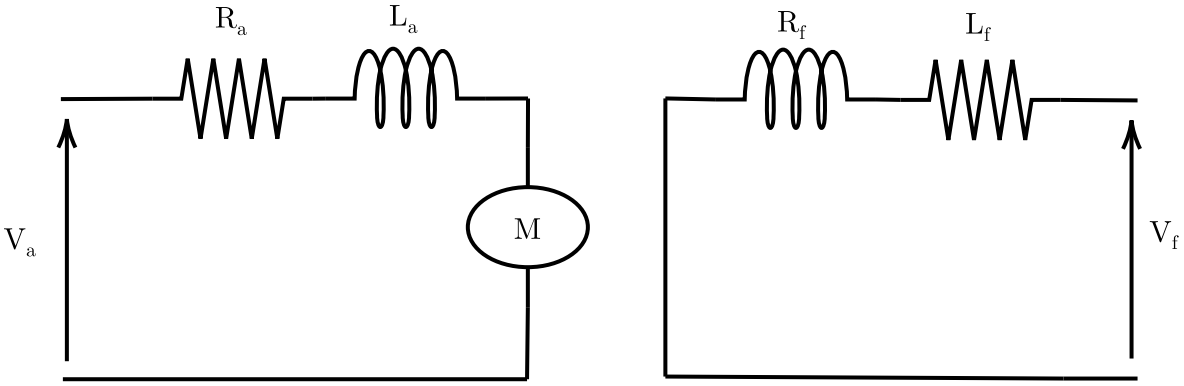

Figura 6.3: Esquema eléctrica del motor DC

Entonces a partir de este primer esquema (6.3), vamos a sacar las correspondientes ecuaciones:

$$\begin{aligned} V_a &= R_a \cdot I_a + L_a \cdot \frac{dI_a}{dt} + E_g \\ V_f &= R_f \cdot I_f + L_f \cdot \frac{dI_f}{dt} \\ E_g &= K_v \cdot w \cdot I_f \\ T_d &= K_t \cdot I_a \cdot I_f \\ P_d &= \omega \cdot T_d \end{aligned} \tag{6.1}$$

6.1.2.- Motor utilizado en el trabajo: Obtención de los parámetros del Motor DC

En este caso, como no disponemos del datasheet, ni de ninguna otra referencia del motor, vamos a realizar los siguientes ensayos del motor, para así poder, obtener los datos pertinentes y poder realizar las simulaciones correspondientes.

Para obtener los parámetro del motor, hemos realizado los siguientes ensayos: Ensayo de vacío, medida de la resistencia y de la impedancia y obtención de la constante de velocidad.

A continuación se describirán los ensayos realizados, para la obtención de dichos valores.

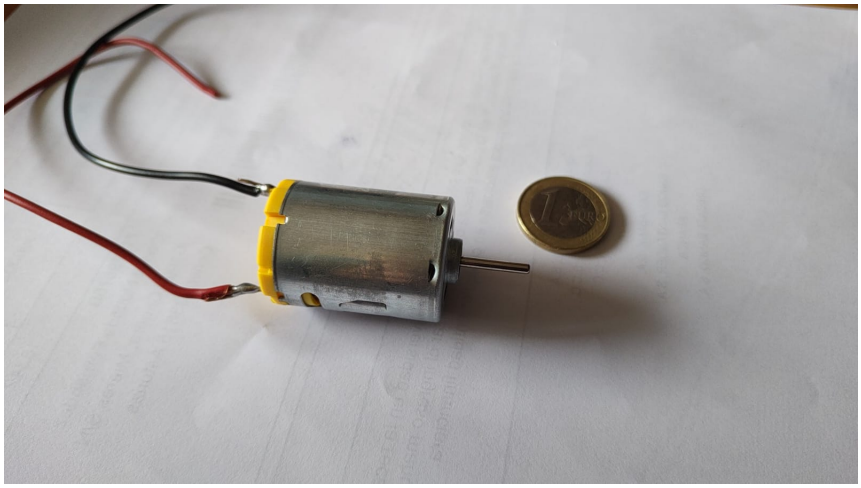

Figura 6.4: Motor de continua utilizado en el trabajo

6.1.2.1.- Ensayo de Vacío

Para realizar este ensayo, vamos a disponer del motor funcionando sin ninguna carga, es decir, el motor se encontrará en pleno funcionamiento a diferentes tensiones sin ninguna carga que suponga un consumo de corriente excesivo.

Para este ensayo disponemos de una **fuente de tensión** conectada directamente a los bornes del motor, de manera que le induciremos la tensión directamente al motor.

Tensión	Corriente
1	0.4
2	0.48
3	0.56
4	0.6
5	0.69
6	0.7
7.2	0.89

Cuadro 6.1: Datos obtenidos en el ensayo de vacío

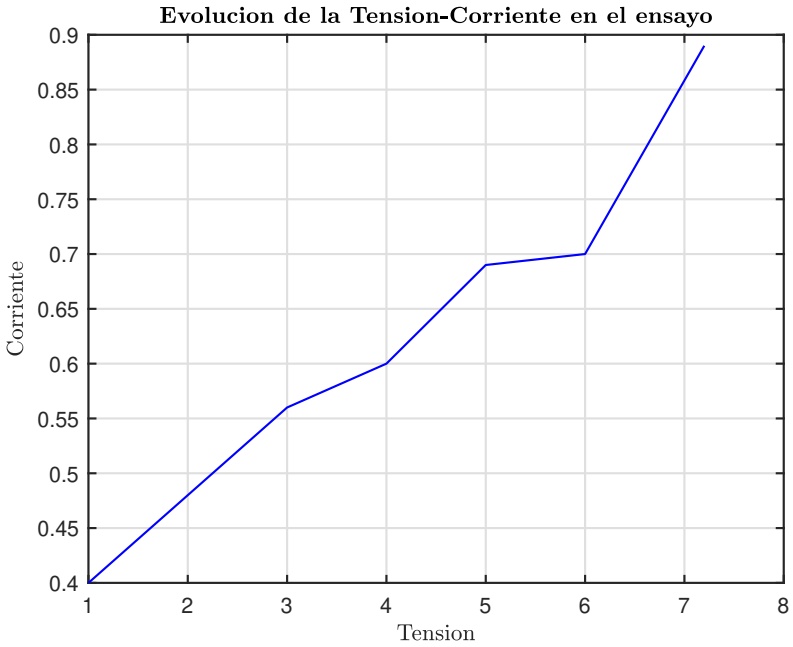

Figura 6.5: Evolución de la Tensión-Corriente en el ensayo

6.1.2.2.- Medida de la resistencia y la impedancia interna

Para la medida de la R_a y la L_a del motor, disponemos de un **analizador de impedancias**. El dispositivo en cuestión nos permite realizar un barrido de frecuencias para caracterizar el comportamiento bajo estas condiciones, en este caso el motor. Es capaz de medir valores de capacidad, inductancia y de resistencia.

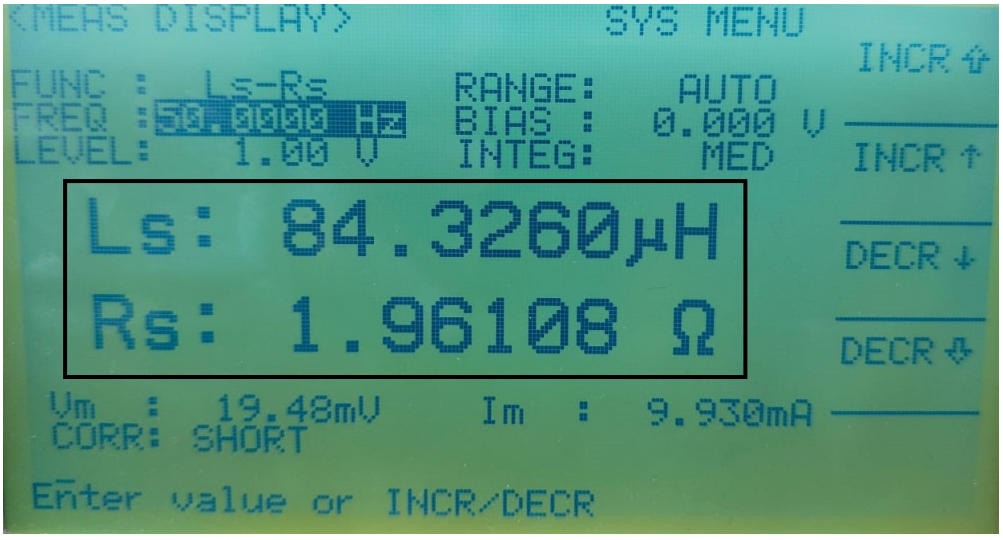

Figura 6.6: Resultados del analizador de impedancia (medida realizada a 50 Hz)

En la figura 6.6, se nos muestra los valores obtenidos, $L_a = 84{,}3260\ \mu H$ y una $R_a = 1{,}9611\ \Omega$.

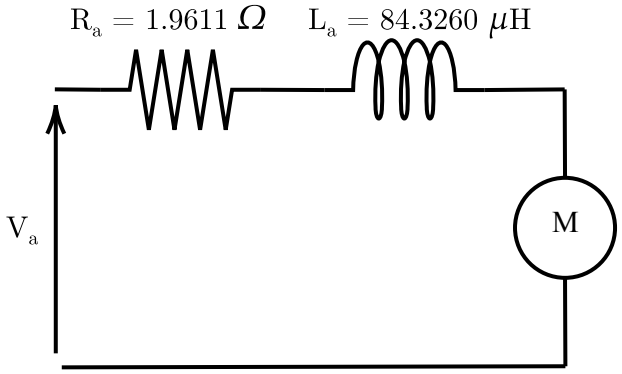

Figura 6.7: Circuito de entrada

6.1.2.3.- Obtención de la constante de velocidad

En este caso, hallamos la constante de velocidad construyendo un circuito que nos mida dicha velocidad, el montaje es el siguiente:

En el extremo del motor, hemos incorporado una carga con dos tornillos, en forma de hélice, situado para equilibrar la carga.

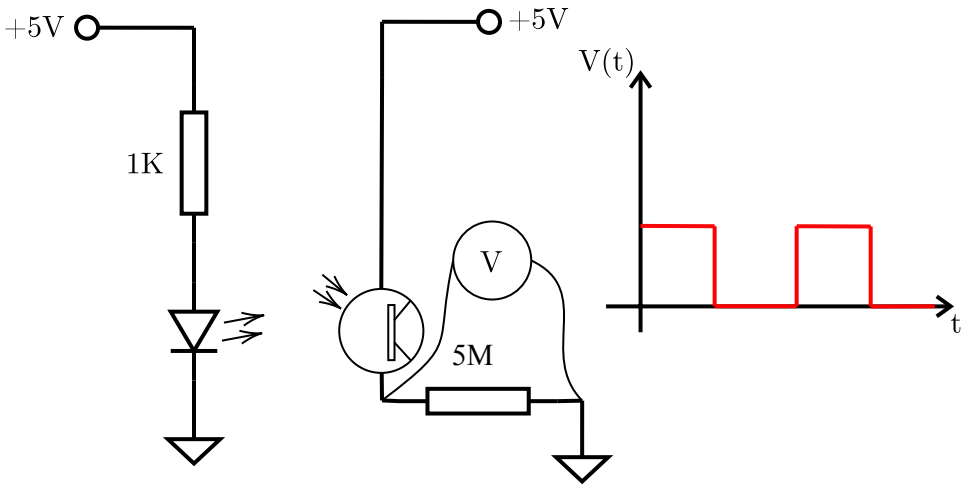

Figura 6.8: Circuito de medida de velocidad

En la figura 6.8 se nos muestra el circuito montado para esta medición, en la parte izquierda tenemos un emisor de luz, el led, y en la parte derecha tenemos un **fototransistor**, este funciona como un diodo, siendo su ánodo la parte superior, y el cátodo, la parte inferior, entonces se encontraría en polarización inversa[1], como esta corriente es muy baja, ponemos una resistencia bastante grande, de manera que compensamos esta corriente tan baja.

Entonces si realizamos la medición en extremos de la resistencia de 5 $M\Omega$ tenemos que cuando recibe la luz, la tensión será positiva, de aproximadamente 4 Voltios, mientras que

[1] Ocurre cuando una tensión negativa en bornes del diodo, hace pasar al corriente en sentido negativa.

en caso de que se intercepte la recepción de luz, este tendrá tensión cero.

Con este planteamiento, tenemos una secuencia de pulsos (6.8) a una determinada frecuencia, dicha frecuencia será la velocidad de giro del motor.

Tensión (V)	**Corriente** (A)	**Frecuencia** (Hz)
1	0.45	40
2	0.7	96
3	1	157
4	1.4	214
5	2	268.5

Cuadro 6.2: Datos obtenidos en el ensayo de velocidad

En este caso los datos obtenidos son los mostrados en la 6.2. Si hacemos una comparación con respecto a los datos obtenidos en el ensayo de vacío, 6.1, el consumos de corriente es mayor, esto es debido a que al tener una carga, se precisa de un mayor consumo de corriente para poder compensar el par.

Sabiendo que la tensión nominal del motor es la de 5 Voltios, entonces obtenemos las revoluciones por minuto.

$$\begin{aligned} F_{req} &= 268{,}5\ Hz \\ \omega &= 268{,}5\ \frac{vueltas}{s} \\ \omega &= 16110\ rpm \end{aligned} \tag{6.2}$$

6.1.2.4.- Resultados

Como resultado final del análisis del motor, tenemos los parámetros del motor descritos en la siguiente tabla:

Parámetro	Valor
R_a	1.9611 Ω
L_a	84.3260 μH
R_f	12 Ω
L_f	0.02 μH
Momento de inercia	2.18 gcm^2
V_t (nominal)	5 V
n	16110 rpm
I_f	1 A

Cuadro 6.3: Datos del motor DC

6.1.3.- Funcionamiento del motor de continua: Programa de Control en MicroPython

A continuación vamos a describir las condiciones de funcionamiento del motor de continua.

El programa de control permitirá funcionar al mismo en dos cuadrantes. En la figura 6.9 se muestra la máquina de estados del funcionamiento del motor, tal y como se puede observar. El motor tendrá un arranque y frenado suave, y permitirá un giro a izquierda y a derecha.

En este caso vamos a realizar el programa en función al esquema representado en la figura 6.9, este programa tiene la función de hacer que el motor en sentido antihorario y horario. Partimos de una máquina de estados[2], que consta de 3 estados:

- **Estado 0**: Este se dará cuando iniciemos el programa, en este estado el motor se encontrará parado. De este estado podemos partir a accionar el motor para *girar a derecha* o para accionar el motor para que *girar a izquierda*.
- **Estado 1**: Para llegar a este estado, pasaremos por un *arranque suave*, este consistirá en arrancarlo de manera progresiva. Este consistirá en cada 0.01s aumentará el ancho del pulso PWM.

```
for i in range(0,712,1):
```

[2]modelo de comportamiento basado en entrada y salidas, en el que las salidas son dependientes de las entradas

```
IN1.duty(i) # Salida del duty
sleep(0.01) # Parada de 0.01 segundos
```

En esta sección de código se observa, que aplicamos cada 0.01 segundos, un duty de valor *i*, de esta manera no aplicaremos de golpe la tensión seleccionada.

El principio de este tipo de arranque es la protección del motor, aumentar la vida útil del mismo y protegerlo contra picos de corriente.

- **Estado 2**: Al igual que el estado 1, realizaremos lo mismo. Tanto para este estado y el *Estado 1*, tomaremos en cuenta que podemos desplazarnos entre cualquier estado, es decir, desde el *Estado 0*, podremos ir al *Estado 1* y al *Estado 2*, desde el *Estado 1*, podremos ir al *Estado 0* y al *Estado 2*, y desde el *Estado 2*, podremos ir al *Estado 1* y al *Estado 0*.

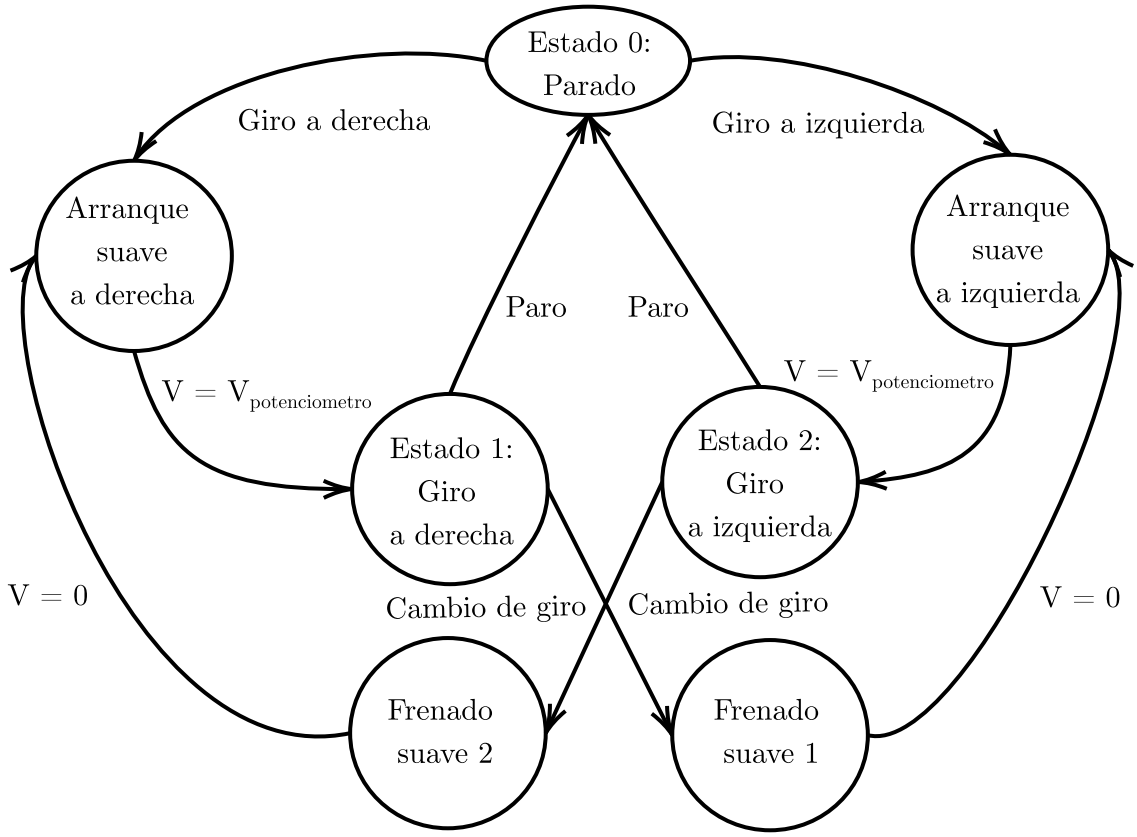

Figura 6.9: Máquina de estados del programa 1

La solución del programa se encuentra en la página pública de GitHub: Programa de control del motor DC

6.2.- Motor Paso a Paso

A un nivel interno del motor, el motor está compuesto de un estator y un rotor, una parte fija, y otra móvil, en este caso, el rotor esta dentado, se asemeja a un piñón. Por parte del estator, estamos contando con bobinados alimentados uno continuación de otro, por lo que cuando se induce uno, se produce un campo electromagnético entre ambos que permite atraer el piñón hacia dicho campo.

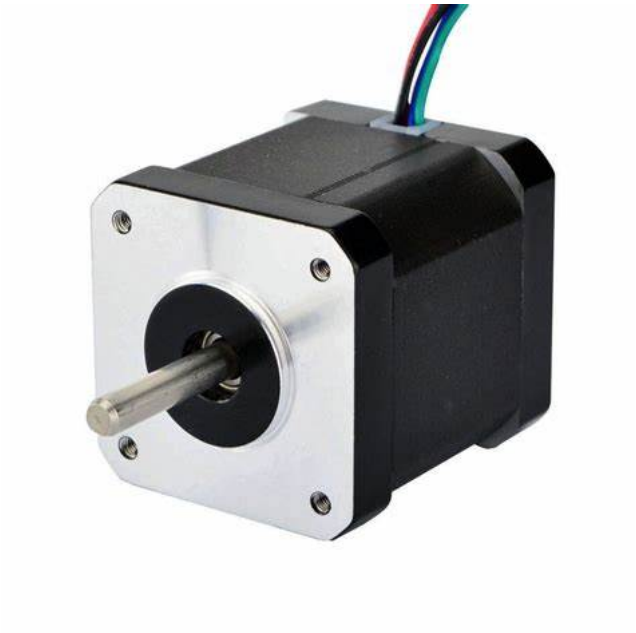

Figura 6.10: Motor Paso a Paso (MAXON Motores)

En la figura 6.11 vemos como está compuesto el bobinado interno del motor, el esquema del motor mostrado, equivale a un motor Paso a Paso **bipolar**, esto quiere decir que la tensiones de salida y de entrada de la bobina con controlables, a diferencia que en el **unipolar**, los bobinados internos tiene un punto común ([5]).

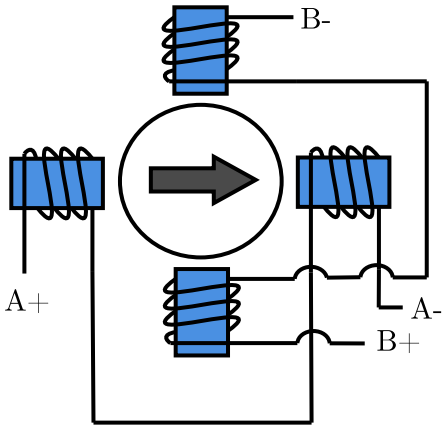

Figura 6.11: Funcionamiento del Motor Paso a Paso

Entonces, analizando el funcionamiento interno, y completándolo con un microcontrolador, tenemos como resultado el mostrado en la figura 6.12. A nivel lógico, tenemos como resultado, la tabla 6.5, en ella se muestra la secuencia de pasos a seguir para lograr el giro de manera continua, por tanto la secuencia de giro sería, paso 1, 2, 3, 4, 1, 2, 3, ... y en caso de hacerlo girar en sentido inverso, realizaríamos la secuencia en sentido inverso.

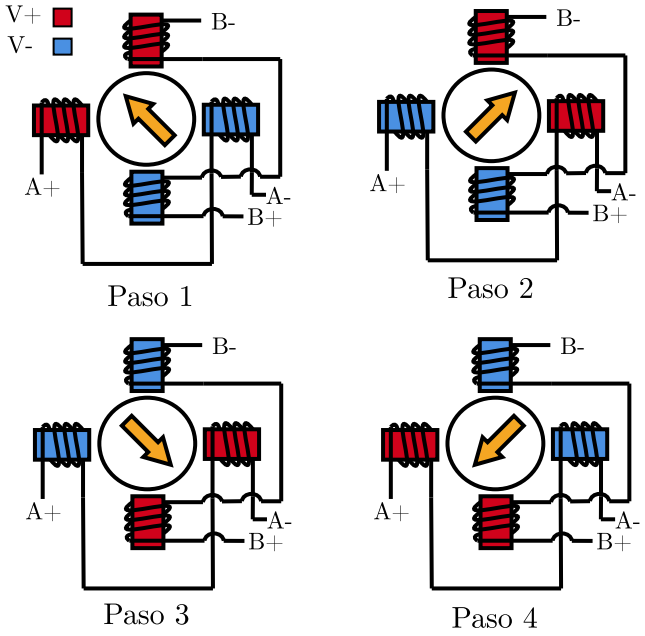

Figura 6.12: Evolución del giro del Motor con paso completo

En este caso, mostramos como hacemos el giro con la técnica de *paso completo*. A nivel físico, el motor tendría 4 cables, cada uno correspondiente a estos terminales, pues bueno, con la combinación de estas entradas, en un orden secuencial, tendríamos el giro del motor.

Paso	A+	A-	B+	B-
1	ON	OFF	OFF	ON
2	OFF	ON	OFF	ON
3	OFF	ON	ON	OFF
4	ON	OFF	ON	OFF

Cuadro 6.4: Secuencia de Paso Completo

6.2.1.- Parámetros del Motor Paso a Paso

En la sección anterior, hemos hablado de como es la estructura interna del motor en cuestión, entonces, a través de un **analizador de impedancias**, como el utilizado en el capítulo 6.1. Entonces tenemos 4 cables de entrada, por lo que si encontramos los que estén conectados a la misma bobina, cada uno a cada extremo, tenemos la medida de la impedancia interna.

Frecuencia	Inductancia	Resistencia
0 Hz	0 mH	4,5 Ω
20 Hz	16,8 mH	4,0 Ω
50 Hz	17,8 mH	5,5 Ω
100 Hz	22,29 mH	8,9 Ω
1 KHz	9,4 mH	12,0 Ω
2 KHz	8,7 mH	28,3 Ω

Cuadro 6.5: Medidas del analizadar de impedancias

El ensayo en cuestión, consiste en ir aplicando una tensión sinusoidal, de amplitud constante a 1 Voltio, y variando la frecuencia desde 0 hasta 2 KHz.

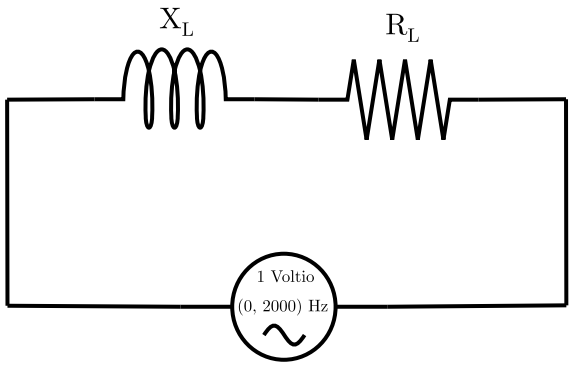

Figura 6.13: Circuito equivalente para la medición de la inductancia interna del Motor Paso a Paso

6.2.2.- Funcionamiento del motor Paso a Paso: Programa de control en MicroPython

En este caso el control es muy sencillo, solo buscamos una velocidad de giro, con la condición de que se pueda cambiar el sentido de giro.

La velocidad de giro se realiza consiguiendo una evolución de la secuencia de pasos, de igual manera que se muestra en la tabla 6.5, de igual manera si queremos realizar un cambio de sentido, debemos realizar en sentido inverso al que hemos hecho con el anterior.

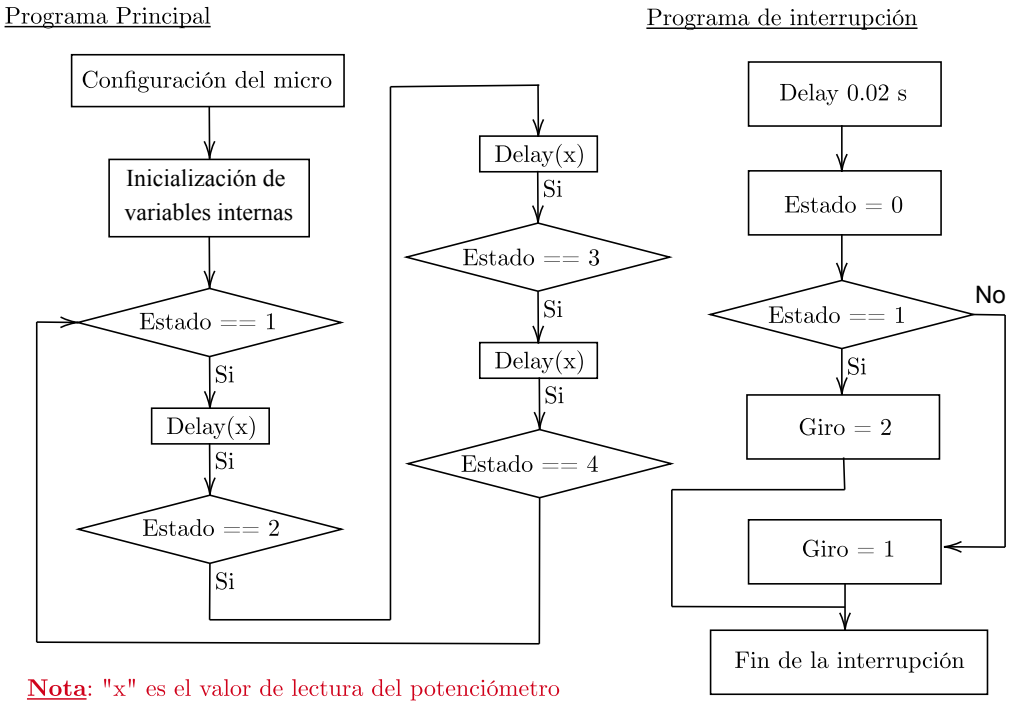

Figura 6.14: Flujograma del Programa de Control para el Motor Paso a Paso

De manera paralela, leeremos la lectura del potenciómetro, que nos permitirá controlar la velocidad de giro, aplicando un delay entre secuencia y secuencia.

El programa en cuestión lo tenemos disponible en la página pública de GitHub: Programa de Control del Motor PAP.

6.3.- Motor Brushless

Los motores Brushless o motores BLDC, son motores de continua sin escobillas, también conocidos como motores electrónicamente conmutados.

El motor Brushless está compuesto, como cualquier otro motor, de un estator, un rotor y una carcasa, esto a un nivel muy ambiguo. Si nos adentramos en el motor, podemos observar, que la estructura interna es diferente, con respecto al motor Paso a Paso, y al motor DC.

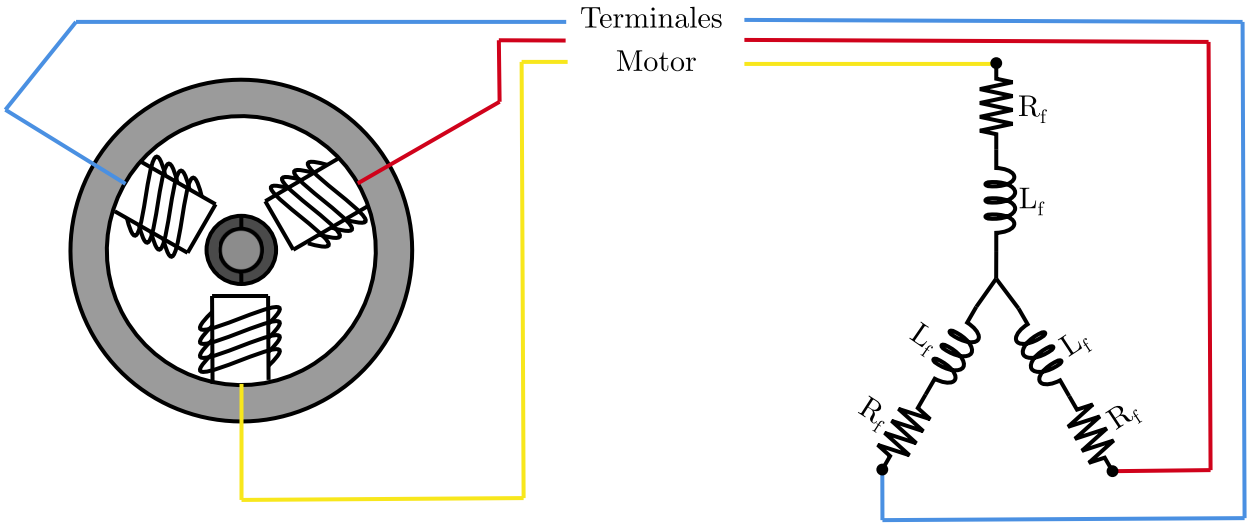

Figura 6.15: Funcionamiento del Motor Brushless

En la figura 6.15 se nos muestra el bobinado del motor junto al circuito equivalente. En este caso disponemos de una resistencia en serie con una inductancia por cada fase del motor.

Al igual que el motor Paso a Paso disponemos de una secuencia para el funcionamiento del mismo. Para ello vamos a necesitar de las **Sondas de efecto Hall**. Estas sondas funcionan de manera simple, cuando se excita el campo magnético nos devuelve un valor ”1”, y en caso contrario, nos devuelve un ”0”.

Tal como pasa en el Stepper, para la secuencia de giro del motor, se ha de seguir una serie de pasos, tal como se describen en la figura 6.16. Esta consiste en ir jugando con el PWM, la masa y los transistores abiertos de los terminales.

En la etapa de potencia de dicho motor, necesitamos disponer de un puente de 3 ramas, de la siguiente manera:

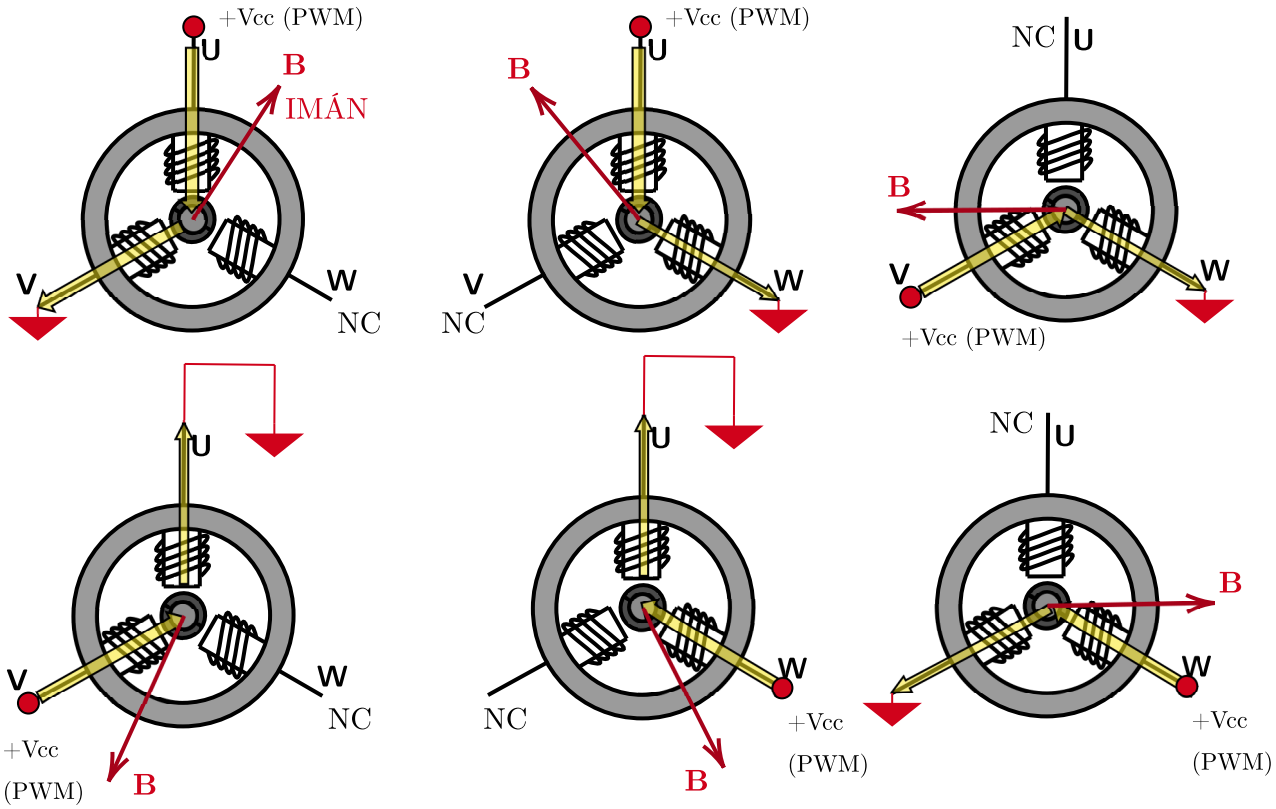

Figura 6.16: Secuencia de Giro del Motor BLDC

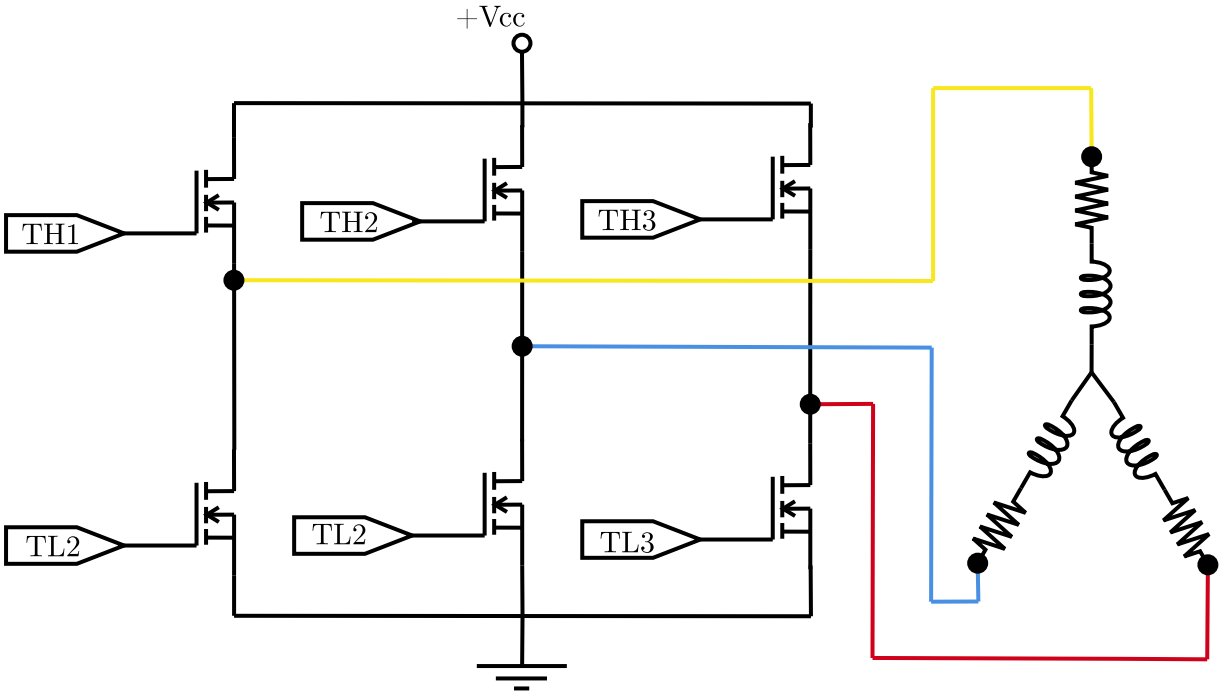

Figura 6.17: Topología de Potencia para el Control del Motor BLDC

En la figura 6.17 se muestra como sería la topología de potencia para el control, cada una de las ramas se correspondería a cada una de las fases.

Si tomamos como ejemplo la secuencia de la figura 6.16, al principio tendríamos la fase U con un pulso PWM, esto lo haríamos con uno de los transistores. la fase V a tierra, esto se consigue dejando el transistor de abajo cerrado y el de arriba abierto. Y por último, tendremos las fase W, este se conseguiría dejando tanto el transistor de arriba y el de abajo abierto.

Tal como se hace con esta primera, secuencia, las siguientes se hacen de la misma manera.

Una vez analizada la secuencia de giro y el funcionamiento del motor Brushless, vamos a ver como leer la posición del rotor en cada paso de secuencia con los **Sensores Hall**.

La función básica de estos sensores es la de realimentar, es decir, en un sistema de control de lazo cerrado, disponemos de una planta, una acción y un sensor, en este caso el sensor, son los sensores Hall. En función de la lectura de los sensores vamos a tener una acción diferente, por parte de la etapa de potencia.

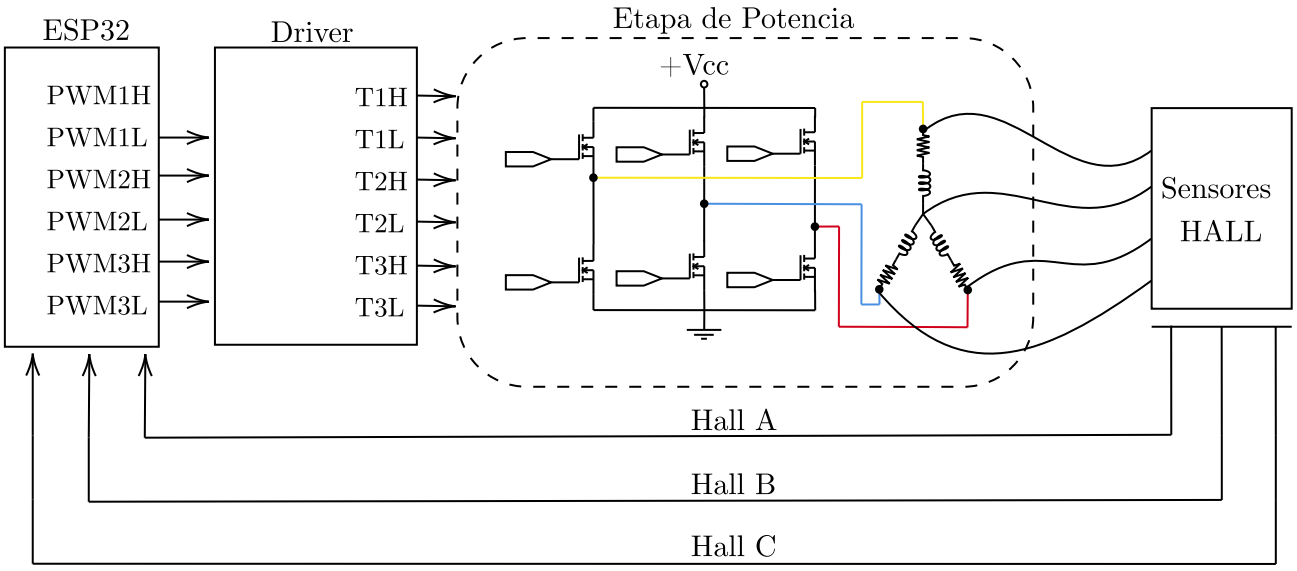

Figura 6.18: Sistema de Control para el Motor BLDC

En este punto del diseño, ya solo nos quedaría ordenar la secuencia en función a la lectura de los los sensores de efecto Hall, por tanto tendríamos como resultado:

Siguiendo la secuencia marcada en la figura 6.16, se muestra la lectura de las sondas correspondiente a cada una de las fases, en este caso la lectura de la sonda correspondería a las *Sonda Hall A* con la *Tensión de Fase U*, la *Sonda Hall B* con la *Tensión de Fase V* y la *Sonda Hall C* con la *Tensión de Fase W*.

Si seguimos la secuencia de arriba abajo, veremos como evolucionará el giro del motor BLDC, de manera correspondiente a la mostrada en la figura 6.16.

Sonda Hall A	Sonda Hall B	Sonda Hall C	Fase U	Fase V	Fase W
1	-1	0	Vcc	GND	NC
1	0	-1	Vcc	NC	GND
0	1	-1	NC	Vcc	GND
-1	1	0	GND	Vcc	NC
-1	0	1	GND	NC	Vcc
0	-1	1	NC	GND	Vcc

Cuadro 6.6: Medidas del sensor Hall

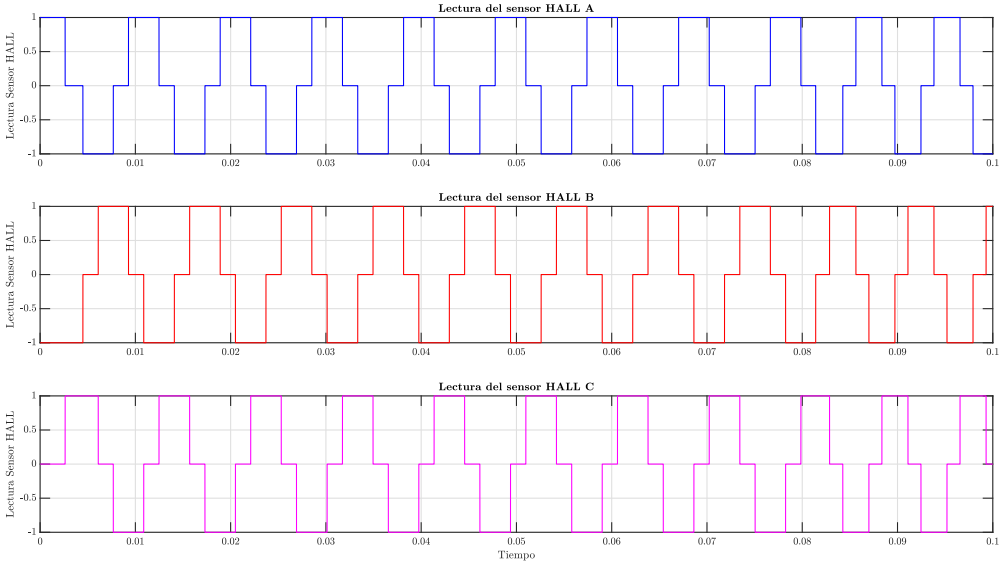

Figura 6.19: Evolución gráfica de la lectura de los sensores Hall

Una vez terminado con la topología de potencia correspondiente al motor BLDC, los sensores Hall, y el control del mismo, vamos a proceder a desarrollar el programa de control que se corresponde con esta sección.

En el mundo ideal la lectura de los sensores Hall no es como la representada en la figura 6.16, sino que solo leemos un 1 lógico o un 0 lógico, la referencia de 1, -1 y 0 se hace para poder representar y entender la secuencia de los estados de las distintas fases, siendo así un 1 cuando sea *Vcc*, un -1 cuando se corresponda a *GND*, y un 0 cuando este abierta la rama, *NC*.

6.3.1.- Parámetros del Motor Brushless

En este caso disponemos de un motor Brushless **RS PRO Brsuhless DC Motor**, es un modelo comercial proporcionado por la empresa RS, dentro de las cuales sus parámetros son los siguientes:

Supply Voltage	24 V
Power Rating	180 W
Current Rating	758 mA
No Load Speed	12900 rpm
No Load Current	0.055 A
Stall Torque	9.44 mNm
Stall Currente	0.629 A
Maximum Efficiency	80 %
Back-EMF Constant	14.8 V / 1000 rpm
Electrical Phasing	120º electrical
Terminal Resistance	1.8 Ω
Terminal Inductante	2.23 mH
Torque Constant	0.025 Nm/A
Speed Constant	1700 rpm/V
Speed / Torque Gradient	843 rpm/mNm
Mechanical Time Constant	8.85 ms
Rotor Intertia	24 kgm^2x10^{-6}
No. of Poles	8
No. of Phase	3
Maximun Peak Current	5.4 A

Cuadro 6.7: Parámetros del motor Brushless

En la tabla 6.7 se nos muestra los parámetros del motor, en este caso, son todo parámetros sacados del datasheet del mismo.

6.3.2.- Funcionamiento del motor Brushless: Programa de Control en MicroPython

El programa es sencillo, disponemos de un programa para realizar la secuencia de giro del Brushless, pudiendo cambiar el sentido de giro, junto al control de la velocidad del mismo, cambiando el tiempo de espera entre estado y estado.

Cada estado del programa hace referencia a los mostrados en la tabla 6.6, el estado del programa evoluciona de igual manera a las secuencia de las tabla, asignando a cada estado, cada secuencia.

De manera paralela ejecutamos un programa de interrupción, que permita hacer el cam-

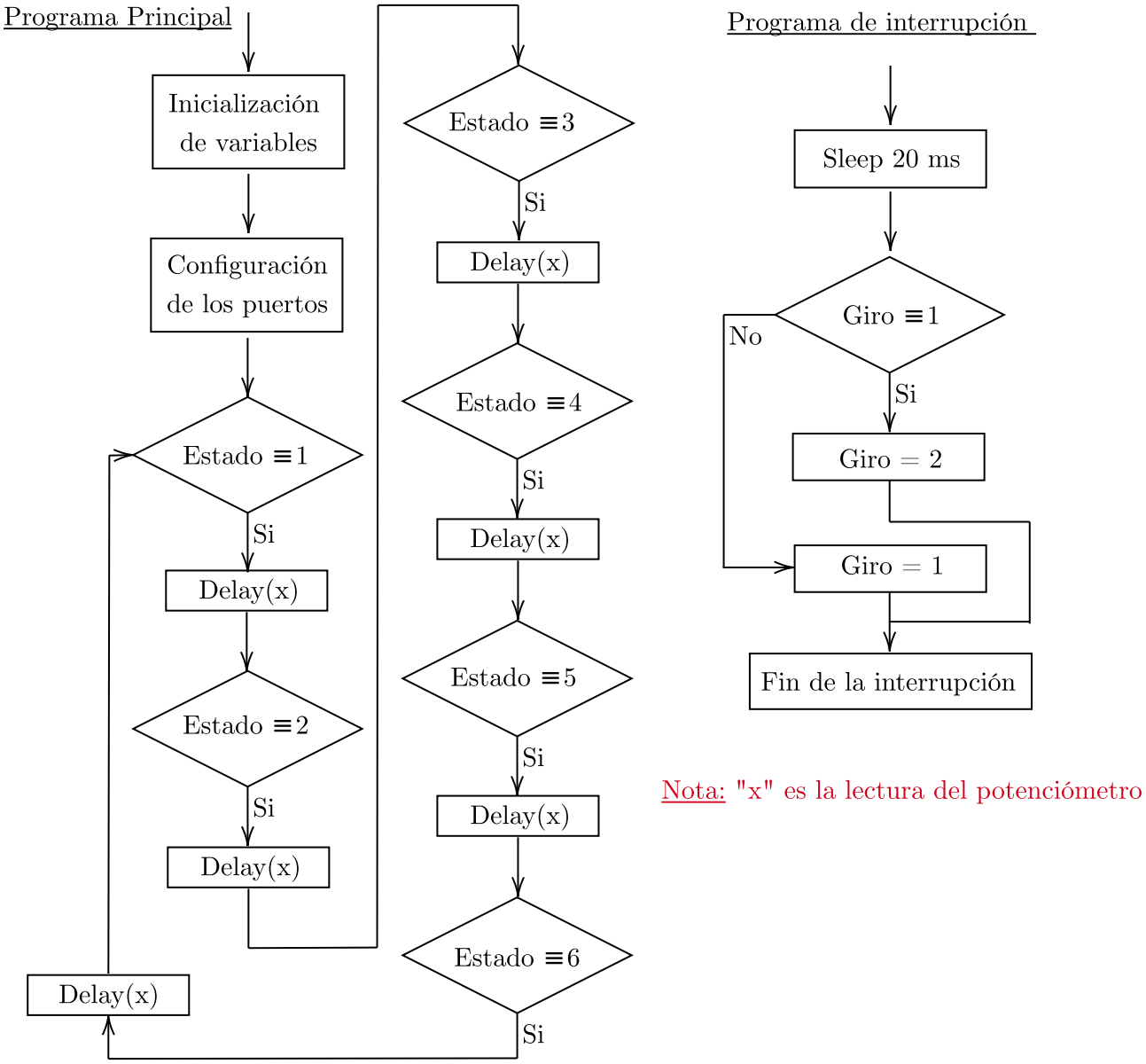

Figura 6.20: Flujograma del Programa de Control para el Motor Brushless

bio de giro, el cambio de giro, al igual que en el motor Paso a Paso, se realiza de igual manera, cambiando el orden de las secuencias, empezando al revés.

De igual manera el control de velocidad se hace controlando el tiempo de espera entre secuencia y secuencia.

El programa realizado lo tenemos disponible en la página pública de GitHub: Programa de Control del Motor Brushless

7. Control de motor con tecnologías IoT

En el capítulo 3, hacemos referencia de las características del ESP32, en este caso, una de las cosas fundamentales que hace al MCU, lo que es, es la comunicación **Wifi**.

El sistema de control que utilizaremos para el trabajo es un sistema basado en tecnologías IoT, dichas tecnologías nos permitirán, de forma remota, hacer el control de todo los parámetros del motor.

7.1.- Introducción a la comunicación IoT

Adentrándonos de manera breve en la historia y la importancia de esta revolución, esta comunicación va de la mano de un nuevo movimiento relacionado con la nueva revolución industrial, a este movimiento se le denomina **Industria 4.0**[1]

El término IoT proviene del inglés *Internet of Things*, es decir, *Internet de las cosas*. Este término esta asociado con la conexión de la electrónica con el mundo de internet, un ejemplo básico, sería la toma de datos de un sensor conectado a un microcontrolador, pudiendo leer dicha información y guardarla en una base de datos.

7.2.- Comunicación MQTT

El protocolo de comunicación que utilizaremos en este caso el **MQTT**.

MQTT se basa en un protocolo de mensajería basado en estándares, o conjunto de reglas, que se utiliza para la comunicación de un dispositivo a otro.

Aunque existan otro métodos de comunicación, nosotros trabajaremos con este, debido a su alta facilidad de trabajo, su gran eficiencia y su fiabilidad con respecto a otros protocolos de comunicación.

En la figura 7.1, se muestra un ejemplo de comunicación MQTT, en este caso, disponemos de 3 elementos básicos de trabajo:

1. **Cliente**: Este cliente tendrá acceso a la información del sensor, en este caso.

[1] Nuevo movimiento industrial asociada a la conexión con internet de la industria, a través de las tecnologías IoT

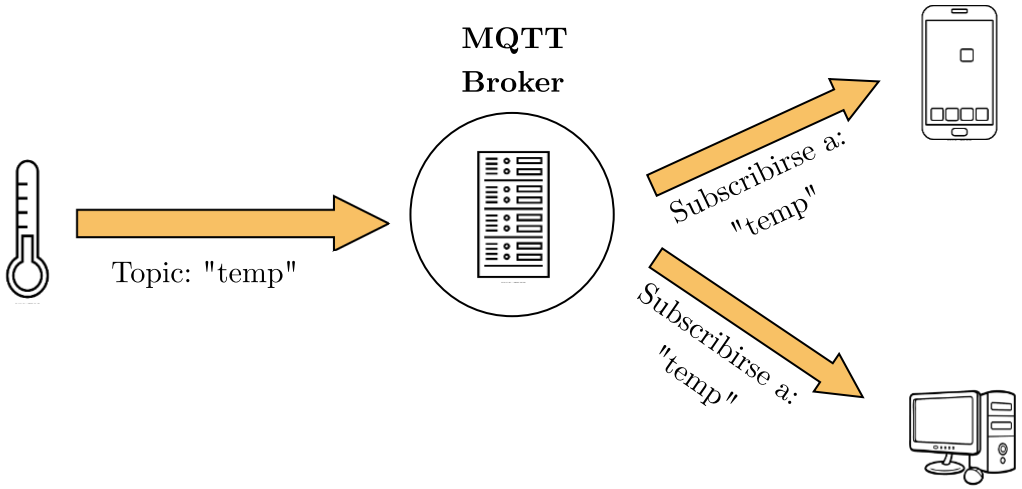

Figura 7.1: Comunicación MQTT

2. **Tópico**: Nos subscribiremos a un *tópico*, el tópico funciona como una dirección, es decir, esta dirección nos dirá donde se encuentra la información en cuestión.

3. **Broker**: Toda la información recibida y enviada, pasará por el Broker, el Broker es un servidor que almacena, transfiere y recibe toda la información que pasa por los tópicos.

7.2.1.- Comunicación Servidor

Vamos adentrarnos en la parte mas compleja de este trabajo, que es la comunicación con el servidor. El servidor en cuestión, se encuentra en el laboratio del grupo de investigación **CE3I2**.

En este caso, abarcaremos dos maneras diferentes de acceso. Este acceso puede ser sin seguridad TLS o con seguridad TLS.

7.2.2.- Conexión sin Seguridad TLS

En caso de querernos conectarnos al servidor con este tipo de seguridad, vamos a necesitar disponer de primer una conexión wifi y de la librería **umqtt**.

Esta librería nos permite usar las funciones propias para la comunicación, es por tanto, que nos va a ser esencial, por ello, vamos adentrarnos dentro de las funciones que nos ofrece.

Como punto de partida, para la explicación, vamos a hacer un pequeño programa para conectarnos al servidor e enviar mensajes.

1. Como primer paso, vamos a importar las librerías necesarias:

```
import network
from umqtt.simple import MQTTClient
import machine
from time import sleep
```

En este caso disponemos de 4 librerías, las básicas como hemos visto antes, la *machine*, *time*, y *network*, y como nueva librería, tenemos la librería **umqtt.simple**, esta librería integra todos los módulos necesarios para realizar esta comunicación.

2. En este segundo caso vamos a disponer de dos librerías externas, que no son necesarias, pero las vamos a utilizar para facilitar esta tarea:

```
import string
import random
```

Una librería nos permitirá el manejo de *strings*[2], y la otra librería nos permitirá el manejo de números aleatorios u otros tipos de datos.

3. Como segundo paso, configuraremos la conexión Wifi del mismo, tal como lo hemos hecho en la sección 5.6.

4. Con la conexión Wifi ya lograda, vamos a proceder a generar un nombre de cliente, configurar el servidor, los puertos de entrada ...

```
name_client = 'mallada_'+''.join(random.choice(string.ascii_uppercase
+ string.digits)
                                 for x in range(8))

mqtt_server = '156.35.154.170'
mqtt_port = 1883
mqtt_user = 'motores'
mqtt_password = '2motores3'
mqtt_topic = 'ae/motor1/ordenes'

mqtt_client = MQTTClient(name_client, mqtt_server, port = mqtt_port,
user = mqtt_user, password = mqtt_password, ssl = False)

while True:
    try:
        mqtt_client.connect()
```

[2]Cadena de caracteres

```
except Exception:
    continue
break
```

Si hemos seguido adecuadamente los pasos hasta este punto, ya tendremos la conexión con el servidor lograda, en este caso, hemos declarado los parámetros a meter en el servidor, los hemos volcado dentro de la clase creada, *mqtt_client*, y meteremos todo dentro de un *while*, para lograr la conexión.

Antes de continuar con el siguiente apartado, vamos a hacer un breve inciso sobre el uso de las librerías *string* y *random*. En este caso, para asegurarnos de que no haya complicaciones con el uso de varios nombres iguales, y que el servidor no detecte el correcto, añadimos un *string* aleatorio, que genere de manera automática cada vez que nos conectemos con el servidor. En conclusión, si no hacemos este paso, puede ser que no pase nada, pero hay que recordar que no podemos hacer dos usuarios con el mismo nombre, debido a que el servidor lo tomará como una intrusión y no nos dejará conectarnos.

5. Como último paso, procederemos a leer la información recibida por el micro, desde una terminal con conexión a internet externa, y la mostraremos por la consola del ordenador:

```
def leer(topic, message):
    print(message)

mqtt_client.set_callback(leer)
mqtt_client.subscribe(mqtt_topic)

while True:
    mqtt_client.wait_msg()
```

Definimos una función que nos permitirá leer los mensajes, esta función hará de *call-back*, esto se lo pasaremos a la función a continuación, nos subscribimos la tópico en cuestión, y en el bucle infinito, esperaremos a que el cliente nos mande el mensaje, para así poder mostrarlo por pantalla.

```
import network
from umqtt.simple import MQTTClient
import machine
from time import sleep

import string
import random

name_client = 'mallada_'+''.join(random.choice(string.ascii_uppercase
+ string.digits)
for x in range(8))

mqtt_server = '156.35.154.170'
mqtt_port = 1883
mqtt_user = 'motores'
mqtt_password = '2motores3'
mqtt_topic = 'ae/motor1/ordenes'

while True:
    try:
        mqtt_client.connect()
    except Exception:
        continue
    break

def leer(topic, message):
    print(message)

mqtt_client.set_callback(leer)
mqtt_client.subscribe(mqtt_topic)

while True:
    mqtt_client.wait_msg()
```

Code Listing 7.1: Programa para recibir mensajes por MQTT

Como resultado tenemos el programa que se muestra, un programa sencillo, y que nos va a servir de base para realizar distintos programas con infinitas posibilidades, no solo en accionamientos, sino que podremos abarcar distintos tipos de tareas, desde encender y apagar un led a distancia, como controlar la corriente que circula por un circuito.

7.3.- Aplicación de la conexión MQTT para el control de un motor DC

Aprovechando los conocimientos sobre la conexión MQTT, vamos aplicar estos conceptos para controlar un motor de continua, como el visto en el capítulo 6.1.

En este caso vamos a realizar un programa para controlar la velocidad del motor, y poder realizar una parada de emergencia.

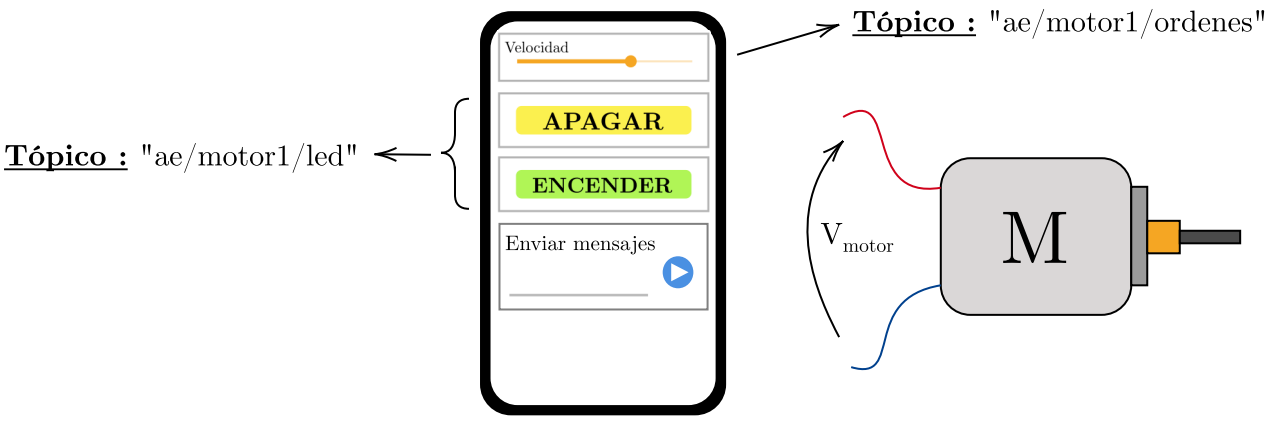

Figura 7.2: Control del motor mediante el móvil

En la figura 7.2, se nos muestra la configuración hecha desde el móvil para controlar el motor, en este caso disponemos de dos opciones, una barra superior, para controlar la velocidad del motor, esta enviará un valor desde 0 hasta 100. Por otro lado, tenemos las dos opciones de *apagar* y *encender*, que nos permitirán, apagar de manera interrumpida el motor en cualquier momento, al igual que lo podemos volver a poner en marcha, con al opción de *encender*.

El programa propuesto en esta ocasión se muestra, con su correspondiente flujograma, en la figura 7.3. El programa consta de un primera parte en la que se configuraremos todos los parámetros, declararemos las variables necesarias, nos conectamos a Internet y con el servidor del laboratorio. Seguido de esto, ejecutamos en bucle, el valor de duty recibido. La función *callback()* funcionará como una interrupción, en cuanto llegue el mensaje será interrumpido y leerá el valor en cuestión, en caso de que el llegué desde el tópico *ae/motor1/ordenes*, mientras que el otro tópico, nos indicará si para o no el motor.

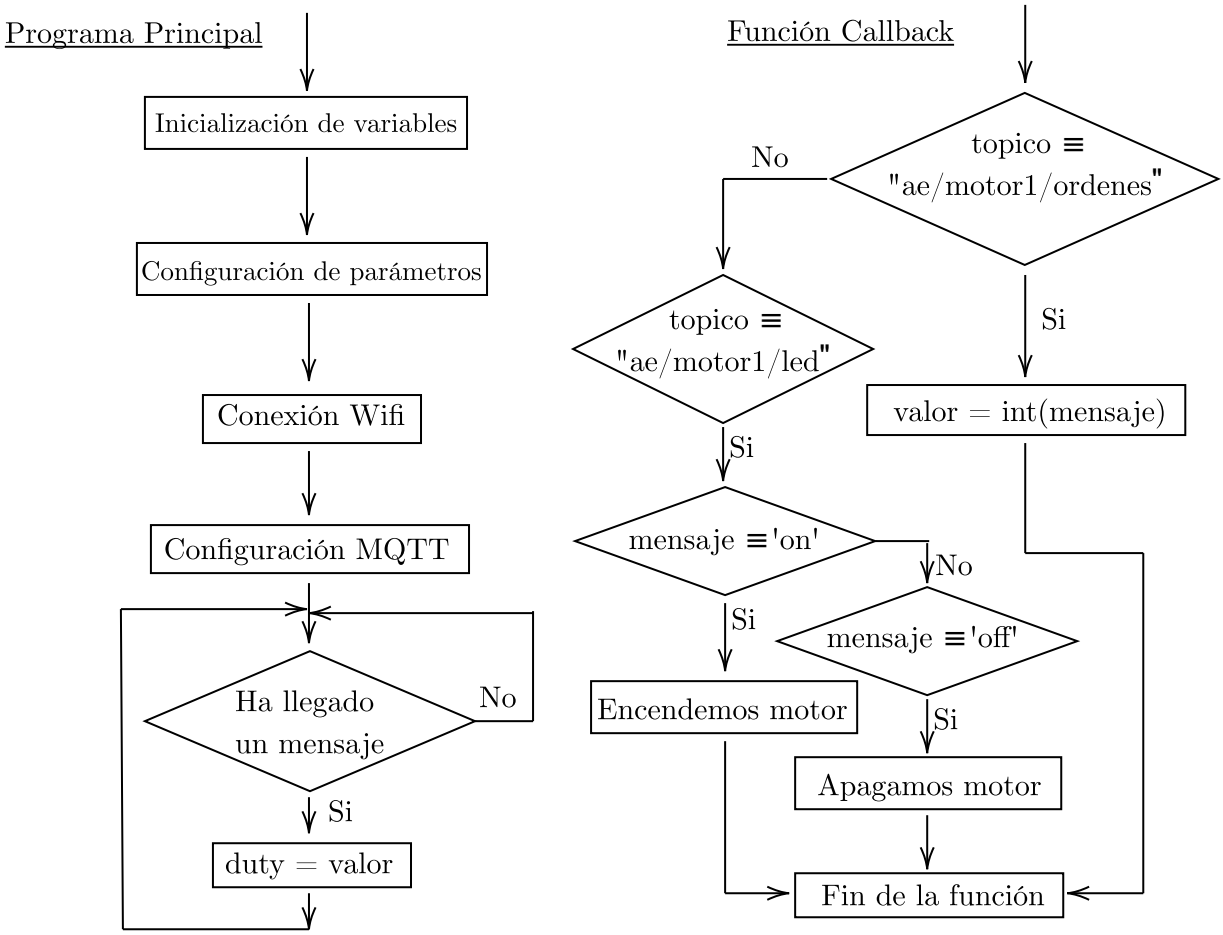

Figura 7.3: Flujograma para el control del Motor DC, vía MQTT

El programa realizado se encuentra disponible en la página de GitHub: Programa de Control de un Motor DC, vía MQTT

8. Diseño de la topología de Potencia

La topología de potencia utilizada es un inversor en puente monofásico. En este caso hemos utilizado un driver comercial que integra dos puentes en H, el driver utilizado es el L298N ([6]).

El desarrollo de la etapa de potencia, se ha divido en las siguientes fases: Etapa de entrada, sensor de corriente y driver L298N.

A continuación se desarrollarán más en detalle cada una de las etapas que se acaba de mencionar.

8.1.- Etapa de Entrada

Antes de nada, vamos a comenzar por la etapa de alimentación a la placa, en este caso disponemos de una etapa de entrada que consta de dos diodo de protección.

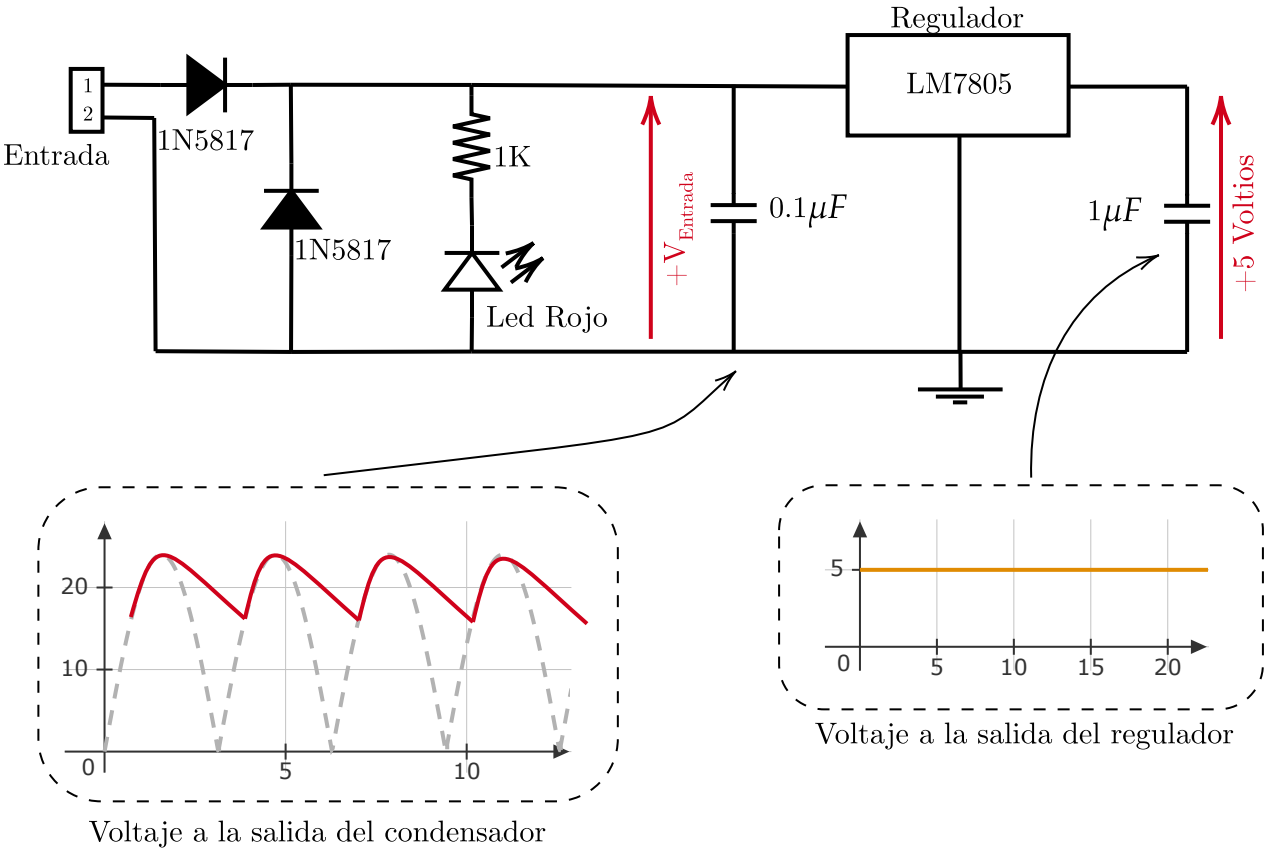

Figura 8.1: Etapa de Entrada + Regulador de tensión

Tal como se muestra en la figura 8.1, tras el diodo en paralelo, tenemos el led rojo, que

nos indicará cuando este alimentado la placa.

Esta tensión resultante será la tensión de entrada rectificada y con filtro por condensador, es decir, la tensión positiva.

Posterior a esta etapa, tendremos el *regulador*, en este caso hemos escogido el **LM7805**, este regulador, permite una entrada de tensión de máximo 24 Voltios. Los condensadores de entrada y salida son los recomendados por el fabricante para dar una tensión de salida de 5 Voltios.

8.2.- Sensor de corriente

En la teoría de control, necesitamos de una sistema de realimentación para poder realizar un control en *lazo cerrado*.

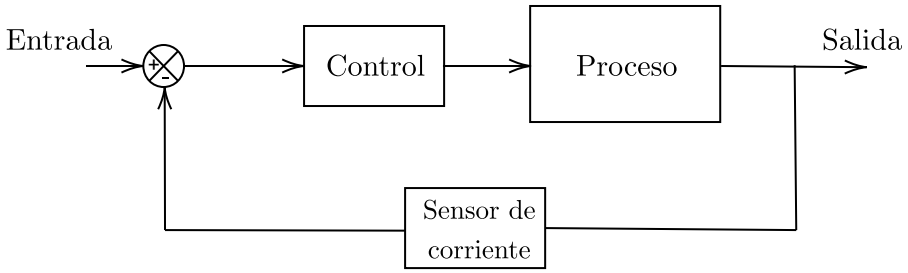

Figura 8.2: Sistema de control

En este caso, lo usaremos para tomar la medida de salida del motor, por lo que podremos medir su velocidad en función a la salida del sensor. Como mostramos anteriormente, la relación entre la tensión del motor y la velocidad, es una relación proporcional directa, por tanto podremos obtener la velocidad angular del motor en el instante actual.

Para realizar esta tarea, vamos a disponer de nuestro sensor corriente, el modelo **MAX9919**. Este modelo mide la tensión de una resistencia en función de la intensidad que corre por esta misma, es decir, la corriente que circula por la resistencia es la que circula por el motor, y la que referenciaremos en este sistema de control.

$$V_{medida} = I_{motor} \cdot R \tag{8.1}$$

En este caso la V_{medida}, es la tensión de referencia, resultante de corriente que circula por el motor por la resistencia (Ley de Ohm).

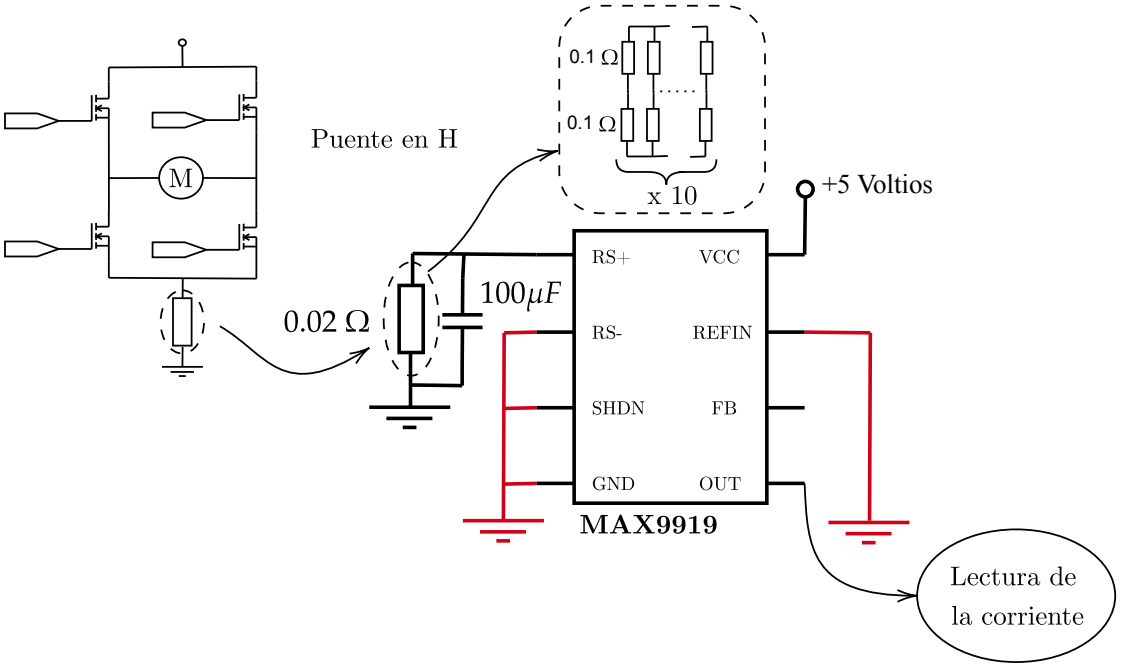

Figura 8.3: Conexión del Sensor de corriente

En la figura 8.3 vemos que la resistencia de referencia, que hablamos en el párrafo anterior, dicha resistencia es el punto de referencia para la medida de la corriente, debido a la alta corriente que circula por ello, vamos a disponer de dos resistencia en serie de 0,1Ω y 10 ramas en paralelo, por lo que tenemos:

$$\frac{1}{R_T} = \frac{1}{R} \cdot 10 = \frac{1}{0{,}1 \cdot 2} \cdot 10 \tag{8.2}$$

Lo que nos da como resultado una resistencia equivalente de 0.02 Ω

8.3.- Driver L298N

En este caso, como estamos disponiendo del lenguaje de MicroPython, una de las desventajas de este es que no tiene implementado puertos PWM complementarios, esto no quiere decir que el ESP32 no lo tenga, es más, tiene un módulo de control de motores, con 4 puertos PWM complementarios.

Para solucionar este problema, hemos decido disponer de los driver **L298N**, estos se componen de dos puente en H, con cada una de las ramas complementadas entre sí.

En la figura 8.4 se nos muestra como es la topología de potencia del driver, en este caso,

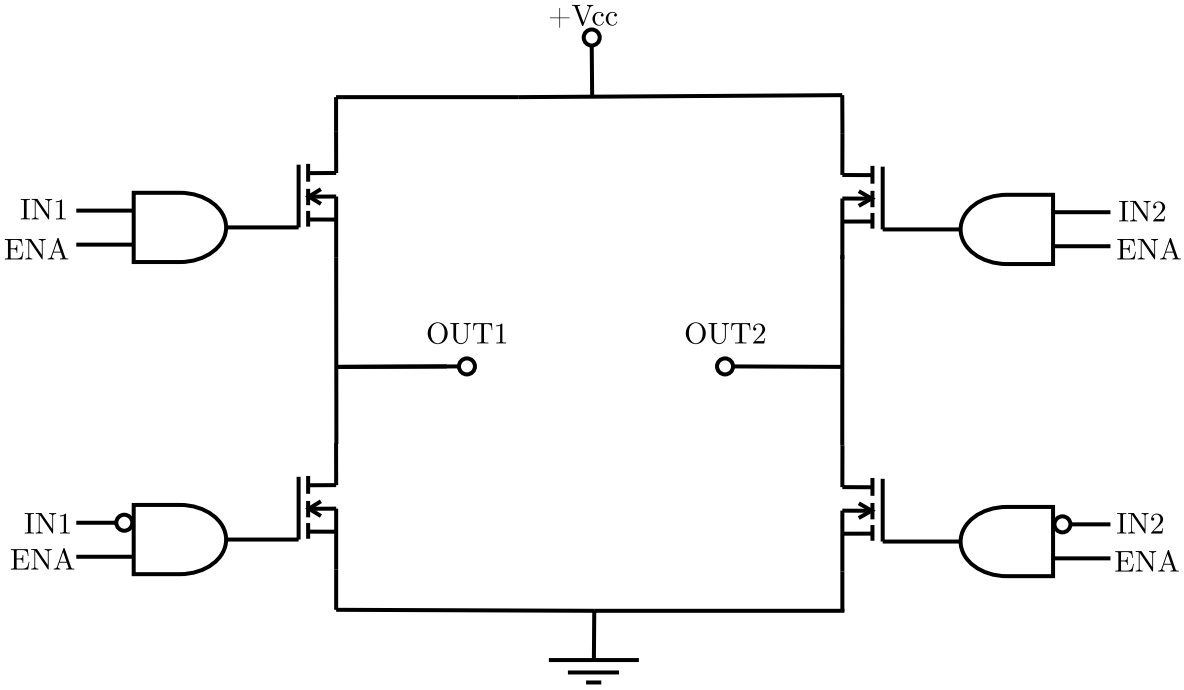

Figura 8.4: Esquema de la topología de potencia del Driver L298N

disponemos de un puente en H, con 3 entradas, estas se corresponderían a IN1, IN2 y ENA, dos de ellas, se corresponden a cada una de las ramas, y la otra entrada, ENA, se trataría de la entrada que permite abrir todas las ramas, en caso de paro, es decir, si esta entrada no se pone a uno no se puede trabajar con las ramas, y en caso de que la entrada sea un cero lógico, podemos abrir todas las ramas, independientemente de la entrada que se le introduzca a IN1 o IN2.

Entonces, al incluir las puertas AND, podremos evitar que se formen cortos entre las ramas, y de esta manera, no estar trabajando con los puertos PWM complementarios.

Como se muestra en la topología de la figura (8.4), vemos que las salidas de cada rama, se corresponde a OUT1 y a OUT2, entonces las combinaciones posibles serían las siguientes:

En la tabla de verdad, tabla 8.1, se muestra las posibles combinaciones que pueden tomar las entradas del driver, aunque no se va a tomar siempre distintas soluciones.

ENA	IN1	IN2
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Cuadro 8.1: Combinaciones de la entradas del puente en H

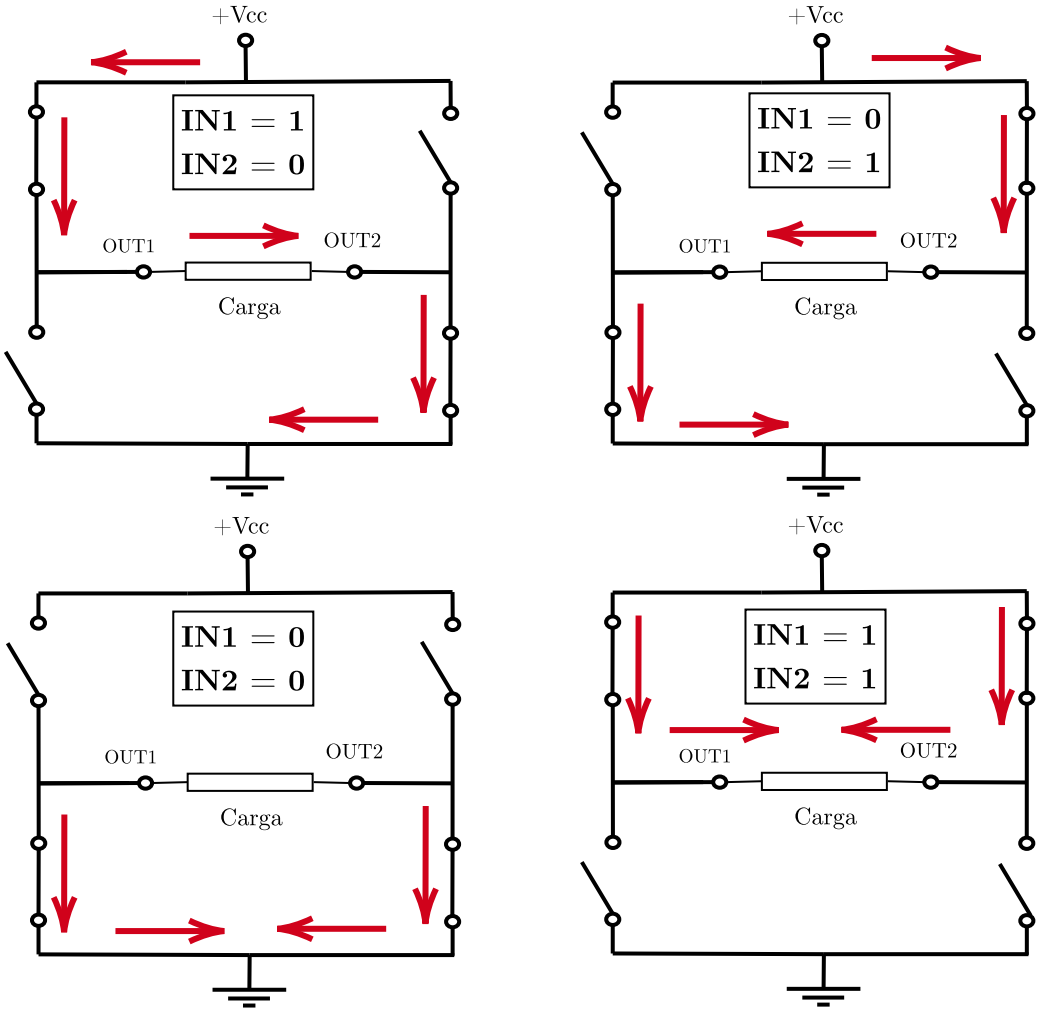

Figura 8.5: Combinaciones del puente en H en función de IN1 y IN2 (siempre que ENA sea 1)

Como podemos ver, figura 8.5, se pueden ver las distintas combinaciones de circulación de la corriente a través del puente, como podemos ver, siempre que ENA, reciba un 1 lógico, podemos realizar 4 combinaciones diferentes. En nuestro caso, siempre trabajaremos con las dos combinaciones de arriba, básicas para el cambio de giro.

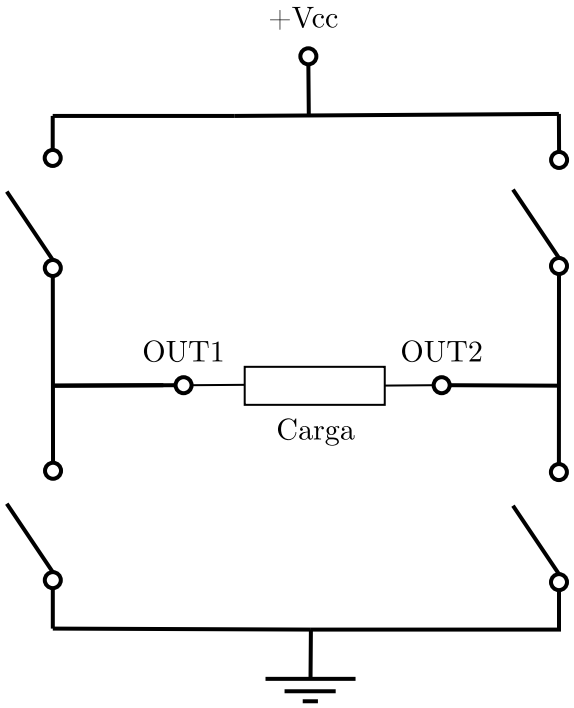

Figura 8.6: Puente en H, cuando la entrada ENA sea 0

Siempre que ENA sea igual a 0, todos los transistores estarán abiertos, por tanto la entrada IN1 e IN2, no dependerán para el funcionamiento del puente en H.

8.4.- Diseño PCB

Para el diseño de la PCB, hemos utilizado el **Altium21**.

En los anteriores aparatados hemos explicado las partes mas complejas que componen la placa, como son los driver y la alimentación, aunque no lo hayamos comentado, hemos añadido un par de led, unos pulsadores, un potenciómetro ...

Hemos añadido todos los elementos necesarios para completar todas las herramientas de MicroPython que hemos abordado a lo largo del trabajo.

En este caso tenemos como resultado una placa de 120,396 mm de ancho y de 96,52 mm de largo, se podría decir que tenemos una placa compacta con casi todas las prestaciones de una placa comercial de entrenamiento.

Disponemos como resultado final, el poder controlar 4 motores de continua, 2 motores Paso a Paso o un motor Brushless, están pueden ser las combinaciones que podemos hacer con los distintos motores. Esta es la gran ventaja de haber escogido el driver L298N.

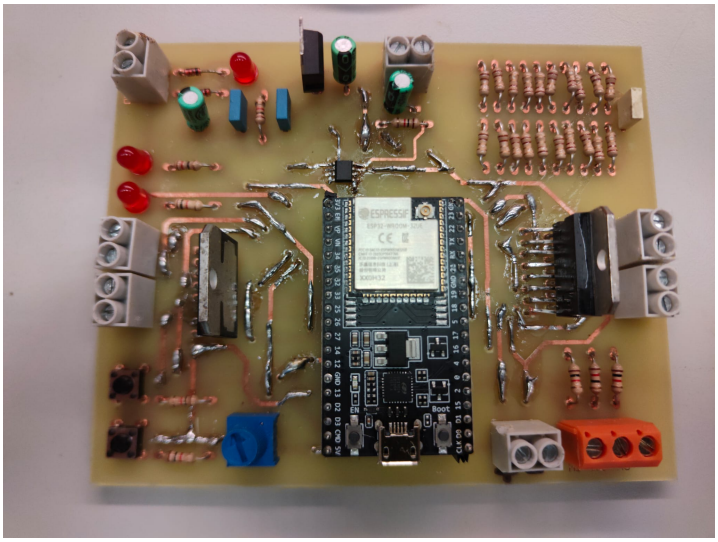

Figura 8.7: PCB de entrenamiento

La distribución de las salidas en las placas son las mostradas en la figura 8.8. La salida de los puentes, son las clemas laterales, 2 por cada lado, que se encuentran en la placa. En la entrada superior, disponemos de la alimentación de entrada de los driver y el MCU, y a su derecha tenemos la alimentación de la parte de potencia. En la parte posterior, tenemos dos clemas, una de 3 y otra de 2, la de 3 se corresponde a las entradas de los sensores Hall, y la de 2 a la alimentación de los mismos.

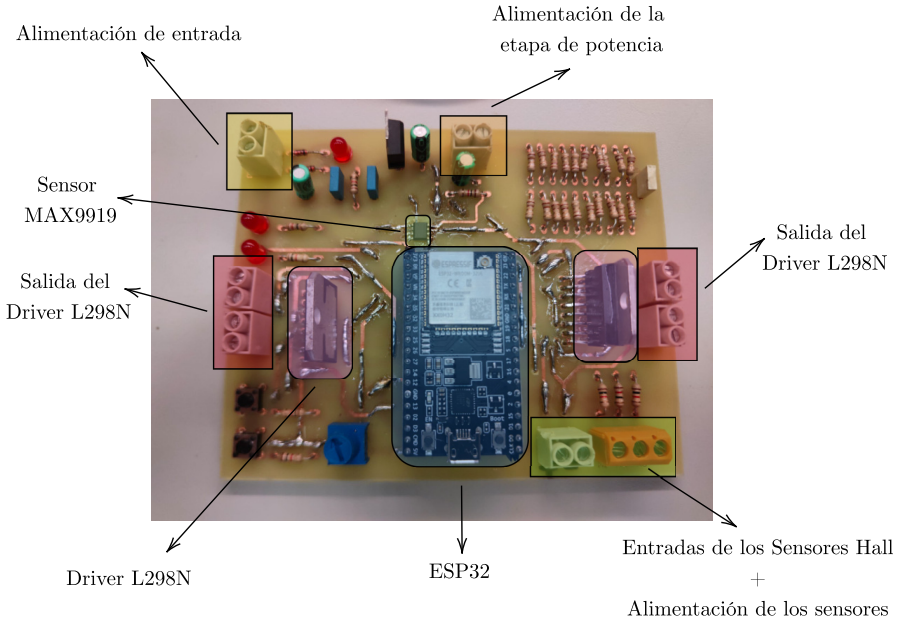

Figura 8.8: Componentes de la placa

9. Conclusiones y Futuras Mejoras

9.1.- Conclusiones

El resultado de este trabajo, ha sido un análisis de los distintos motores que se abarcan el mundo de la electrónica, junto a las distintas maneras de realizar el control de las mismas, junto a estos conceptos, hemos conseguido realizar una PCB, que pueda sustentar todos los conceptos abarcados en el trabajo.

Entonces podemos dividir este trabajo en las siguientes partes:

- **Breve estudio del ESP32:** En este caso se ha estudiado, analizado y sacado aquellos parámetros del ESP32 que nos pudiese traer problemas en un futuro a nosotros o cualquier otro usuario que disponga de este trabajo. Además se explica el breve funcionamiento del mismo, de manera superficial, junto a las posibilidades de la configuración de los puertos.
- **Instalación, introducción y uso al firmware de MicroPython:** Como compilador, hemos dispuesto del firmware de MicroPython, este nos permitirá un uso mas sencillo para los programas, reducir la curva de aprendizaje para todo aquel que no haya programado un microcontrolador anteriormente, además de ser ventajoso a la hora de desarrollar aplicaciones u programas, debidos a que muchos de ellos se comparten con el Python básico.
- **Estudio y ensayo de los motores:** Como primer punto, hemos estudiado los motores a utilizar, motor de continua con escobillas, el motor Brushless y el motor Paso a Paso, acompañando a como es el funcionamiento y los parámetros de los mismos. De esta manera podremos desarrollar el prototipo para el ensayo de los mismos, y el programa de control de estos.
- **Estudio y desarrollo de la topología de Potencia:** Para poder implementar los programas desarrollados se ha decidido disponer del integrado L298N, tras varios intentos y ensayos, se ha decido que el uso de este driver es el más eficaz, además de el más sencillo de manejar para evitar que se realice algún corto indebido, o se pueda causar algún problema con los motores.
- **Programación e implementación de sistemas IoT con el protocolo MQTT:** Se ha desarrollado un breve tutorial y una introducción para el desarrollo de aplicaciones IoT apliacadas a los accionamientos electrónicos, es por ello, que se ha introducido el funcionamiento del protocolo MQTT y la implementación de los mismos.

- **Desarrollo de un prototipo para el ensayo de motores:** Una vez realizado los ensayos pertinentes, se ha desarrollado una placa PCB para implementar los programas propuestos u otros programas, es por ello, que se ha desarrollado en función a que funcione como placa de entrenamiento, es decir, se ha implementado un potenciómetro, pulsadores, led´s, etc.

Llegar hasta este punto del trabajo ha sido gracias a los conocimientos aprendidos a lo largo de toda la carrera, sobre todo el haber cursado *Accionamientos Electrónicos* y *Desarrollo de Prototipos*, el haber obtenido la habilidad necesaria para trabajar en el laboratorio, junto a la implementación de sistemas electrónicos, son conceptos básicos que se han llevado a cabo en estas asignaturas. En cuanto al marco teórico de este trabajo, los conocimientos aprendidos, fueron gracias a las bases dadas en *Electrónica de Potencia* y en *Sistemas Electrónicos Digitales*, el paso por ellas ha sido de gran ayuda tanto para la parte digital, como para el desarrollo, estudio y ensayo de la parte de potencia del mismo.

9.2.- Futuras Mejoras

Como se indica a lo largo del trabajo, hemos desarrollado un prototipo con propósito educacional.

Desarrollo de una placa mejor, como podría ser el disponer de transistores, es decir, poder realizar, que permitan una mayor tensión de bus, una alimentación de la etapa de potencia mayor, junto a poder realizar otras aplicaciones, como viene siendo, poder hacer un el control de un inversor, poder controlar motores de alterna, u otra aplicación que necesite de unos valores de tensión y corriente mayores.

Por otro lado, otra propuesta de mejorar, desde la parte de desarrollo de software, podría ser el implementar otro protocolos de comunicación, como puede ser, la comunicación AMQP, HTTP, etc. Aunque un paso predecesor a este, sería pulir la comunicación MQTT por el puerto 8883, es decir, poder trabajar desde con un protocolo de seguridad TLS, sin necesidad de estar entrando por el puerto 1883.

Otro punto a tomar en cuenta, es diseñar un mejor metodo de la toma de parámetros del motor, es decir, en este caso las mediciones solo las realizamos con el motor DC, pero no hemos diseñado ningún método para la medición de otros valores, como viene a ser la velocidad, par motor, etc, sería otro posible paso a mejorar.

Por último quería enfatizar como posible mejora, sería el poder aprovechar de la manera

mas eficiente el ESP32, es decir, el ESP32, tal como se cuenta en el capítulo 3, posee dos núcleos de trabajo, el punto es poder trabajar de manera paralela con ambos núcleos ejecutando tareas simultáneas, como puede ser realizar un control PI en un núcleo, y el otro utilizarlo para enviar los datos por MQTT.

A. Prototipo de placa PCB para el ensayo de motores eléctricos

La placa desarrollada se ha desarrollado en el programa de diseño de placas PCB **Altium Designer**. En el se ha desarrollado el esquema de conexiones junto a la vías de la placa, junto a las disposiciones de los distintos componentes.

El esquemático realizado en el proyecto ha sido el mostrado en la figura A.1, este se nos muestra las distintas conexiones que se han realizado junto a los componentes elegidos y el calibre de los mismos.

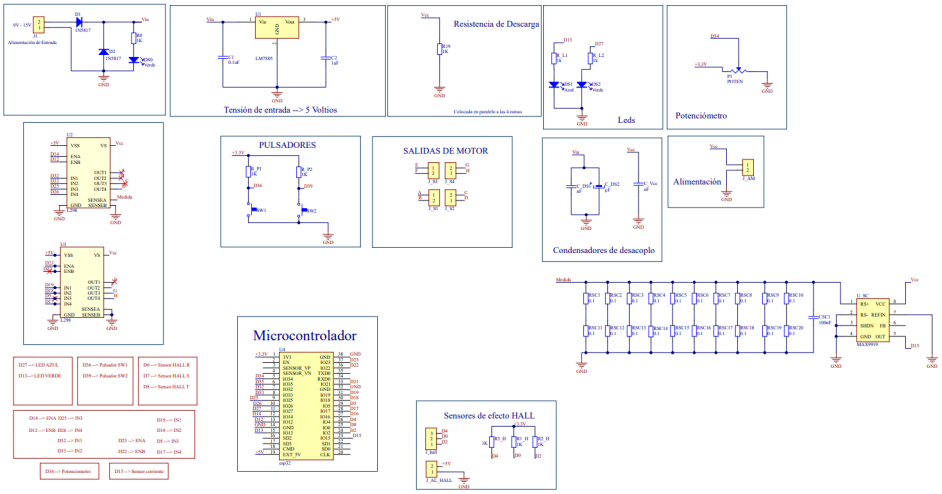

Figura A.1: Esquema de conexiones de la placa PCB

En la cara superior (figura A.2) encontramos el marco de los componentes seleccionados junto a la vías correspondientes de la cara superior.

En la cara inferior disponemos de un mayor número de vías, debido a que hemos llevado todas las vías de la etapa de potencia, junto a casi todas de las de control, por esta cara, esta decisión fue tomada en función a que se han seleccionado solo componentes de inserción para esta tarea.

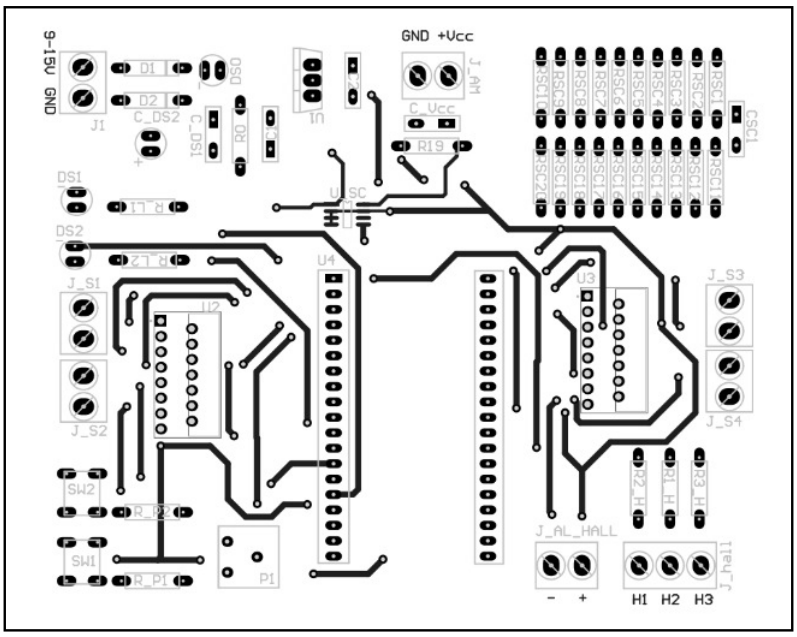

Figura A.2: Cara superior de la placa PCB

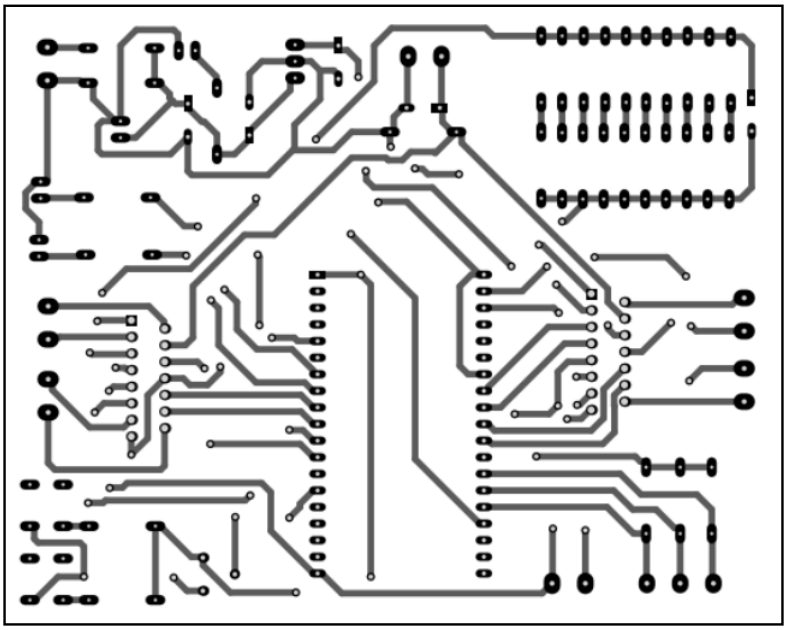

Figura A.3: Cara inferior de la placa PCB

Bibliografía

[1] R. Pasic, I. Kuzmanov, and K. Atanasovski, "Espressif esp32 development board in wifi station communication mode," *TEMEL-ij*, vol. 4, no. 1, pp. 1–6, 2020.

[2] C. Bell, *MicroPython for the Internet of Things*. Springer, 2017.

[3] N. H. Tollervey, *Programming with MicroPython: embedded programming with microcontrollers and Python*. .ºReilly Media, Inc.", 2017.

[4] R. A. T. Figueroa and W. Valderrama, "Control de velocidad y sentido de giro para un motor dc.," *Infometric@-Serie Ingeniería, Básicas y Agrícolas*, vol. 3, no. 1, pp. 103–115, 2020.

[5] J. Artigas, L. Barragán, and A. Sanz, "Tarjeta de control de motor paso a paso para practicas de pld," 1970.

[6] E. Yime Rodríguez and J. Páez Almentero, "Diseño mecatrónico de una shield de arduino para el control de motores dc con escobillas," *Prospectiva*, vol. 14, no. 1, pp. 73–79, 2016.