Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERÍAS INDUSTRIALES

Grado en Ingeniería Electrónica Industrial y Automática

TRABAJO FIN DE GRADO

Control de un motor de corriente continua mediante microcontrolador ESP32

Autor:
Sergio Fontanillo López

Tutores:

Eduardo Zalama Casanova **José Candau Pérez**

Departamento de Ingeniería de Sistemas y Automática

Julio – 2025

AGRADECIMIENTOS:

Quiero expresar mi más sincero agradecimiento a todas las personas que me han acompañado y apoyado no solo durante la realización de este trabajo, sino incondicionalmente durante toda mi vida.

En primer lugar, a mis padres, por su esfuerzo, confianza en mí durante toda la carrera. A Diego, por su apoyo constante y ánimo incondicional en momentos difíciles

A Carla, por alegrarme el corazón y ser una fuente de constante inspiración. Por toda su paciencia, cariño y motivación que me han ayudado a seguir adelante cuando más lo necesitaba.

A mi grupo de amigos, por acompañarme en este recorrido con su compañía, humor y apoyo en cada etapa.

A mis tutores, Eduardo Zalama Casanova y José Candau Pérez por su dedicación, su guía y su paciencia a lo largo de este trabajo. Gracias por compartir sus conocimientos, por orientarme y por transmitirme la confianza necesaria para afrontar cada reto.

Gracias a todos. Sin vosotros, llegar hasta aquí habría sido mucho más difícil.

RESUMEN

Este trabajo tiene como objetivo la actualización integral de una maqueta didáctica empleada en la asignatura *Fundamentos de Automática* en la Escuela de Ingenierías Industriales de la Universidad de Valladolid. En primer lugar, se analiza la maqueta original basada en un microcontrolador PIC y el entorno AUTOPIIC, destacando sus funcionalidades y limitaciones. A continuación, se presenta un nuevo diseño que incorpora un microcontrolador ESP32 de mayores prestaciones, un puente H L298N y diversos periféricos como un codificador rotativo KY-040 y una pantalla LCD I2C, lo que permite dotar al sistema de una interfaz HMI más versátil e intuitiva.

El proyecto contempla la comunicación bidireccional con Simulink (MATLAB) para la identificación y control avanzado del motor de corriente continua, así como la posibilidad de trabajar de manera autónoma mediante modos de funcionamiento configurables: identificación, control PID local y control externo en tiempo real.

Esta actualización persigue no solo simplificar la complejidad técnica del montaje, sino también mejorar su valor didáctico, permitiendo a los estudiantes experimentar con lazos de control reales y comprender la influencia de fenómenos prácticos como el desgaste mecánico, la zona muerta o la saturación de las señales, aspectos que no suelen contemplarse en modelos teóricos ideales.

PALABRAS CLAVE

Maqueta didáctica, ESP32, control PID, identificación de sistemas, Simulink, motor de corriente continua, automatización industrial.

ABSTRACT

This project aims to comprehensively update a didactic training platform used in the subject *Fundamentos de Automática* at the School of Industrial Engineering, University of Valladolid. Firstly, the original prototype based on a PIC microcontroller and the AUTOPIIC environment is analyzed, highlighting its features and limitations. A new design is then proposed, integrating an ESP32 microcontroller with enhanced capabilities,

an L298N H-bridge driver, and several peripherals such as a KY-040 rotary encoder and an I2C LCD display, providing a more versatile and intuitive human-machine interface.

The system enables bidirectional communication with Simulink (MATLAB) for advanced motor identification and control, as well as fully autonomous operation via configurable modes: identification, local PID control, and real-time external control.

This update not only reduces the technical complexity of the assembly but also enhances its educational value by allowing students to experiment with real control loops and to understand the impact of practical factors—such as mechanical wear, dead zones, or signal saturation—that are not usually considered in ideal theoretical models.

KEYWORDS

Didactic platform, ESP32, PID control, system identification, Simulink, DC motor, industrial automation.

ÍNDICE

1. INTRODUCCIÓN Y OBJETIVOS 13

1.1 INTRODUCCIÓN 13

1.2 JUSTIFICACIÓN Y MOTIVACIÓN 14

1.3 OBJETIVOS 14

1.3.1 OBJETIVOS ESPECÍFICOS 15

2. FUNDAMENTOS TEÓRICOS 17

2.1. FUNDAMENTOS DE LA IDENTIFICACIÓN DE SISTEMAS 17

2.2. PRINCIPIOS DEL CONTROL PID 18

2.2.1. CONTROL EN LAZO ABIERTO 18

2.2.2. CONTROL EN LAZO CERRADO 19

2.2.3. INTRODUCCIÓN AL CONTROL PID 19

2.3. MODELADO DE MOTORES DE CORRIENTE CONTINUA 23

2.4. CONTROL DE UN MOTOR ELÉCTRICO: EL PUENTE H 27

2.5. COMUNICACIÓN ENTRE MICROCONTROLADORES Y PERIFÉRICOS 30

2.5.1. El ECOSISTEMA ARDUINO 30

2.5.2. ARDUINO 31

2.5.3. JUSTIFICACIÓN DEL USO DE ARDUINO 32

2.5.4. PROGRAMAR EN ARDUINO IDE 32

2.5.5. EL ENTORNO MATLAB 37

2.5.6. SIMULINK 39

2.5.7. JUSTIFICACIÓN DEL USO DE SIMULINK 42

2.6. PROTOCOLOS DE COMUNICACIÓN 44

2.6.1. PROTOCOLO UART 44

2.6.2. PROTOCOLO I2C 47

3. MAQUETA PREVIA – DISEÑO Y SELECCIÓN DE COMPONENTES 52

3.1. DESCRIPCIÓN DE LA MAQUETA PREEXISTENTE 52

3.2 MOTOR DE CORRIENTE CONTINUA 54

3.3 SOFTWARE DE MOTORPIIC 56

3.4 LIMITACIONES DE LA MAQUETA ANTERIOR 57

4. NUEVA MAQUETA – ANÁLISIS, DISEÑO Y DESARROLLO 59

4.1. SELECCIÓN DE COMPONENTES - HARDWARE DE LA MAQUETA 59

4.1.1. ACTUALIZACIÓN DEL MICROCONTROLADOR 60

4.1.2. MÓDULO ESP32-WROOM-32 62

4.1.3. ACTUALIZACIÓN DEL PUENTE H EN LA MAQUETA 67

4.1.4. MOTOR DE CORRIENTE CONTINUA 73

4.1.6. OTROS ELEMENTOS DE LA PLACA A DESARROLLAR 83

5. DESARROLLO DEL FIRMWARE 96

5.1. ESTRUCTURA DEL CÓDIGO 96

5.1.1. LECTURA DE SEÑALES 97

5.1.2. CONTROL DEL MOTOR 98

5.1.3. INTERFAZ HMI: MENÚ Y NAVEGACIÓN 99

5.2. MODOS DE CONTROL 101

5.2.1. MODO IDENTIFICACIÓN 102

5.2.2. MODO PID LOCAL 103

5.2.3. MODO SIMULINK 104

6. DESARROLLO DEL MODELO EN MATLAB/SIMULINK 112

6.1. MODELADO MATEMÁTICO DEL MOTOR 112

6.2. CONFIGURACIÓN DE LA INTERFAZ HARDWARE IN THE LOOP (HIL) 114

6.3. ESTRATEGIAS DE IDENTIFICACIÓN EXPERIMENTAL 119

6.4. IMPLEMENTACIÓN DEL CONTROL PID 121

6.5. ANÁLISIS DE RESULTADOS 122

7. CONCLUSIONES, LÍNEAS DE FUTURO Y PROPUESTAS DE MEJORA 125

8. BIBLIOGRAFÍA 131

ANEXOS 136

A. ESQUEMAS ELÉCTRICOS DETALLADOS 136

B. LISTADOS DE MATERIALES Y COSTE 145

C. CÓDIGO FUENTE PRINCIPAL 149

D. MODELOS 3D (STL) 159

E. ARCHIVOS GERBER DE LA PLACA 161

F. MANUAL DE USUARIO Y ANEXOS EXTERNOS AL DOCUMENTO 162

ÍNDICE DE FIGURAS

Figura 1: Representación de un lazo de control (correspondiente a un control de lazo cerrado). Extraído de [1]. 18

Figura 2: Representación de un control en lazo abierto. Elaboración propia. 18

Figura 3: Representación de un control en lazo cerrado. Extraído de [2]. 19

Figura 4: Ejemplo PID con el acelerador de un coche. Extraído de [3]. 21

Figura 5: Esquema conceptual de motor eléctrico DC. Extraído de [4]. 23

Figura 6: Modelo de simulación para un motor DC. Extraído de [5]. 25

Figura 7: Función de transferencia del sistema. Elaboración propia. 26

Figura 8: Ilustración de un puente H diseñado en la aplicación KiCAD. Elaboración propia. 27

Figura 9: Ilustración del funcionamiento de la señal PWM con una bombilla LED. Extraído de [6]. 29

Figura 10: Logotipo del entorno de desarrollo y placas electrónicas Arduino. Extraído de [7]. 31

Figura 11: Interfaz del programa Arduino IDE. Elaboración propia. 34

Figura 12: Selección del microcontrolador en el entorno de Arduino IDE. Elaboración propia. 36

Figura 13: Software de desarrollo MATLAB. Extraído de [8]. 37

Figura 14: Breve ilustración de la interfaz de MATLAB. Elaboración propia. 38

Figura 15: Imagen extraída de MathWorks. Objetivos de proyectos con Simulink. Extraído de [9]. 41

Ilustración 16: Representación de protocolo UART entre dos dispositivos. Extraído de [10]. 44

Figura 17: Diferencias entre baudios y bps. Extraído de [11]. 45

Figura 18: Ejemplo de trama protocolo UART. Imagen editada y extraída de [12]. 46

Figura 19: Ilustración de comunicación empleando el protocolo I2C. Extraído de [13]. 48

Figura 20: Vista general de la maqueta "Motorpiic". Extraído de documentación anexada. 53

Figura 21: Interfaz del software AUTOPIIC. Elaboración propia. 56

Figura 22: Microcontrolador dsPIC30F4012. Extraído de hoja de datos anexada. 60

Figura 23: Arquitectura del dsPIC30F6011A. Extraído de hoja de datos anexada. 61

Figura 24: Disposición de pines dsPIC. Extraído de hoja de datos. 62

Figura 25: Microcontrolador ESP32 DevKit V1 de 38 pines. Extraído de [14]. _____ *63*

*Figura 26: Diagrama funcional de la ESP32 DevKit V1. Extraído de [14].*_____ *64*

*Figura 27: Pinout del ESP32 DevKit V1. Extraído de [14].*_____ *65*

Figura 28: Puente H HIP4081. Extraído de la hoja de datos anexada. _____ *67*

Figura 29: Diseño de control para velocidad y sentido de giro con medición de corriente para HIP4081, empleado en la maqueta. Extraído de la hoja de datos anexada. _____ *68*

Figura 30: Controlador L298N ZIP-15. Extraído de la hoja de datos anexada. _____ *68*

*Figura 31: L298N en formato módulo. Extraído de la hoja de datos anexada.*_____ *69*

Figura 32: Diagrama de bloques del L298N para el control de dos motores (A y B). Extraído de la hoja de datos anexada. _____ *70*

Figura 33: Breve resumen de la hoja de datos del diodo 1N5819. Extraído de la hoja de datos anexada. _____ *72*

*Figura 34: Diagrama de bloques del regulador L7805CV. Extraído de la hoja de datos anexada.*___ *72*

*Figura 35: Conexiones internas del L7805CV. Extraído de la hoja de datos anexada.*_____ *73*

Figura 36: Motor de corriente continua JGA25 – 370 con reductora y codificador incremental en cuadratura. Extraído de [15]. _____ *74*

Figura 37: Disposición de pines codificador incremental. Elaboración propia. _____ *74*

*Figura 38: Principio de funcionamiento de las pistas A y B en el codificador incremental. Extraído de [16].*_____ *76*

Figura 39: Ubicación de la resistencia Rsa para medición de corriente. Extraído de la hoja de datos anexada y editado. _____ *77*

*Figura 40: Diseño del OpAmp LM358P. Extraído de la hoja de datos anexada.*_____ *79*

Figura 41: Configuración de OpAmp para amplificador no inversor. Extraído de [17]. _____ *80*

Figura 42: Disipador de calor empleado en la placa electrónica. Extraído de [18]. _____ *83*

*Figura 43: Socket DIP-8 para amplificador y pines hembra para ESP32. Elaboración propia,*_____ *83*

Figura 44: Codificador rotativo KY-040 (izquierda), pantalla I2C LCD 1602 (centro) y pines macho (derecha). Elaboración propia, _____ *84*

*Figura 45: Logotipo del software libre KiCAD. Extraído de [19].*_____ *84*

Figura 46: Esquema de conexiones eléctricas de la placa. Elaboración propia. _____ *87*

Figura 47: Vista superior de la placa en el visor 3D de KiCAD. Elaboración propia. 88

Figura 48: Vista inferior de la placa desde el visor 3D de KiCAD. Elaboración propia. 89

Figura 49: Resultado de la fabricación de la placa electrónica (sin el amplificador, debido a una versión anterior de esta). Elaboración propia. 90

Figura 50: Maqueta ensamblada (vista en alzado y principal). Elaboración propia. 91

Figura 51: Etapas de la maqueta. Elaboración propia. 91

Figura 52: Maqueta en funcionamiento. Modo de identificación sin conexión. Elaboración propia. 92

Figura 53: Modo de control Simulink. Conexión directa con MATLAB/Simulink. Elaboración propia. 93

Figura 54: Modo de control local. Elaboración propia. 94

Figura 55: Diagrama de opciones del menú HMI. Elaboración propia. 100

Figura 56: Diagrama de bloques modo Identificación. Elaboración propia. 103

Figura 57: Diagrama de bloques para el modo PID local. Elaboración propia. 104

Figura 58: Declaración de la unión para convertir los valores de las variables de tipo float a formato en bytes. Elaboración propia. 107

Figura 59: Función para la lectura de datos desde Simulink. Elaboración propia. 108

Figura 60: Diagrama de bloques para modo Simulink. Elaboración propia. 109

Figura 61: Diagrama de los modos de control de la maqueta. Elaboración propia. 109

Figura 62: Archivo interactivo MATLAB/Simulink. Elaboración propia. 112

Figura 63: Modelo realizado en Simulink. En la parte superior, se emplean funciones de transferencia, mientras que en la parte inferior se ha realizado el diseño con el modelo no linealizado del sistema. Elaboración propia. 114

Figura 64: Casilla para elegir la preferencia de los datos de entrada en Simulink en lugar de emplear el HMI. Elaboración propia. 116

Figura 65: Visualización durante el periodo de simulación. Elaboración propia. 117

Figura 66: Ejecución del script al finalizar la simulación. Elaboración propia. 119

Figura 67: extracto del código ".ino". Elaboración propia. 158

Figura 68: Soporte para pantalla LCD 1602 I2C del interfaz Humano – Máquina. Elaboración propia. 159

Figura 69: Versiones anteriores del HMI con el codificador integrado, junto a la versión actual. Elaboración propia. _____ *159*

Figura 70: Soporte para el codificador incremental (encoder) KY – 040. Elaboración propia. ____ *160*

Figura 71: Soporte para motor JGA25 – 370 con reductora. Elaboración propia. _____ *160*

Figura 72: OPCIONAL, Soporte para módulos de control. Elaboración propia. _____ *161*

Figura 73: Previsualización archivos Gerber. Elaboración propia. _____ *162*

ÍNDICE DE TABLAS

Tabla 1: Configuración de las entradas en el puente H para determinar el sentido de giro y velocidad de la carga a la salida. *29*

Tabla 2: Elementos de la maqueta desarrollada hasta ahora. *53*

Tabla 3: Características del motor de corriente continua. *54*

Tabla 4: Comparativa de la actualización del microcontrolador en la maquetra de prácticas. *66*

Tabla 5: Comparativa de la actualización del puente H en la maqueta de prácticas. *70*

Tabla 6: Hoja de datos del motor de corriente continua JGA25-370, 12VDC 170RPM con reductora y codificador incremental de efecto Hall. *75*

Tabla 7: Características mecánicas y eléctricas del motor de corriente continua. *76*

Tabla 8: Resolución de los pines de lectura analógicos del conversor analógico digital. *78*

Tabla 9: Rango de tensiones permitidas para lectura del conversor analógico digital. Atenuaciones. *79*

Tabla 10: Tensiones y corriente que atraviesa el motor DC para la adecuación de los datos. Valores amplificados por el operacional. *81*

Tabla 11: Función de los pines correspondientes al microcontrolador ESP32 ..... *85*

Tabla 12: Número de bits por cada formato compatible con MATLAB. *106*

Tabla 13: trama de datos correspondiente al microcontrolador ESP32. ..... *107*

Tabla 14: trama de datos correspondiente a Simulink. *108*

Tabla 15: Parámetros de partida motor JGA25-370. ..... *113*

Tabla 16: Actualización de los valores del modelo motor DC. 120

Tabla 17: : Componentes empleados para la construcción de la placa electrónica con los enlaces de compra respectivos. 145

CAPÍTULO 1:

INTRODUCCIÓN Y OBJETIVOS

1. INTRODUCCIÓN Y OBJETIVOS

1.1 INTRODUCCIÓN

Los lazos de control son la base detrás de la gran mayoría de los procesos industriales. Permiten supervisar y controlar variables físicas como la velocidad, posición, corriente, tensión o temperatura, con el objetivo de mantenerlas en valores deseados e imperturbables ante factores externos. Por ello, es necesario conocer el funcionamiento y diseño de estos sistemas en el ámbito de la ingeniería, especialmente en disciplinas como la automatización.

Uno de los dispositivos más recurrentes a utilizar en el ámbito industrial es el motor eléctrico, presente desde líneas de producción hasta aplicaciones robóticas, pasando por tareas de transporte, manipulación y energía. En particular, los motores de corriente continua destacan por su fácil operabilidad, comportamiento dinámico predecible y compatibilidad con la mayoría de los sistemas de control. Son por ello una buena herramienta no solo para aplicaciones reales, sino también didácticas, donde se busca enseñar y experimentar con diferentes técnicas de control en tiempo real.

El constante avance de la electrónica embebida junto con softwares que implementan herramientas de cálculo y simulación (MATLAB) han abierto nuevas posibilidades de diseño y validación de estos sistemas, especialmente en entornos de desarrollo y educativos. Las maquetas de carácter didáctico permiten a los alumnos comprender el funcionamiento de estos y poner en práctica técnicas de control como los algoritmos PID.

Este proyecto se enmarca en la actualización de una maqueta para prácticas ya existente, inicialmente desarrollada a partir de un microcontrolador PIC, junto con un entorno propio de desarrollo. Debido a las limitaciones de potencia, conectividad y respuesta de cálculo, se propone el rediseño completo del sistema mediante una plataforma moderna basada en el ESP32, un microcontrolador de doble núcleo con conectividad WiFi y Bluetooth, capaz de comunicarse de manera eficiente con el entorno de simulación Simulink (MATLAB) a través del protocolo de comunicación UART. Ambas maquetas recopilan la información de un motor de corriente continua con reductora y encoder.

El rediseño, además de presentar mejoras de software y electrónicas, contempla la posibilidad de añadir un control PID tanto de manera interna (en el microcontrolador) como externa (en Simulink), ampliando las posibilidades de control y el análisis en lazo abierto y cerrado frente a la antigua maqueta.

1.2 JUSTIFICACIÓN Y MOTIVACIÓN

Este proyecto tiene como propósito proporcionar una herramienta educativa más robusta y flexible para el aprendizaje en las prácticas "Fundamentos de Automática", además de trabajar en tiempo real con los parámetros que el sistema ofrece. MATLAB es un entorno ampliamente usado por los estudiantes, y establecer una compatibilidad de comunicación directa con la maqueta facilita en gran medida el trabajo para ellos. Por otro lado, al integrar un control PID interno (local) es posible adaptar el uso de la maqueta a otras áreas de estudio, como control digital.

El desarrollo de este trabajo ha supuesto un desafío técnico en cuando a conocimiento y dominio de las múltiples herramientas empleadas (Fusion360, KiCAD, MATLAB/Simulink, Arduino IDE). La motivación de este proyecto se origina a partir de experiencias vividas a lo largo de la carrera. Comenzó a partir de haber cursado varias de las asignaturas como Mecatrónica, Instrumentación Electrónica, Analógica y Métodos y Herramientas para el Diseño Electrónico.

Finalmente, ha supuesto un reto de autoaprendizaje, con una estructura diferente a lo que uno está acostumbrado académicamente en el grado. Además, la idea de facilitar a los futuros estudiantes la comprensión y el estudio de un tema tan importante en la rama de automatización ha sido muy estimulante, comprometiéndome así de realizar un proyecto capaz de ser más intuitivo para el usuario en cuanto a interfaz y uso de este.

1.3 OBJETIVOS

El objetivo principal de este trabajo es desarrollar una maqueta didáctica funcional para el control de un motor de corriente continua utilizando un ESP32 como controlador principal. Esta maqueta debe ser capaz de comunicarse con Simulink en tiempo real a través del puerto serie (UART), permitiendo el control desde el entorno de simulación como desde el propio microcontrolador. Además, se busca que el sistema pueda cambiar entre controladores PID internos y externos, y que permita analizar el comportamiento del motor bajo diferentes configuraciones de lazo abierto y cerrado, realizando especial hincapié en la velocidad como variable de estudio. Partiendo de una maqueta ya existente

y cuyas características están quedando obsoletas, este trabajo pretende sustituir las funciones que esta desempeña, mejorando la velocidad de respuesta de datos y accesibilidad al usuario, entre otras.

1.3.1 OBJETIVOS ESPECÍFICOS

- Sustituir la antigua arquitectura basada en el microcontrolador PIC por una solución más moderna con un ESP32, mejorando así el rendimiento del sistema y velocidad de cálculo.
- Diseñar y fabricar una placa electrónica que incorpore este nuevo microcontrolador con ayuda del sofware de diseño "KiCAD" además de los componentes necesarios para el correcto control del motor y una interfaz intuitiva.
- Desarrollar el software de control en el entorno Arduino IDE, incluyendo la lectura y tratamiento de datos procedentes de los periféricos (encoder del motor, encoder rotativo, pantalla LCD I2C y doble puente H).
- Añadir al sistema una opción de selecciones de modos de control (integrado en la propia maqueta o en Simulink) para facilitar el análisis comparativo.
- Diseñar e imprimir en 3D los elementos estructurales necesarios para la creación de la nueva maqueta.
- Evaluar el comportamiento del sistema tanto en lazo abierto como cerrado, recogiendo los datos más relevantes.
- Documentar todo el proceso, desde el diseño electrónico, programación, simulación y validación, analizando los resultados obtenidos.
- Ofrecer una configuración basada en módulos comerciales como alternativa a la fabricación de la placa y comparar los costes económicos.

CAPÍTULO 2:

FUNDAMENTOS TEÓRICOS

2. FUNDAMENTOS TEÓRICOS

Antes de profundizar en el desarrollo de la maqueta, es conveniente repasar algunos de los conceptos básicos que sustentan este trabajo. A continuación, se presentan de manera resumida los temas a tratar en este proyecto:

2.1. FUNDAMENTOS DE LA IDENTIFICACIÓN DE SISTEMAS

El control de sistemas permite regular y establecer el comportamiento de dispositivos y procesos físicos. Su principal objetivo es garantizar que una variable de salida adquiera el valor de una consigna deseada a lo largo del tiempo, independientemente de las perturbaciones externas al sistema y asegurando estabilidad y precisión. Esto es posible a través del análisis y diseños de algoritmos que, a partir de una señal de referencia (o consigna) y la medición de la variable controlada, generan una acción de control adecuada para alcanzar los requisitos buscados. Prácticamente la mayoría de los procesos industriales requieren de algún tipo de control para operar correctamente [40].

El lazo de control es la estructura más representativa de los sistemas de regulación automática. Esta se compone de los siguientes elementos (ver figura 1):

- **Planta/Proceso/Sistema:** es el sistema físico que se desea controlar (en este trabajo, será un motor de corriente continua). La señal que entra al bucle se denomina **variable manipulada** y es aquella en la que el usuario modifica de forma directa para observar su efecto sobre otra variable. A la salida de la planta, obtenemos la variable controlada o de salida, la cual es realimentada y traducida para ser comparada por la señal de referencia (consigna)
- **Controlador/Regulador:** genera una señal de control adecuada (**variable controlada**) en función de la señal de error introducida. Esta se calcula como la diferencia entre la salida y la señal de entrada o **variable deseada**.
- **Actuador:** traduce la señal de control del regulador en una acción física sobre el proceso.
- **Medidor/Sensor:** mide la variable de salida del proceso.

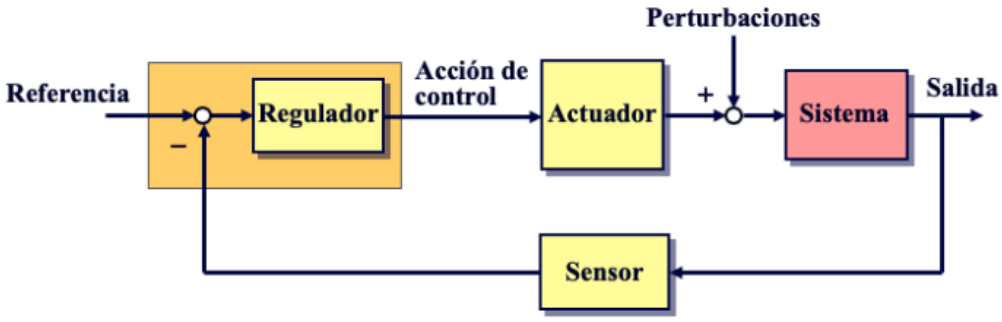


Figura 1: Representación de un lazo de control (correspondiente a un control de lazo cerrado). Extraído de [1].

2.2. PRINCIPIOS DEL CONTROL PID

La gran mayoría de los sistemas automatizados en ingeniería se diseñan con el fin de que un proceso siga una referencia deseada o consigna, garantizando en medida de lo posible la precisión y estabilidad de esta. Para ello, existen dos maneras de abordar el control: lazo abierto y lazo cerrado.

2.2.1. CONTROL EN LAZO ABIERTO

El control en lazo abierto es aquel en la que la acción de control está determinada única y exclusivamente por la entrada o consigna, independientemente de la respuesta real del sistema (ver figura 2). No existe una medición de la salida para corregir a la entrada las posibles desviaciones o perturbaciones (no hay una realimentación) [40]. Un ejemplo básico puede ser el funcionamiento de un horno microondas sencillo: el electrodoméstico no tiene la capacidad de medir la temperatura real de la comida en su interior. Funciona únicamente según las condiciones iniciales del sistema y el valor de entrada o consigna (en este caso, será el temporizador del microondas). Este calentará su interior durante el tiempo establecido por la consigna, independientemente del contenido o temperatura inicial.

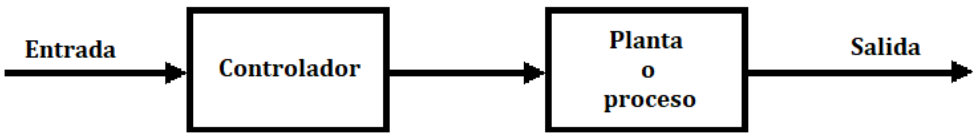


Figura 2: Representación de un control en lazo abierto. Elaboración propia.

Este tipo de sistemas asume que el comportamiento del proceso es completamente predecible y no sufre alteraciones externas. Aunque este tipo de control es sencillo y económico, la principal desventaja reside en la incapacidad de adaptación frente a cambios en el entorno (se desenchufa de la corriente, se vacía el interior estando en marcha, la temperatura exterior cambia…).

2.2.2. CONTROL EN LAZO CERRADO

El control en lazo cerrado incorpora una retroalimentación de la salida real del sistema, con el fin de ajustar de manera dinámica la acción de control y así reducir las diferencias entre consigna y salida real (ver figura 3), logrando una mayor precisión y adaptabilidad frente a cambios o perturbaciones [41]. Un ejemplo puede ser la calefacción de una vivienda (el termostato): si la temperatura exterior es inferior al valor establecido, el sistema incrementa el calor. En caso contrario, permanece en reposo hasta que la temperatura se reduzca. Así, permite mantener la variable controlada dentro de los valores deseados.

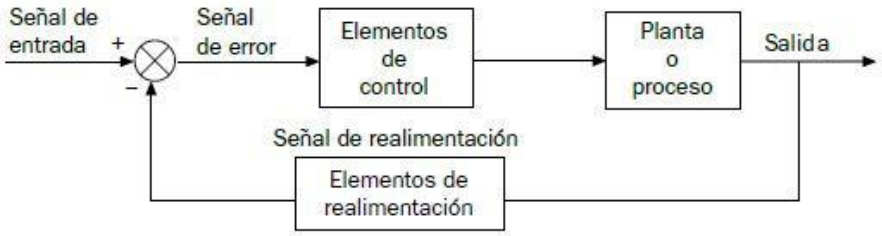


Figura 3: Representación de un control en lazo cerrado. Extraído de [2].

Los sistemas en lazo cerrado se emplean en situaciones donde se requiere una alta precisión, estabilidad y respuesta ante perturbaciones. Los sistemas en lazo abierto en cambio pueden ser adecuados para procesos altamente repetitivos, que no son críticos y buscan minimizar costes.

2.2.3. INTRODUCCIÓN AL CONTROL PID

En el interior de casi todos los sistemas de control automático modernos, hay un algoritmo que lleva más de 100 años aportando estabilidad a procesos dinámicos. Es tan utilizado como subestimado, y tan simple de entender como difícil de dominar en la práctica. Se trata del control PID, un sistema que, en esencia, busca corregir errores y reducir al máximo la influencia de las perturbaciones en el sistema. Las siglas PID

provienen de tres términos: Proporcional (P), Integral(I) y derivativo (D). Estos términos describen cómo responde el sistema al error. Su comportamiento tiene una lógica bastante intuitiva. Aunque suene complejo al principio, es en realidad un tipo de control automático muy utilizado en sistemas industriales.

Este mecanismo trabaja en un ciclo constante: detecta la diferencia (el error) entre un valor real (lo que está ocurriendo) y un valor deseado (lo que queremos que ocurra). Ese error puede parecer pequeño, pero incluso las variaciones mínimas pueden ser críticas en sistemas delicados.

Entonces, ¿qué hace el PID? Aplica una corrección inteligente basada en tres "términos":

- **Proporcional**, que responde directamente a cómo de grande es el error.
- **Integral**, que toma en cuenta cuánto tiempo ha estado presente ese error (parecido a un acumulador).
- **Derivativo**, que analiza cómo de rápido está siendo modificado ese error.

La combinación de estos tres factores permite que el sistema se ajuste de manera precisa, suave y continua, buscando siempre reducir la diferencia entre lo real y lo ideal.

Por ejemplo: un usuario conduce un coche por una carretera y quiere mantener una velocidad constante. Si el vehículo reduce su velocidad, el conductor presiona el acelerador. Si esta diferencia de velocidad es mayor, el conductor lo presionará con una fuerza mayor. Eso es exactamente la acción proporcional (P). Por otro lado, si el conductor permanece por debajo de la velocidad deseada a pesar de pisar el acelerador durante un tiempo, es evidente que hay que ajustar otros parámetros. El conductor acumula ese error. Esta acumulación es la acción integral (I). Si por el contrario el conductor siente que se está acercando demasiado rápido a la velocidad objetivo, este levanta el pie del acelerador. Se anticipa a la situación. Esto es la acción Derivativa (D). (Ver figura 4)

Figura 4: Ejemplo PID con el acelerador de un coche. Extraído de [3].

Un controlador PID replica este razonamiento humano, pero de forma precisa, continua y sin fatiga física. La fórmula del PID como expresión en el dominio del tiempo es la siguiente:

$$u(t) = K_p \cdot e(t) + K_i \cdot \int_0^t e(\tau)d\tau + K_d \cdot \frac{de(t)}{dt}$$

- *u(t)* es la señal de salida del controlador (acción de control)
- *e(t)* es el error en el tiempo, definido como *e(t) = r(t)-y(t)*, siendo *r(t)* la referencia deseada e *y(t)* la salida real del sistema.
- K_p, K_i, K_d son las ganancias proporcional, integral y derivativa respectivamente.

Cada término tiene un propósito:

- Proporcional "$K_p \cdot e(t)$" responde al valor actual del error. Cuanto mayor sea el error, mayor será la corrección.
- Integral "$K_i \cdot \int_0^t e(\tau)d\tau$" responde a la suma del error en el tiempo. Esto ayuda a eliminar el error residual (offset, error estacionario) que no es posible corregir únicamente con la parte proporcional.
- Derivativa "$K_d \cdot \frac{de(t)}{dt}$" predice el comportamiento futuro del error en función de su velocidad de cambio. Es capaz de suavizar la respuesta, amortiguando las oscilaciones del sistema.

Una de las ventajas más claras de los controladores automáticos es que no solo actúan, sino que también indican cómo se encuentra el sistema en tiempo real. Es decir, te muestran el valor actual del parámetro que están regulando. En muchos casos, estos controladores se colocan cerca del punto donde se mide la variable, lo cual evita tener

que usar transmisores adicionales para enviar señales. Menos complicaciones, menos margen de error.

En el dominio de Laplace:

A la hora de modelar un sistema, es muy común trabajar en el domino de Laplace, debido a que facilita los cálculos algebraicos con las funciones de transferencia. En este caso, el PID en este dominio se representa de la siguiente manera:

$$U(s) = \left(K_p + \frac{K_i}{s} + K_d \cdot s\right) \cdot E(s)$$

Donde U(s) es la señal de salida del controlador y E(s) es la entrada de error en el sistema. Permite conectar el controlador con el resto del sistema (planta) para analizar la respuesta transitoria, estabilidad y errores en régimen permanente.

Ejemplo: Horno eléctrico

Se quiere controlar la temperatura de un horno. Se dispone de un sensor que recoge la temperatura actual "y(t)" y se requiere de un valor de "r(t)" 200ºC. El error del sistema será la diferencia entre el valor deseado "r(t)" y el medido "y(t)".

Si se emplea únicamente la acción proporcional, el sistema reaccionará rápido al error, pero en régimen permanente existirá un error estacionario (constante). Al agregar el término integral, este error desaparecerá. Si además se incluye el término derivativo, será posible evitar sobrepasar los 200ºC cuando el horno comience a calentarse muy rápido.

Esta es la funcionalidad del PID. Cuando los tres componentes se ajustan correctamente, se obtienen resultados es estables y precisos.

Ajuste y sintonización de los PID

No hay una única forma de ajustar los parámetros *Kp, Ki* y *Kd* de estos controladores. Es posible sintonizarlos mediante prueba y error, aplicando reglas empíricas (Ziegler-Nichols, Cohen Coon) o técnicas más avanzadas como la optimización numérica o el control adaptativo.

Si el término *Kp* aumenta en exceso, el sistema adquiere un comportamiento oscilatorio, e incluso inestable. Si, por el contrario, el término *Ki* es muy alto, se introduce

un retardo significativo en la respuesta. Si *Kd* es excesivo, el sistema puede reaccionar de manera errónea ante pequeños cambios o ruidos en el sensor. La clave reside en encontrar un equilibrio entre los tres parámetros.

Como conclusión, el PID es un sistema que data del siglo XX, y que aun así sigue siendo una técnica eficaz, robusta y versátil en aplicaciones de control industrial (siempre que se sintonice correctamente) [41].

2.3. MODELADO DE MOTORES DE CORRIENTE CONTINUA

Los motores DC o de corriente continua son dispositivos que convierten la energía eléctrica en energía mecánica por medio de la interacción de los campos magnéticos generados por el bobinado en su interior. Se rigen por el principio de la ley de Lorentz, que describe la fuerza a la que se somete un conductor en presencia de un campo magnético cuando circula una corriente a través de él [4]. Su expresión es la siguiente:

$$\vec{F} = I\vec{L}\, x\, \vec{B} \qquad (2.1.1)$$

La fuerza es el resultado del producto vectorial entre la longitud del conductor y el campo magnético al que es sometido multiplicado por la corriente que circula a través de este. El vector resultante es siempre perpendicular a ambos (ver figura 5):

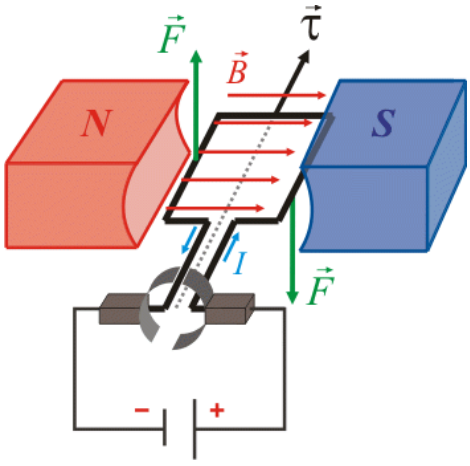


Figura 5: Esquema conceptual de motor eléctrico DC. Extraído de [4].

Es uno de los dispositivos más versátiles de la ingeniería moderna. Desde juguetes hasta satélites, pasando por ascensores, robots, sistemas de ventilación, etc. Los motores DC forman parte de muchas aplicaciones físicas, y poseen la capacidad de responder rápidamente a cambios en la tensión de alimentación, lo que los convierte en un candidato

ideal para sistemas donde se requiere controlar con precisión la velocidad o la posición del eje.

A diferencia de otros tipos de motores (como los de inducción o paso a paso), el motor DC responde de manera prácticamente instantánea a la tensión que recibe. Esto lo hace predecible y, sobre todo, controlable.

Estructura básica de un motor DC:

- Rotor (o armadura): es la parte móvil, que incluye las bobinas por las que circula la corriente.
- Estátor: lugar donde se genera el campo magnético, ya sea mediante imanes permanentes o electroimanes.
- Conmutador: un sistema mecánico (presente en motores con escobillas) que invierte el sentido de la corriente en las bobinas para preservar el giro del motor.
- Escobillas: transmiten la corriente desde el exterior al conmutador. Suelen estar fabricadas a base de carbón/grafito. Al estar en contacto directo con el rotor, producen un desgaste con el tiempo, y deben ser reemplazas para que el motor gire correctamente.
- Eje: la salida mecánica del sistema, que transmite el movimiento rotacional.

El devanado está formado por bobinas aisladas entre sí, dispuestas alrededor del núcleo de la armadura

Existen otras versiones de motores DC denominadas “brushless”, cuyo principio de funcionamiento es similar, pero se eliminan las escobillas, mejorando su durabilidad y eficiencia. Por otro lado, el control de estos motores se vuelve más avanzado. Es necesario analizar las necesidades de nuestro sistema para elegir correctamente el tipo de motor adecuado.

Modelo matemático de un motor DC:

Son adecuados para realizar modelos de simulación en base a sus expresiones matemáticas. En el caso de este proyecto, se seleccionan unas condiciones específicas que simplifican en gran medida los términos, resultando así en un modelo más sencillo (que veremos más adelante). El modelo matemático se puede dividir en dos secciones: el circuito eléctrico y el sistema mecánico:

1. **Expresión matemática del modelo eléctrico**:

$$u_m(t) = R_m i(t) + L_m \frac{di(t)}{dt} + K_b \omega(t) \qquad (2.1.2)$$

Donde u_m es la tensión aplicada al motor (en voltios V), $i(t)$ es la corriente que circula por la armadura (en amperios A), L_m es la inductancia del devanado (en henrios H), R_m la resistencia de la armadura (en ohmnios Ω), K_b es la constante de fuerza electromotriz (medido en V·s/rad) y $w(t)$ es la velocidad angular del eje (en rad/s).

El último término de la expresión "$K_b w(t)$" representa la fuerza contraelectromotriz: una tensión que se genera de forma instantánea en sentido contrario al avance del motor.

2. **Expresión matemática del modelo mecánico**:

$$K_m i(t) = J_m \frac{d\omega(t)}{dt} + B_m \omega(t) \qquad (2.1.3)$$

Donde J_m es el momento de inercia del motor (expresado en Kg·m^2), B_m es el coeficiente de fricción viscosa (N·m·s) y K_m es la constante del par motor (ver figura 6).

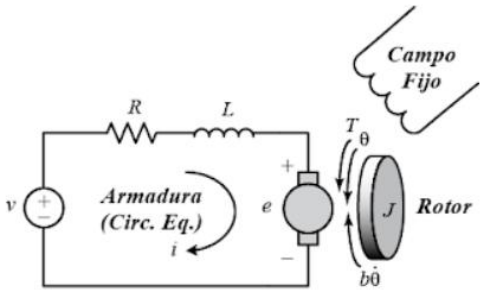


Figura 6: Modelo de simulación para un motor DC. Extraído de [5].

Partiendo de unas condiciones concretas, como que el valor de la inductancia del devanado es mucho menor que la resistencia de la armadura es posible despreciar el término de la autoinducción (L_m). Además, el motor que se seleccionará para este proyecto posee una reductora interna, donde su momento de inercia (J_{eq}) es muy similar al momento de inercia del motor (J_m). Por tanto, las expresiones se reducen a las siguientes:

$$V_m(t) - K_m w_m(t) = R_m I_m(t) \qquad (2.1.4)$$

$$T_m(t) = K_m I_m(t) \qquad (2.1.5)$$

$$T_m(t) - nT_c(t) = J_{eq}\frac{dw_m}{dt} \qquad (2.1.6)$$

Una vez obtenidas las ecuaciones reducidas, trasladamos el modelo al dominio de Laplace. Se trata de un sistema MISO (del inglés: Multiple Inputs, Single Output). Es posible recrear la respuesta del sistema a partir de un espacio de estados, pero en este caso, realizamos las funciones de transferencia correspondientes, anulando en cada sección una de las entradas, para posteriormente unir todo el sistema.

$$V_m(s) - K_m w_m(s) = R_m I_m(s) \qquad (2.1.7)$$

$$T_m(s) = K_m I_m(s) \qquad (2.1.8)$$

$$T_m(s) - nT_c(s) = s \cdot J_{eq} w_m(s) \qquad (2.1.9)$$

Inicialmente, anulamos una de las entradas: $T_c=0$ (*cte*). La función de transferencia resultante es la siguiente:

$$G_v(s) = \frac{w_m(s)}{V_m(s)} = \frac{K_m}{R_m J_{eq} s + {K_m}^2} \qquad (2.1.10)$$

Se establece V_m a 0 y resolvemos de nuevo la función de transferencia para la otra entrada:

$$G_{T_c}(s) = \frac{w_m(s)}{T_c(s)} = \frac{-n}{J_{eq}s + \frac{{K_m}^2}{R_m}} \qquad (2.1.11)$$

Completamos el sistema unificando ambas funciones de transferencia (ver figura 7):

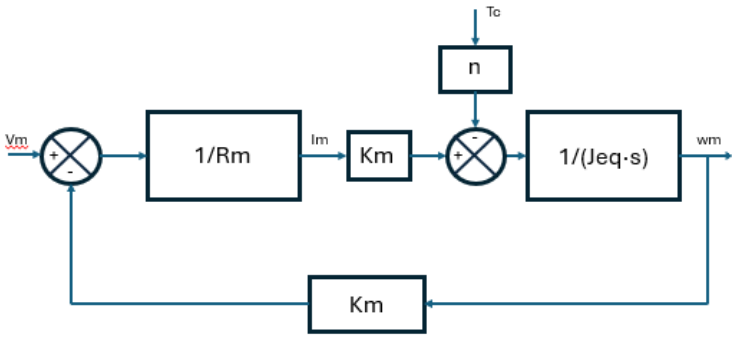


Figura 7: Función de transferencia del sistema. Elaboración propia.

Una vez obtenido el modelo de simulación para el motor, sólo es necesario introducir los datos específicos del motor que se desee modelar, los cuales vienen dados por el fabricante de cada motor.

Respuesta del motor sin control:

Si se aplica una tensión constante al motor y no se emplea ningún tipo de control, este se comportará como un sistema de primer o segundo orden con retardo, en función

de su diseño. Esto puede dar lugar a un sobreimpulso inicial, provocar un retardo en alcanzar la velocidad objetivo o errores en régimen permanente [5].

2.4. CONTROL DE UN MOTOR ELÉCTRICO: EL PUENTE H

Si el controlador PID se asemeja al cerebro del sistema, el puente H es sin duda el corazón eléctrico de este. Es el circuito que traduce las decisiones del controlador en magnitudes físicas (ej: corriente que atraviesa el motor para lograr el giro, detenerlo o cambiar su dirección).

Un puente H es un circuito electrónico de potencia que permite aplicar una tensión positiva o negativa de forma controlada a los terminales de un motor eléctrico. Debido a sus características, hace posible cambiar el sentido de giro del motor. Además, si se combina con señales PWM (Modulación por Ancho de Pulso), es capaz de controlar la velocidad de giro. Se denomina puente H debido a su diseño, porque se parece literalmente a la letra "H". El motor se sitúa en el centro, y los cuatro interruptores (transistores o MOSFETs) están dispuestos en los extremos verticales de la letra (ver figura 8).

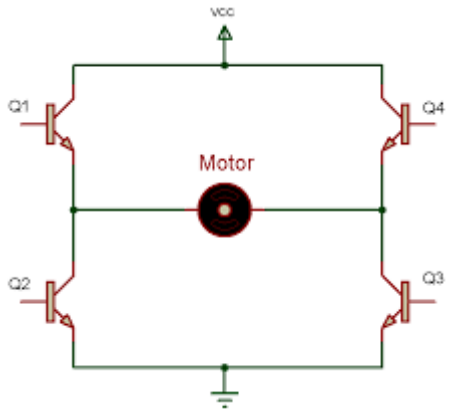


Figura 8: Ilustración de un puente H diseñado en la aplicación KiCAD. Elaboración propia.

Al accionar *Q1* y *Q4* (se cierra el circuito), el motor gira en un sentido. En cambio, al accionar *Q2* y *Q3*, el motor gira en sentido contrario. Si ninguno de los transistores está cerrado, entonces el motor es capaz de girar libremente sin potencia. Este dispositivo permite modificar así el sentido de giro del motor sin la necesidad de intercambiar los cables manualmente.

Ventajas que ofrece el puente H

- Control total de la dirección solamente aplicando dos señales lógicas

- Control de la velocidad a partir de la modulación PWM (Pulse Width Modulation).
- Compatible con microcontroladores, gracias a que puede enviar señales de baja tensión.

Gracias al puente H, un microcontrolador como Arduino o un sistema embebido es capaz de realizar un control sobre motores de mucha potencia como si se tratara de un pequeño LED.

Control de sentido y velocidad con PWM

Regular la velocidad de giro del motor es posible modulando el ancho de pulso que se envía a dicho dispositivo. Es decir, en lugar de aplicar una tensión fija, se envía una señal cuadrada con ciclos de encendido y apagado muy rápidos [6]. (Ver figura 9).

$$V_{motor} = V_{DC} \cdot D \qquad (2.1.12)$$

Donde "*D*" es el ciclo de trabajo de la señal cuadrada:

- *D = 1* → motor girando a máxima velocidad (la que permite la tensión V_{DC}).
- *D = 0.5* → el motor gira a la mitad de su velocidad ($V_{DC}/2$).
- *D = 0* → el motor está apagado.

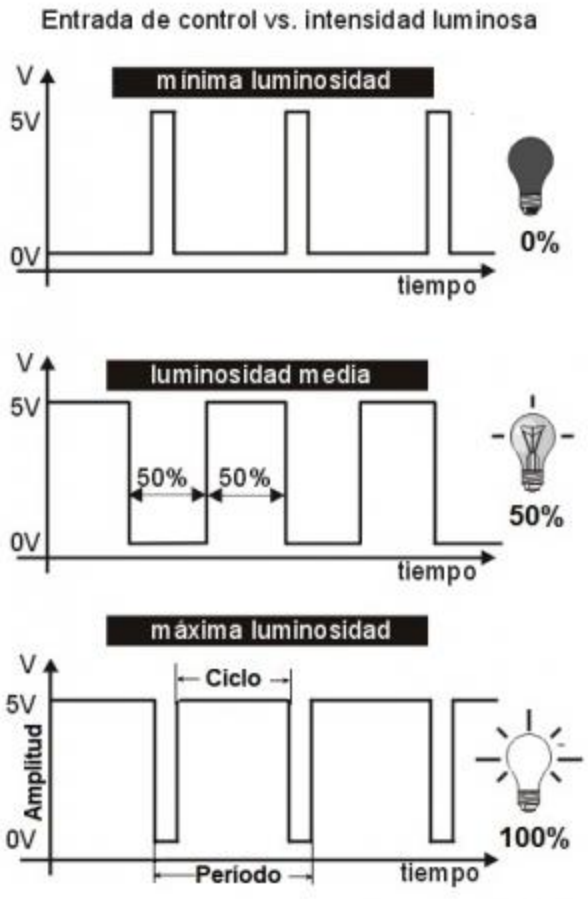


Figura 9: Ilustración del funcionamiento de la señal PWM con una bombilla LED. Extraído de [6].

Al combinar ambos aspectos del puente H, podemos controlar el sentido (con dos pines de dirección) y la velocidad de giro (con una señal PWM) de cualquier motor DC.

Tabla 1: Configuración de las entradas en el puente H para determinar el sentido de giro y velocidad de la carga a la salida.

Sentido A	Sentido B	PWM	Resultado
0	0	-	Freno libre (el motor gira manualmente)
0	1	50%	Giro antihorario al 50% de velocidad
1	0	80%	Giro horario al 80% de velocidad
1	1	-	Freno activo (stall)

Relación con el controlador PID

El puente H no reemplaza al PID, sino que ejecuta las decisiones de este. Supongamos que el PID calcula una salida de control de 2.5V para mantener 10 rpm. Esta

señal se convierte en un ciclo de trabajo (D) del 20%, que el puente H aplicará al motor con la polaridad adecuada, según la dirección que se desee. Si el error se modifica, el PID también lo hace con su salida. El puente H ajusta el PWM y polaridad. En conjunto, forman un control preciso para los motores de corriente continua.

A lo largo del capítulo se ha analizado tanto la teoría como la práctica del control. Se ha explicado cómo un motor de corriente continua puede convertirse en un elemento fundamental en un sistema de control de alta precisión, no sólo gracias a su diseño electromecánico, sino al controlador PID y el puente H, que permite ejecutar cada orden con exactitud. Controlar un motor no sólo es hacerlo girar. Debe hacerlo con sentido, alcanzar una velocidad determinada y mantenerla, cambiar de dirección sin perder el control. Se trata de traducir intenciones abstractas – como "avanzar más rápido", "detenerse ahora" o "corregir ese desvío" – en impulsos eléctricos, señales digitales y respuestas físicas.

Ahora que conocemos a fondo los principios que sustentan al control de motores DC – modelos, comportamiento dinámico, respuesta a señales y control inteligente mediante PID y puente H, es posible aplicar estos conocimientos en el mundo real. Un ejemplo bastante conocido son los AGVs (Vehículos de Guiado Automático) de la fábrica Renault en Valladolid, los cuales automatizan la logística interna. Cada AGV incorpora motores de corriente continua para tracción y dirección. Para lograr una velocidad constante, con cambios suaves y de forma segura, se emplean los lazos de control PID. Esto permite minimizar el error de seguimiento de la trayectoria y una mayor eficiencia y seguridad en la planta.

En el siguiente capítulo se introducirá la maqueta previa a este trabajo, que ha servido como entorno de pruebas y validación de todo lo que se ha desarrollado hasta ahora en el trabajo. Desde el montaje físico hasta la integración electrónica, con el fin de analizar qué aspectos de la maqueta son necesarios actualizar.

2.5. COMUNICACIÓN ENTRE MICROCONTROLADORES Y PERIFÉRICOS

2.5.1. El ECOSISTEMA ARDUINO

Como ya se mencionó antes, Arduino es una de las bases más importantes en este proyecto. Cuando se habla de electrónica accesible y creativa, es casi imposible no pensar en esta plataforma. En este apartado, vamos a adentrarnos en las razones por las que Arduino se ha convertido en una de las opciones más populares en el mundo de la tecnología y la innovación (ver figura 10).

Figura 10: Logotipo del entorno de desarrollo y placas electrónicas Arduino. Extraído de [7].

2.5.2. ARDUINO

Arduino es mucho más que una simple placa electrónica. Es una plataforma de hardware y software de código abierto, pensada para ser fácil de usar, incluso si nunca has manejado un cable o escrito una línea de código. Lo interesante de Arduino es que puede leer una entrada –como la luz captada por un sensor, la pulsación de un botón o incluso un mensaje que llega por Internet– y convertirla en una acción: encender un LED, mover un motor o enviar datos a la nube. Se programa en un lenguaje muy sencillo (basado en Wiring) y usando el entorno Arduino IDE (cuya base es el lenguaje de Processing). Una vez escrito el código, este se envía al microcontrolador integrado en la placa.

Arduino nació en 2005, en el Instituto de Diseño de Interacción de Ivrea, Italia. Al principio fue solo una herramienta para que los estudiantes –muchos sin experiencia previa en electrónica– pudieran prototipar ideas rápidamente. Sin embargo, lo que comenzó como algo modesto creció a una velocidad inesperada. Hoy en día, Arduino no se limita a proyectos pequeños. Se utiliza en dispositivos IoT, wearables (dispositivos vestibles), impresoras 3D, sistemas embebidos y mucho más. Su capacidad para adaptarse

lo ha convertido en el corazón de una infinidad de creaciones, desde objetos cotidianos hasta complejos instrumentos científicos [31].

El aspecto más importante que destacar es la comunidad que tiene detrás. Arduino es, en esencia, una plataforma de hardware libre y código abierto, donde estudiantes, entusiastas, artistas, programadores y profesionales de todo el mundo han formado una red viva y activa que comparte conocimientos, ideas y soluciones. Gracias a este espíritu colaborativo, Arduino sigue evolucionando y creciendo cada día. Se asemeja a una gran conversación global donde todos aportan algo.

2.5.3. JUSTIFICACIÓN DEL USO DE ARDUINO

El entorno de Arduino (Arduino IDE) destaca por gran accesibilidad de código abierto. Es sencillo de aprender, pero se puede complicar hasta tal punto como el usuario decida. Su software es intuitivo, ideal para dar los primeros pasos en la programación, pero lo suficientemente robusto como para que usuarios avanzados puedan desarrollar proyectos serios. Está disponible en cualquier sistema operativo: Windows, Mac o Linux.

Existen otras plataformas para trabajar con microcontroladores, como MicroPython, Parallax Basic Stamp, o Handyboard. Arduino IDE tiene una gran compatibilidad con la mayoría de los microcontroladores del mercado. Esto incluye a los ESP32. Arduino es una herramienta útil porque es posible programar manteniendo el mismo lenguaje de desarrollo que posee, pero empleando las placas de ESP32. En resumen:

- Se trata de una aplicación multiplataforma, gracias a su compatibilidad con varios sistemas operativos y placas de desarrollo.
- Es un entorno de programación sencillo, basado en Processing. Es intuitivo para usuarios principiantes y flexible para aquellos más avanzados.
- Se trata de código abierto, donde es posible crear tus propias librerías o adaptar el sistema a tus necesidades (al estilo C++).
- Tiene una gran comunidad, siendo un hardware abierto, donde los planos y placas están disponibles bajo una licencia “Creative Commons”. Cualquiera puede crear su propia versión o modificar una existente.

2.5.4. PROGRAMAR EN ARDUINO IDE

La gran mayoría de microcontroladores compatibles con Arduino IDE siguen la misma estructura física. Consisten en pequeñas placas de circuito impreso con las siguientes características:

- Entradas y Salidas digitales (I/O): son pines para leer señales o generar salidas de control.
- Entradas analógicas: son pines que integran un conversor analógico/digital para leer sensores de tensión variable.
- PWM (Modulación por ancho de pulso): generan señales de control cuadradas con una anchura de pulso variable (control de velocidad en motores DC, por ejemplo).
- Puertos de comunicación: estas placas suelen integrar conversores a USB, UART, I2C y SPI en la mayoría de las modelos para facilitar la comunicación con periféricos y el ordenador.

El Arduino IDE está basado en C/C++ , pero está simplificado para que cualquiera pueda programar con pocos comandos, y ofrece una interfaz relativamente sencilla para escribir código, compilarlo, subirlo al microcontrolador y monitorizar los datos vía puerto serie (Serial Monitor). Es un editor simple y directo, donde se escribe el código (sketch), se verifica y carga a la placa por USB. Además, permite integrar bibliotecas de terceros para expandir funcionalidades (control de motores, sensores, PID, etc.)[30].

Su estructura básica es la siguiente:

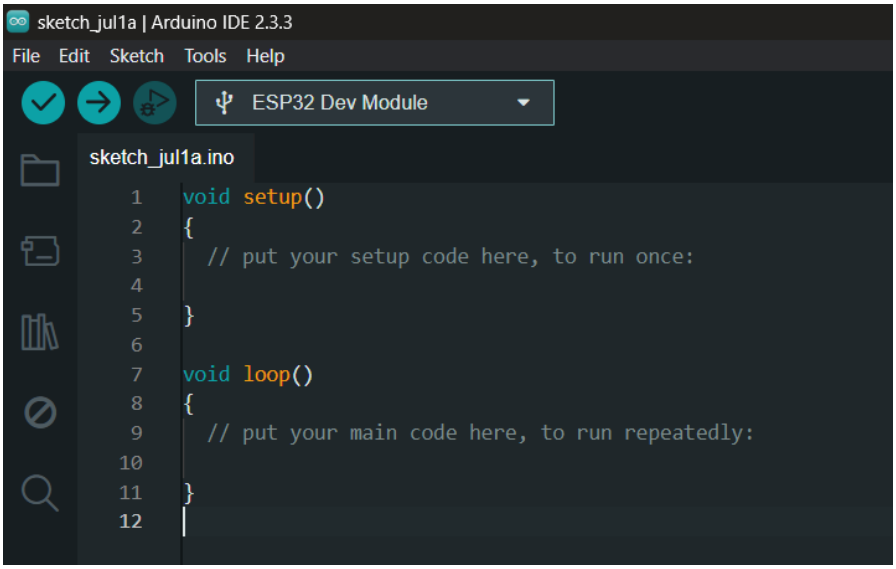


Figura 11: Interfaz del programa Arduino IDE. Elaboración propia.

Se compone de dos secciones (ver figura 11):

- void setup(): se trata del código de inicialización. Se encarga de configurar determinados pines, inicializa la comunicación serial y librerías. Esta sección se ejecuta una única vez.
- void loop(): Esta sección alberga el código principal del proyecto. Se repite de forma cíclica, una vez compilado y subido al microcontrolador.

Entradas y salidas de datos:

Lectura digital y analógica:

pinMode (x, INPUT);
int estado = digitalRead (x);
int valor = analogRead (Ay);

En las líneas de código mostradas, se detalla la configuración necesaria para establecer el pin "*x*" como una entrada de lectura digital. Para ello, es necesario establecer el modo del pin en "void setup()" como entrada o "INPUT". Posteriormente, si se desea realizar una lectura de forma repetitiva, se crea una variable de tipo entero que almacena el estado digital de lectura del pin "*x*" (*0* o *1*). Por otro lado, si se desea realizar lecturas de forma analógica, se debe escoger un pin determinado para ello. En la mayoría de los

microcontroladores hay secciones de patillas reservadas para la lectura analógica. La mayoría de estas comienzan con la letra “A”, seguido del número de pin.

Salida digital y PWM:

pinMode (x, OUTPUT);
digitalWrite (x, HIGH);
analogWrite (Ay, 128);

Para la salida digital es necesario configurar el pin deseado como “OUTPUT” o salida. Después, seleccionamos dicho pin y establecemos su a alto (HIGH) o bajo (LOW) correspondientes a 3.3V o a 0V. De la misma manera, es posible escribir el valor de una señal PWM en un pin con salida analógica. En este ejemplo, el microcontrolador empleado posee una resolución de 8 bits (256 valores, 0 a 255). Al establecerlo a 128, corresponde a una señal cuadrada con un ciclo de carga del 50%. En la actualización de la maqueta, la resolución máxima es de 12 bits, ampliando el rango así a 4095 valores posibles. Se debe tener en consideración que la lectura de los pines analógicos solo puede leer hasta 3.3 V directos al pin. Para tensiones mayores, es necesario realizar divisores de tensión. Por otro lado, es posible configurar la atenuación y resolución máxima de los mismos según el tipo de microcontrolador que se emplee. Para seleccionar un dispositivo compatible, es necesario ubicar el panel superior del entorno, ir a “Herramientas” o “Tools”, desplazar el cursor a “Placas” o “Boards” y elegir el microcontrolador. Posteriormente, se debe configurar el puerto COMx en el que está conectado el dispositivo (ver figura 12). Una vez realizadas las comprobaciones, sólo es necesario subir el código del programa y el microcontrolador lo almacenará y ejecutará de forma cíclica mientras esté alimentado [32].

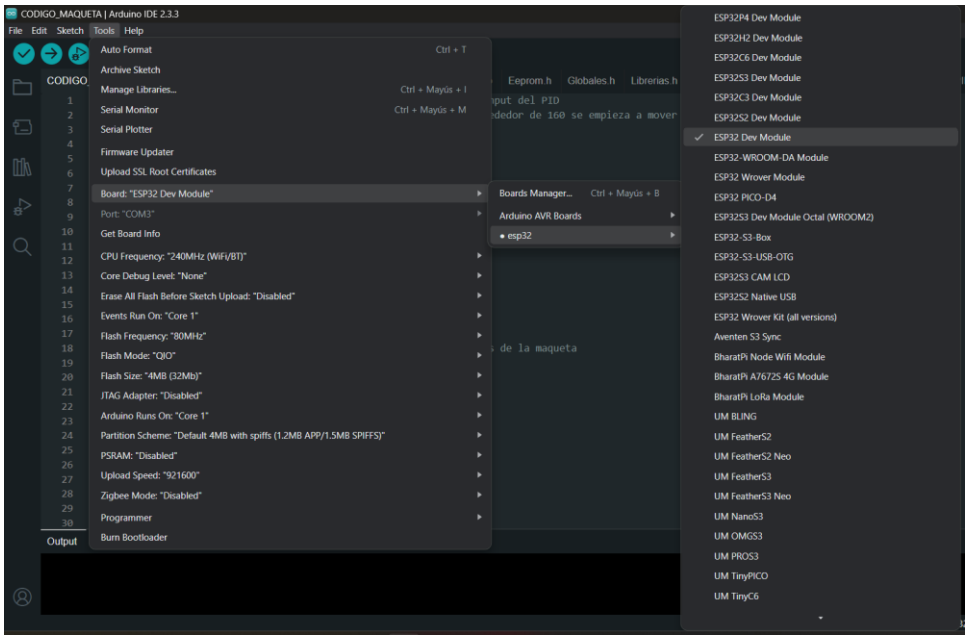


Figura 12: Selección del microcontrolador en el entorno de Arduino IDE. Elaboración propia.

Aunque Arduino nació como una herramienta educativa, se ha ganado un puesto en proyectos serios de automatización y prototipado industrial, como, por ejemplo:

- La domótica: el control de luces, alarmas, puertas automáticas o termostatos desde el teléfono móvil.
- Robótica: seguidores de línea, brazos mecánicos, drones, líneas de montaje.
- Monitorización de variables: estaciones meteorológicas, temperatura en invernaderos, control de la humedad en espacios cerrados.
- Interacción con otros sistemas: Arduino IDE es capaz de programar microcontroladores de otros entornos con la adaptación de sus librerías. Además, puede comunicarse con Raspberry Pi, PLCs o ESP32. Puede utilizar MQTT o protocolos como Modbus para la integración de sistemas SCADA.
- Industria 4.0: Los prototipos basados en Arduino suponen un coste económico mucho menor que el uso de PLCs profesionales. Actualmente, la potencia computacional de estos dispositivos es lo suficientemente potente para realizar el control/monitorización en entornos industriales.

En resumen, el gran valor de Arduino reside en su bajo coste, amplia comunidad y su filosofía de código abierto. Es compatible con una gran variedad de sensores en el mercado, actuadores y módulos de comunicación. Es un puente muy útil para relacionar la educación con la ingeniería aplicada.

2.5.5. EL ENTORNO MATLAB

MATLAB (ver figura 13), cuyo nombre viene de Matrix LABoratory (laboratorio matricial), empezó como un programa para realizar cálculos matriciales y álgebra lineal. En la actualidad, es un entorno de desarrollo que mezcla lenguaje de programación, simulación, diseño de interfaces y visualización de datos [32].

Figura 13: Software de desarrollo MATLAB. Extraído de [8].

Este programa se volvió popular gracias a la forma en la que simplifica sistemas y comportamientos complejos. Ingenieros, científicos y estudiantes lo usan para modelar, analizar, automatizar procesos y hasta diseñar prototipos sin necesidad de ser expertos en el lenguaje de bajo nivel. Resulta sencillo el manejo de matrices y vectores, sin necesidad de declarar tipos de datos ni reservar espacio de memoria (en función de qué aplicaciones). MATLAB se encarga de ello de forma automática, mientras que el usuario escribe unas pocas líneas de código. Este proceso libera tiempo para centrarse en lo importante: la lógica y los resultados [33]. Sus principales herramientas son las siguientes (ver figura 14):

- Ventana de comandos: Es el lugar donde se pueden realizar experimentos rápidos. Se escribe una operación, se presiona la tecla “Enter” y arroja el resultado.
- Editor de scripts (archivos .m): En este tipo de archivos es posible escribir programas completos, guardar cálculos, automatizar rutinas o diseñar algoritmos complejos.
- Workspace: Es la mesa de trabajo digital. Aquí, se visualizan todas las variables activas, su tamaño, el tipo de formato y su valor actual. Si no se realiza una limpieza del Workspace los datos permanecen organizados y pueden ser reutilizables.

- Historial de comandos: almacena todos los comandos empleados en la ventana de comandos. Una opción para acceder a ellos es utilizando la flecha superior e inferior del teclado.
- Toolboxes: Son conjuntos de herramientas específicas para áreas de trabajo concretas (procesamiento de señales, visión artificial, análisis estadístico, aprendizaje automático, control de sistemas dinámicos, etc…). Se tratan de extensiones de software y paquetes de hardware que permiten profundizar sobre determinados campos de interés. Estas extensiones se ubican en el panel de "Add – Ons", entre otras opciones.
- Simulink: Es un entorno de simulación gráfica que permite modelar sistemas dinámicos con diagramas de bloques. Por ejemplo, es posible diseñar un sistema de control para posteriormente simularlo y analizar su comportamiento antes de construirlo.
- App Designer: Permite crear interfaces gráficas de usuario (GUI) sin la necesidad de programar desde cero. Es posible hacer aplicaciones propias con botones, controles deslizantes y gráficas.
- Simscape: Es capaz de simular sistemas físicos multidominio (eléctrico, mecánico, hidráulico) usando componentes prediseñados. Entre sus funcionalidades, está el análisis de circuitos neumáticos de forma virtual, o realizar pruebas a motores eléctricos [36].

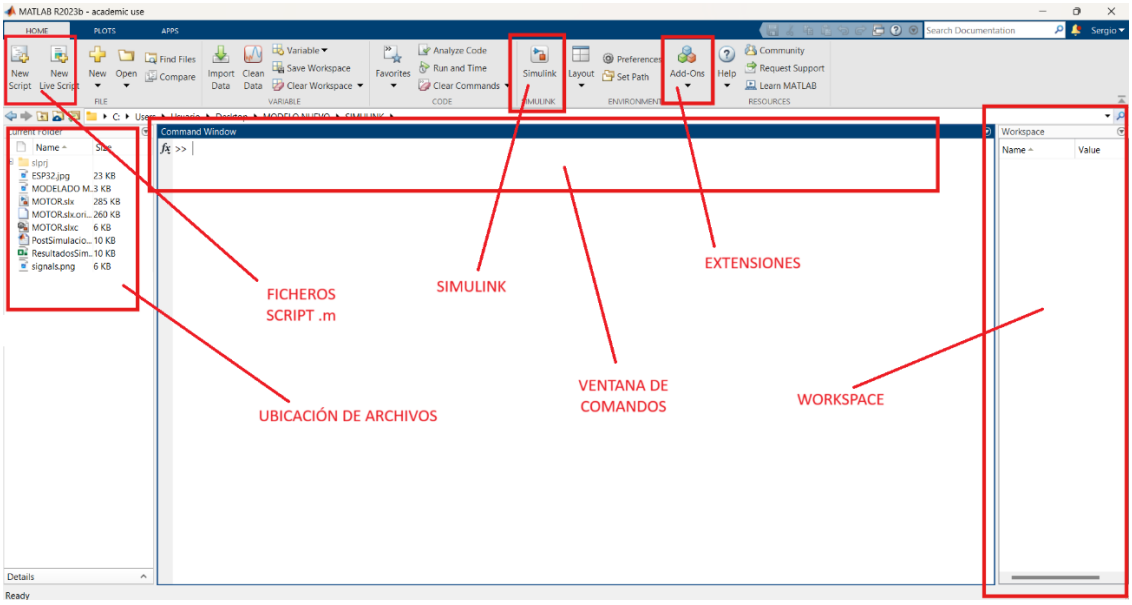


Figura 14: Breve ilustración de la interfaz de MATLAB. Elaboración propia.

En resumen, MATLAB es una herramienta muy útil para la automatización industrial, el diseño de sistemas embebidos y la integración con hardware real. Esto abre un abanico de posibilidades como la comunicación con PLCs, microcontroladores o tarjetas Arduino y Raspberry Pi para pruebas de algoritmos de control en tiempo real (Hardware in the Loop, o "HIL"). Además, ofrece integración con lenguajes como Python, C/C++ o Java, lo que facilita combinar su potencia numérica con otros entornos. También es compatible con sistemas SCADA y bases de datos industriales, de forma que se pueden extraer datos de sensores, analizarlos e incluso generar reportes automatizados de los mismos. Por otro lado, MATLAB también permite aprovechar GPUs y clusters para procesar grandes volúmenes de datos, cosa que es importante en sectores como el industrial, aeronáutico, automotriz o energético.

Por ejemplo, una aplicación muy útil puede ser una empresa automotriz como Renault en Valladolid, que puede utilizar MATLAB para modelar un sistema de frenos ABS, simularlo con Simulink, exportar el código a un microcontrolador y luego probarlo en un banco de pruebas físico. Todo ello sin salir del mismo ecosistema.

2.5.6. SIMULINK

Simulink es un laboratorio virtual donde es posible realizar pruebas y análisis sin miedo a quemar circuitos o dañar equipos costosos. Este programa nació como una extensión de MATLAB, aunque hoy en día tiene su propio espacio dentro de este. Es una plataforma para modelar, simular y analizar sistemas dinámicos usando diagramas de bloques y líneas de código [37].

Simulink combina lo visual con lo matemático. De esta manera, estudiantes, ingenieros y científicos pueden ver cómo interactúan los componentes de un sistema – ya sea eléctrico, mecánico, hidráulico o todos juntos – antes de construirlo. Se apoya en MATLAB, pero la simulación está basada en modelos. Es decir, primero se representa matemáticamente el sistema, posteriormente se interconecta y finalmente se pone a prueba. Las principales herramientas de Simulink son las siguientes:

- Biblioteca de búsqueda: Aquí se encuentran todos los bloques organizados por categorías: fuentes, sumadores, integradores, funciones de transferencia, bloques de control lógico, interfaces gráficas y mucho más.
- Editor de modelos: Es el espacio de trabajo. Allí se construyen los diagramas de bloques arrastrando, conectando y ajustando parámetros.
- Scope: es un osciloscopio virtual. Permite observar señales en tiempo real durante la simulación.
- Solver Configuration: Simulink ofrece solvers numéricos que controlan la precisión y velocidad de la simulación. El usuario decide cómo ajustar la exactitud de cálculo y el tiempo de cómputo.
- Model Explorer y Data Dictionary: Organiza variables, parámetros y configuraciones de simulación. Muy útil para diseño de modelos complejos.
- Simscape: Cuando es necesario modelar comportamientos lógicos y máquinas de estados (como, por ejemplo, sistemas que cambian de modo de operación en función de unas condiciones específicas), Stateflow lo resuelve con diagramas de estados y transiciones [39].
- Simulink Coder: En Simulink no solo se puede simular, sino que también es posible generar código C/C++ automático para implementar algoritmos en hardware real, como microcontroladores, DSP (Digital Signal Processing) o PLC (Programable Logic Controller).
- Simulink Real – Time y Hardware in the Loop (HIL): Permite realizar simulaciones en tiempo real conectadas a hardware físico. Esto es necesario para validar algoritmos de control antes de que entren en producción, por ejemplo.

Figura 15: Imagen extraída de MathWorks. Objetivos de proyectos con Simulink. Extraído de [9].

Simulink es muy utilizado en industrias como la automotriz, aeroespacial, robótica, energía renovable y automatización industrial (ver figura 15). La capacidad de validad diseños previamente a su construcción ahorra tiempo y costes [38].

Un ejemplo puede ser el diseño y validación del sistema de ABS de los vehículos, el control de crucero adaptativo o la gestión de las baterías en los eléctricos. Se comienza modelando todo virtualmente, se ajustan los parámetros necesarios hasta que la respuesta deseada sea la ideal, se simula en condiciones extremas y finalmente se procesa a un Harware in the Loop para visualizar su comportamiento en un entorno físico.

Además, Simulink es compatible con PLCs, microcontroladores y hardware de terceros. Con el paquete "Simulink PLC Coder" es posible generar programas en lenguaje estructurado (IEC 61131 – 3) para controladores industriales.

Otro ejemplo son las energías renovables. Simulink es utilizado para modelar turbinas eólicas o sistemas fotovoltaicos, optimizando el rendimiento y anticipándose así a futuros problemas. Gracias a las integraciones realizadas con MATLAB, es posible analizar datos históricos, optimizar parámetros con algoritmos de aprendizaje automático y retroalimentar el diseño.

2.5.7. JUSTIFICACIÓN DEL USO DE SIMULINK

Simulink es un campo de pruebas digital para ideas complejas. Permite experimentar y cometer errores sin riesgos ni grandes costes. Su enfoque visual lo hace intuitivo, pero también es potente y escalable para resolver desafíos de alto nivel en la industria. Por tanto, es un programa muy versátil. Por este motivo, la actualización para la maqueta de prácticas será compatible con MATLAB, siendo capaz de visualizar los datos en tiempo real y simular el modelo del motor de corriente continua, para posteriormente comparar su comportamiento con el real, y diseñar un lazo de control para ajustar de forma precisa la velocidad deseada por el usuario, rechazando en la medida de lo posible las perturbaciones del sistema [38]. En resumen, Simulink es ideal para la nueva maqueta de prácticas por los siguientes motivos:

- Visualización de todo el sistema como un bloque funcional: Con Simulink es posible crear el modelo del motor de corriente continua (DC) de forma gráfica, mediante bloques para la dinámica del motor (mecánica, eléctrica, fricción, etc.). Además, es intuitivo para el estudiante, ya que se aprecia cómo afectan los parámetros físicos (como la resistencia del bobinado, la ganancia del sistema, el torque) al comportamiento real del motor.

- Realizar la identificación de parámetros de forma integrada: Previamente al diseño del controlador, es necesario conocer la planta (resistencia, inductacia, constante de tiempo, fricción, inercia…). Con Simulink es posible utilizar bloques del conjunto “System Identification” o incluso ejecutar scripts en MATLAB para ajustar los modelos matemáticos a datos reales que se registren con la ESP32.

- Generar código para implementarlo en el ESP32: en el caso de emplear la herramienta “Simulink Coder” junto con el paquete de soporte para la ESP32, es posible generar de manera automática el código en C/C++ del controlador y subirlo directamente al microcontrolador.

- Realizar pruebas Hardware in the Loop (HIL): En proyectos educativos o prototipado, Simulink permite hacer HIL: una parte del sistema se ejecuta en el ordenador (el modelo) y la otra en el hardware real (el motor y el ESP32). Así es

posible verificar cómo se comporta el controlador frente a la planta física, mientras se mantiene ese entorno seguro para realizar pruebas.

2.6. PROTOCOLOS DE COMUNICACIÓN

En electrónica y automatización, rara es la vez en la que no sean necesarios protocolos de comunicación. El microcontrolador necesita intercambiar información con sensores para determinar variables como la tensión, la corriente consumida o la velocidad de un motor. Además, también necesita comunicarse con el ordenador, Simulink y otros dispositivos para enviar datos, recibir órdenes e incluso actualizar determinados parámetros. Ahí es donde entran en juego los protocolos de comunicación: consisten en reglas y métodos que permiten que dos o más dispositivos se entiendan independientemente del fabricante. Son idiomas bien definidos que evitan posibles malentendidos eléctricos y lógicos.

Para la nueva maqueta, no solo es necesario enviar datos al ordenador y recibir comandos desde simulink. También es necesario mostrar por una pequeña pantalla LCD los datos en tiempo real y tomar la información que los sensores de la maqueta extraen. Dos de los protocolos más comunes son UART e I2C.

2.6.1. PROTOCOLO UART

UART es el protocolo Universal Asynchronous Receiver – Transmitter, y consiste en un conjunto de reglas que rigen el intercambio de datos en serie entre dos dispositivos. Es un protocolo muy utilizado debido a la facilidad de implementación, costes y versatilidad. Solamente son necesarios dos hilos para enviar y recibir datos (TX y RX). (Ver figura 16).

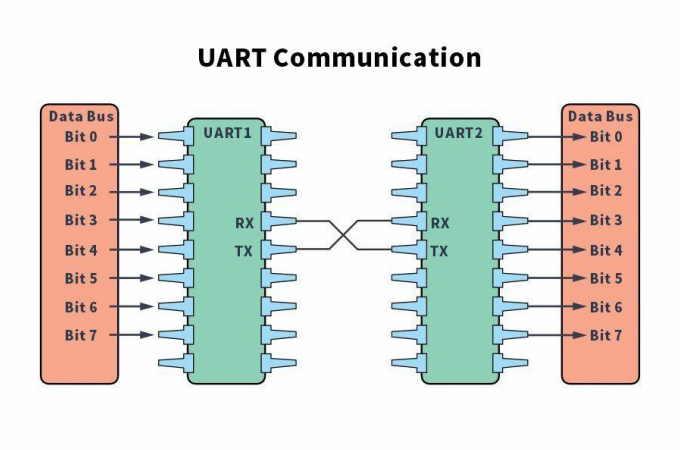


Ilustración 16: Representación de protocolo UART entre dos dispositivos. Extraído de [10].

No son necesarias señales de reloj ni hilos adicionales. Debido a esto, el protocolo funciona de forma asíncrona, lo que indica que cada dispositivo tiene su propio reloj interno para realizar el conteo de tiempo entre bits. La única condición es que la velocidad de transmisión debe ser la misma en ambos dispositivos, lo que comúnmente se denomina como “baud rate”, o “tasa de baudios” [10]. El baudio (Bd) es una unidad que mide la velocidad de transmisión de señales, específicamente el número de símbolos por segundo que se transmiten en un canal de comunicación. Este concepto es importante debido a que la transmisión con la que se trabaja en la maqueta está compuesta por conjuntos de bits representados en caracteres ASCII. Por otro lado, es necesario distinguir entre baudios (Bd) y bits por segundo (bps). Un símbolo (como ASCII) puede representar uno o más bits de información, por lo que la velocidad en baudios no necesariamente coincide con la velocidad en bits por segundo (ver figura 17).

- Baudios: Mide la velocidad de cambio que existe en la señal establecida en el canal de comunicación.
- Bits por segundo: Mide la cantidad de bits que se transmiten por segundo en el canal de comunicación.

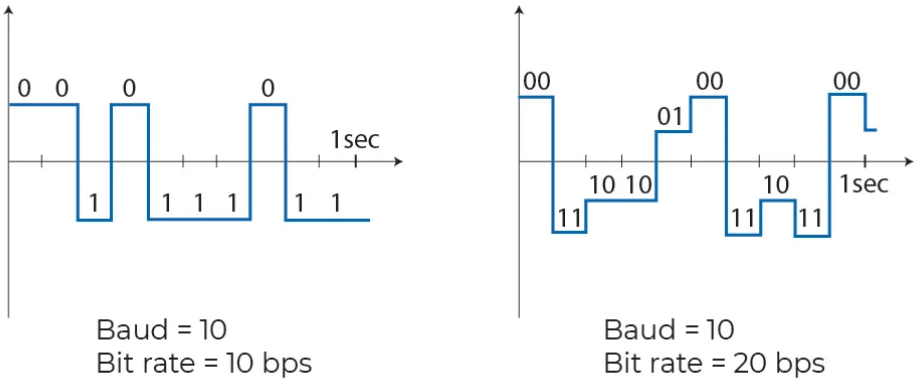


Figura 17: Diferencias entre baudios y bps. Extraído de [11].

UART funciona de la siguiente manera: cada byte (conjunto de 8 bits) se envía bit a bit, comenzando con un bit que indica el inicio de la transmisión de información (bit de inicio). Posteriormente, se transmiten de forma serial los 8 bits de información (o más, en función de la configuración establecida). Después de enviar los bits correspondientes a los datos, opcionalmente se introducen bits de paridad (par/impar) cuyo propósito se basa en la detección de errores en el conjunto de bits enviado. Finalmente, se insertan uno

o dos bits de parada para indicar que la transmisión ha finalizado. Todo ello forma parte de un conjunto que se denomina “trama” (ver figura 18).

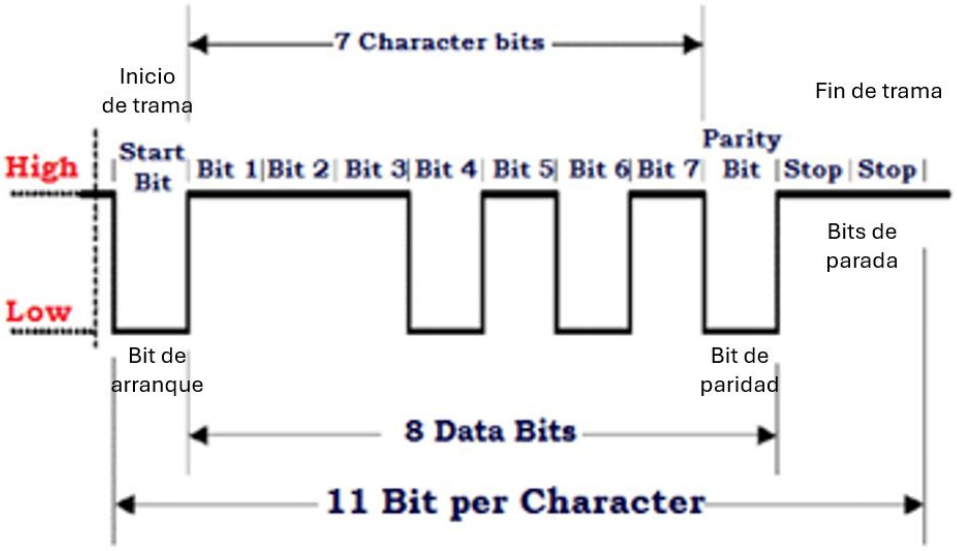


Figura 18: Ejemplo de trama protocolo UART. Imagen editada y extraída de [12].

En la práctica, los datos recogidos por el codificador incremental del motor DC en la maqueta, los valores de tensión y corriente son enviados al ordenador a través de este protocolo. A su vez, el microcontrolador recibe tramas del ordenador, por lo que es necesario distinguir mediante una cabecera a quién corresponde cada mensaje. Esto se verá más adelante en el trabajo, siendo posible gracias a los caracteres ASCII. La velocidad de baudios empleada es de 115200 Bd, aunque también existen 4800, 9600, 19200, 57600, y así hasta 2 millones de baudios (capacidad máxima del Arduino IDE).

Tal y como se observa en la figura 18, el valor de comunicación cuando no se está transmitiendo es de nivel alto. De esta manera, es posible reconocer entre una comunicación en reposo y una comunicación defectuosa (por ejemplo, si los hilos se desconectaran por cualquier razón aparente, no sería posible distinguir el estado de reposo o avería en la conexión si se mantuviera a nivel bajo). Para iniciar la comunicación, basta con llevar a nivel bajo el estado de la comunicación con 1 bit. Inmediatamente después, se introducen los bits de datos y se finaliza con el bit de parada. En algunos casos, suele configurarse con un segundo bit de parada para permitir que el receptor se prepare para la próxima trama.

Por ejemplo, en la nueva maqueta, las tramas enviadas y recibidas llevan una cabecera correspondiente al dispositivo que las realiza. De esta manera, es posible

distinguir qué datos son importantes para el microcontrolador y cuáles para el ordenador. Uno de los dispositivos envía el carácter ASCII “A” mayúscula, correspondiente al número 65 en decimal. Por ello, en bits se traduce a 1000001. A este conjunto de datos se le añade el bit de inicio de trama y el o los bits de parada. Si fuera necesario implementar el bit de paridad, se haría de la siguiente manera:

- En la paridad par, si el número total de “1” en los datos es par, entonces el bit de paridad es “0”. En caso contrario, es “1”, para ajustar siempre el contenido a una cantidad par.
- De la misma manera, si se elige la paridad impar, el bit de paridad se ajustará de manera que el conjunto de datos sea siempre impar.

El procotolo UART tiene muchas aplicaciones, especialmente en el sector industrial. No solamente se limita al uso de dos hilos físicos. También es posible utilizar esta estructura en transmisión inalámbrica, por ejemplo, el Bluetooth. Es un método de comunicación bastante fiable y con la posibilidad de implementación a largo alcance, donde otros protocolos no son capaces de desempeñar una comunicación, como I2C o SPI.

2.6.2. PROTOCOLO I2C

El término I2C cuyas siglas representan “Inter – Integrated Circuit”, o “Circuito inter – integrado”. Este protocolo comenzó en los años 80 de la mano de la empresa Philips (actualmente NXP) y se ha convertido en uno de los principales estándares para conectar varios periféricos de forma ordenada y únicamente con dos hilos [13]. Con I2C, es posible instalar en la misma línea un sensor de corriente, otro de velocidad y una pantalla LCD, por ejemplo, para mostrar los datos del motor, e incluso una memoria EEPROM donde almacenar todos los parámetros. Se trata de una configuración Maestro – Esclavo donde, el maestro es el microcontrolador y los esclavos son los periféricos que se necesitan controlar. Cada esclavo posee una dirección única y la comunicación se realiza de la siguiente manera:

- SDA: Serial Data Line, o Línea Serial de Datos, donde circulan todos los bits de datos a transmitir.
- SCL: Serial Clock Line, o Linea Serial de Reloj, donde se introduce una señal de reloj por parte del microcontrolador para sincronizar la comunicación.

Se trata, por tanto, de un método síncrono, donde el microcontrolador (maestro) marca el "ritmo" de la comunicación sobre los periféricos (esclavos). I2C es capaz de controlar varios dispositivos de forma simultánea, con la desventaja de emplear cables cuyas longitudes no superen más de un metro, debido a la introducción de ruidos, aumento de la capacitancia, flancos de subida y bajada lentos y errores en la lectura. Por otro lado, se recomienda utilizar un par de cables trenzado para reducir lo máximo posible este tipo de inconvenientes. Además, desde el microcontrolador las líneas de SDA y SCL deben tener resistencias de Pull – Up para que la línea permanezca a nivel alto cuando no se están transmitiendo datos (por el mismo motivo que en el protocolo UART) [13].

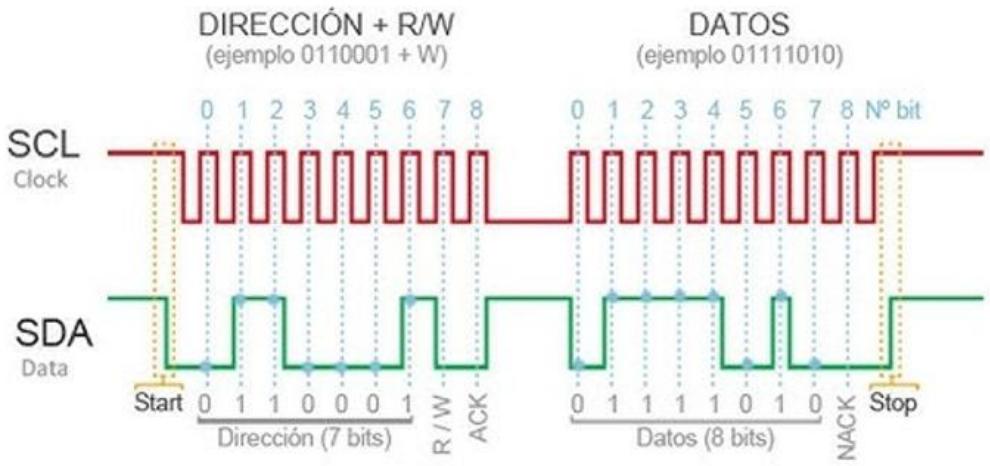


Figura 19: Ilustración de comunicación empleando el protocolo I2C. Extraído de [13].

Como se puede observar en la figura 19, la línea permanece a nivel alto cuando no se realiza ninguna transmisión. En el momento en el que se desea transmitir información, la línea de datos SDA envía un bit de inicio, a partir del cual la señal de reloj SCL comienza a marcar el tiempo de transmisión que se debe seguir y los datos viajan de forma serial. Cada bit se envía cuando SCL está en nivel bajo, mientras que cada bit se lee cuando SCL pasa a nivel alto. De esta manera, el microcontrolador y los periféricos son capaces de distinguir la cantidad de "1" y "0" que existen en la línea de datos. Cada esclavo tiene su propia dirección, compuesta por 7 o 10 bits (lo más común es la primera opción). El maestro inicia y controla la conversación, que se divide en tres fases:

- El maestro comienza la transmisión con un bit de inicio en SDA (nivel bajo) mientras SCL está a nivel alto. En ese momento, todos los dispositivos

permanecen a la escucha. Solo el maestro puede utilizar el bus de datos mientras está ocupado.

- El maestro envía una cadena de 7 bits con la dirección del esclavo al que desea transmitir la información. Posteriormente, manda un bit correspondiente a lectura ("1", el esclavo envía datos al maestro) o escritura ("0", el maestro envía datos al esclavo). El dispositivo cuya dirección coincida con la enviada por el maestro, es la que actúa sobre la línea.

- Envío o recepción de datos (8 bits por bloque): los datos se transmiten con el orden del bit más significativo primero. Cada byte enviado debe ir seguido de un bit con acuse de recibo (ACK, donde el receptor entendió el mensaje y puede continuar, o NACK, donde se interpreta que algo falló y la transmisión debe finalizar).

- Finalmente, se establece la condición de parada, en la que el maestro establece la línea de datos SDA a nivel alto mientras SCL también se encuentra a nivel alto. Todos los dispositivos interpretan que la transmisión ha finalizado.

Por ejemplo, se procede a la lectura de un sensor de corriente INA219 (muy utilizado para medir corriente en motores de corriente continua) a partir de un microcontrolador ESP32. El procedimiento sería el siguiente:

- Condiciones de inicio: El microcontrolador establece SDA a nivel bajo mientras SCL está a nivel alto.
- El maestro envía: Dirección + R/W (Ej: 1000000 + bit de escritura "0"). La dirección corresponde al sensor de corrriente.
- Reconocimiento: Cada esclavo compara dicha dirección con la suya propia y responden con ACK o NACK.
- Esclavo responde: El sensor de corriente reconoce la dirección, envía un ACK y establece SDA a nivel bajo.
- El ESP32 y el sensor intercambian envíos de bytes según la dirección y el bit de escritura. Por cada byte enviado, el receptor debe enviar un ACK.
- El maestro finaliza la comunicación estableciendo SDA a nivel alto mientras SCL también lo está.

Todo este proceso se encuentra representado en la figura 19. A continuación, se introducirá un breve análisis de la maqueta empleada durante los últimos años como sistema educativo de prácticas, prestando especial atención a los componentes de hardware y software empleado para su programación como el interfaz de desarrollo que el estudiante debe manejar.

CAPÍTULO 3:

MAQUETA PREVIA – DISEÑO Y SELECCIÓN DE COMPONENTES

3. MAQUETA PREVIA – DISEÑO Y SELECCIÓN DE COMPONENTES

La idea detrás del trabajo previo al actual, parte de un objetivo: aprender automática aplicando no solo los conocimientos teóricos, sino mediante la práctica. Muchos alumnos cursan la asignatura "Fundamentos de Automática" sin haber visto nunca un sistema real en acción. Este proyecto busca cambiar eso.

Se trata de un sistema educativo integral (desarrollado desde cero) que permite a los estudiantes interactuar con una planta real (denominada Motorpiic) a través de una interfaz gráfica (Autopiic). El objetivo de la maqueta es combinar tanto los conceptos teóricos como los prácticos.

3.1. DESCRIPCIÓN DE LA MAQUETA PREEXISTENTE

Diseño y construcción del equipo de prácticas: Motorpiic

Motorpiic es el núcleo del sistema. Una planta de control sencilla pero lo suficientemente desarrollada como para cubrir prácticas reales en un laboratorio. Está compuesta por:

- Un motor de corriente continua con reductora, seleccionado por su sencillez a la hora de ser modelado y predecible en la práctica.
- Una fuente de alimentación de 12V.
- Un codificador incremental para medir la posición y velocidad con precisión.
- Un potenciómetro para ajustar el valor de tensión deseado y un sensor de corriente que permite la retroalimentación de estado en tiempo real.
- El microcontrolador dsPIC30F4012, que actúa como cerebro del sistema, ejecutando los algoritmos de control.
- Un conector para las comunicaciones con el ordenador.
- Un conversor de puerto serie USB.
- La etapa de potencia necesaria para el control del motor.

Todo ello se ensambla en una maqueta compacta y modular, cuyo diseño está pensado para soportar el uso continuo de los estudiantes, además de ser fácilmente replicable.

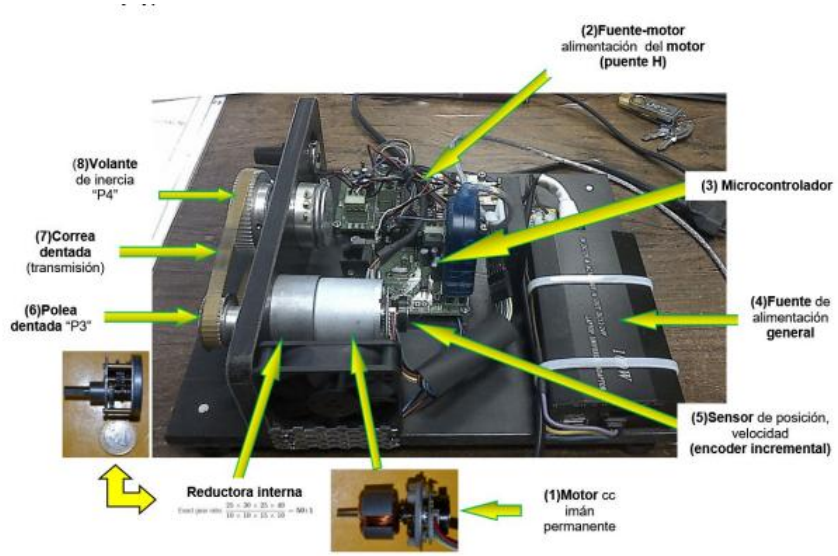


Figura 20: Vista general de la maqueta "Motorpiic". Extraído de documentación anexada.

A continuación, se describirán de forma resumida todos los componentes que forman la maqueta (ver figura 20):

- Etapa de control: Alberga el microcontrolador dsPIC30F4012, la electrónica correspondiente al acondicionamiento de señales, conversores A/D y el sistema de comunicaciones.
- Etapa de potencia: contiene el puente H con los MOSFETs, el driver de control, la medición de corriente y regulación de tensión.

Tabla 2: Elementos de la maqueta desarrollada hasta ahora.

Nº	Componente	Función principal
1	Motor DC con reductora	Es el actuador principal.
2	Codificador incremental	Mide de forma precisa la posición/velocidad.
3	Potenciómetro analógico	Entrada analógica de consigna.

4	Sensor de coriente	Mide el consumo eléctrico del motor (en amperios [A]).
5	Volante de inercia	Simula una carga mecánica al motor.
6	Fuente alimentación	Suministra 12V al motor 5V a la electrónica de control.
7	Conector UART con conversor USB	Comunicación con el ordenador y el microcontrolador.
8	Volante de inercia	Se utiliza para aplicar un par externo al motor sin dañarlo.

3.2 MOTOR DE CORRIENTE CONTINUA

Se trata de un motor DC de 12V y 60W de la marca Pololu. Ha sido seleccionado debido a su bajo coste, compacto y lo suficientemente robusto para prácticas repetitivas. A continuación, se presentan las características técnicas más relevantes:

Tabla 3: Características del motor de corriente continua.

Característica	Valor
Tensión nominal	12 V
Velocidad en vacío (sin carga)	10000 RPM
Corriente nominal	0.3A

Corriente de arranque o bloqueo (stall)	Hasta 5A
Par de bloqueo	0.06 Nm
Inductancia del inducido	0.61mH
Momento de inercia	3.6·10-6 kg·m2

El diseño tiene en cuenta las pérdidas internas de estos motores (de hasta un 50%), así como su comportamiento parabólico en la curva de potencia/par/velocidad.

3.3 SOFTWARE DE MOTORPIIC

El microcontrolador dsPIC30F4012 ha sido programado utilizando el entorno MPLAB IDE, el cual incluye un editor de texto, un programa ensamblador denominado MPASM, un simulador y un organizador de proyectos. Debido a la complejidad que puede existir al programar en lenguaje ensamblador, este microcontrolador ha sido programado en C, y traducido posteriormente a lenguaje máquina por el compilador. Para trasladar el código al dsPIC se ha trabajado con un programador ICD2.

La interfaz visual de la maqueta está diseñada en el programa Visual Basic .NET, donde el usuario puede navegar a través de ella haciendo uso de los botones dispuestos en el lateral izquierdo de la pantalla (ver figura 21).

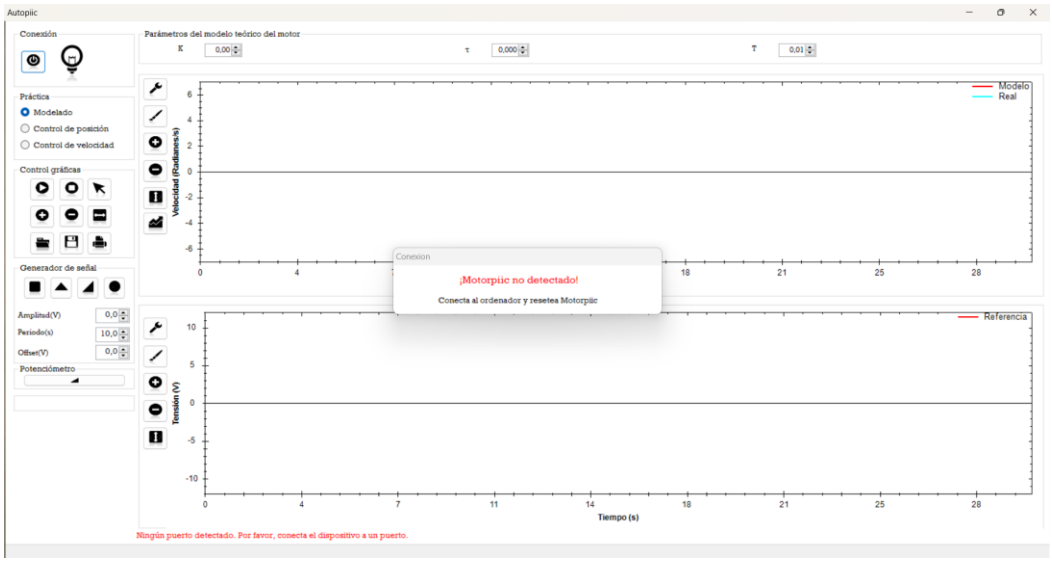


Figura 21: Interfaz del software AUTOPIIC. Elaboración propia.

Las herramientas que dispone la interfaz se pueden resumir en tres grupos:

- Configuración de las gráficas: en esta selección de botones, permite realizar un aumento o disminución del tamaño de las gráficas, el color de representación, el número de puntos representados (periodo de muestreo), desplazarse manualmente por ella, realizar una pausa/reanudar la simulación.

- Configuración de las señales de entrada: en ella, es posible elegir el tipo de entrada que se quiere emplear en la simulación, modificando parámetros como la amplitud y periodo de la señal. Además, es posible realizar un control de la referencia de entrada a partir del potenciómetro incorporado en la maqueta.

- Tipo de control: a la hora de realizar las prácticas, es posible elegir entre tres tipos de control. El primero es de identificación. En este modo, la señal de entrada es la tensión aplicada al motor, cuya respuesta modelo viene determinada por los parámetros de ganancia (K) y tau (τ). En el siguiente modo, se realiza un lazo de control a la posición del motor, y en el tercer modo, a la velocidad. En el panel superior es posible modificar el valor del PID configurado para este control.

3.4 LIMITACIONES DE LA MAQUETA ANTERIOR

Hasta ahora, la maqueta desarrollada se basaba en el microcontrolador PIC, junto con un entorno propietario denominado AUTOPIIC. Si bien desempeñaba las funciones básicas de control e identificación del motor, presentaba diversas limitaciones que motivaron a su rediseño:

- Potencia de procesamiento y memoria: El microcontrolador PIC disponía de recursos computacionales reducidos en comparación con plataformas más actuales. La frecuencia de muestreo y la capacidad para ejecutar algoritmos de control en tiempo real estaban limitadas. Por otro lado, la gestión de comunicaciones simultáneas y cálculos de control PID complejos resultaban poco eficientes.

- Entorno de desarrollo propietario: Aunque el sistema dependía del entorno AUTOPIIC y este estaba correctamente implementado, no era compatible con MATLAB/Simulink, lo que dificultaba la tarea a los estudiantes para trasladar su progreso en las prácticas hacia MATLAB, influyendo así en las prácticas docentes. Por otro lado, la necesidad de aprender un entorno específico implicaba una curva de aprendizaje adicional al estudiante.

- Escasa conectividad: No disponía de conectividaad Bluetooth ni WiFi, además de necesitar configuraciones y cableado específicos para poder realizar una transmisión al ordenador (RS-232, placa adicional para conversión UART a USB).

CAPÍTULO 4:

NUEVA MAQUETA – ANÁLISIS, DISEÑO Y DESARROLLO

4. NUEVA MAQUETA – ANÁLISIS, DISEÑO Y DESARROLLO

Una vez realizado el estudio de la maqueta nueva, se procede a diseñar el nuevo sistema para prácticas. Este sistema debe integrar como mínimo las mismas funciones que su versión antigua, además de ser compatible con el entorno de MATLAB / Simulink a través de los protocolos de comunicación correspondientes. Se tienen, por tanto, los siguientes requisitos mínimos:

- Debe poder realizar las funciones necesarias para la identificación del motor localizado en la maqueta de prácticas.
- Capacidad de implementar un lazo de control PID para monitorizar al menos una de las variables principales del motor (Posición – Velocidad). En este trabajo se estudiará específicamente la variable de velocidad.
- Introducir entradas al sistema por parte del usuario para analizar la respuesta real y modelada del motor (tipo escalón, rampa, constante, senoidal, etc…). Dichas entradas en lazo abierto corresponderán al nivel de tensión aplicado al motor, mientras que, en lazo abierto, serán el valor de consigna para la velocidad deseada (en revoluciones por minuto).
- Diseñar una interfaz gráfica (HMI) para facilitar al usuario el control sobre la maqueta para prácticas.
- Tiempo de respuesta adecuado en la comunicación serial para trabajar en tiempo real, y la obtención de parámetros como la tensión aplicada (en voltios), la corriente consumida por el motor (en miliamperios) y la velocidad calculada a partir del codificador incremental (en revoluciones por minuto).
- Compatibilidad con MATLAB y Simulink para realizar el estudio y visualización de las variables de la maqueta.

Todo ello se llevará a cabo empleando en la medida de lo posible las piezas de las maquetas ya existentes.

4.1. SELECCIÓN DE COMPONENTES - HARDWARE DE LA MAQUETA

Para el nuevo diseño, se reutilizará la fuente de alimentación de 100W, que proporcionan una tensión de salida continua entre 12 y 24 voltios, y la base de madera donde se sitúan la electrónica, la fuente y el motor.

4.1.1. ACTUALIZACIÓN DEL MICROCONTROLADOR

El microcontrolador utilizado en las maquetas hasta ahora ha sido el dsPIC30F4012 (ver figura 22), debido a sus capacidades de control en aplicaciones industriales y robóticas: PWM de alta resolución, comunicación CAN para buses robustos, convertidor analógico-digital (ADC) rápido, etc. Además, integra capacidades DSP (Data Signal Processing), que lo hacen ideal para aplicaciones como control de motores, fuentes conmutadas, procesamiento de señales y demás. La velocidad del microcontrolador se mide en MIPS (Millions of Instructions Per Second).

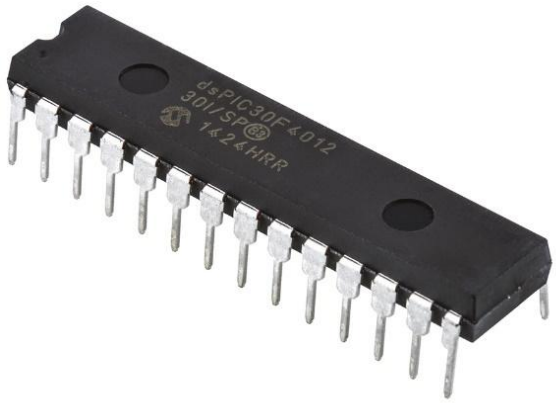


Figura 22: Microcontrolador dsPIC30F4012. Extraído de hoja de datos anexada.

FIGURE 1-1: dsPIC30F6011A/6012A BLOCK DIAGRAM

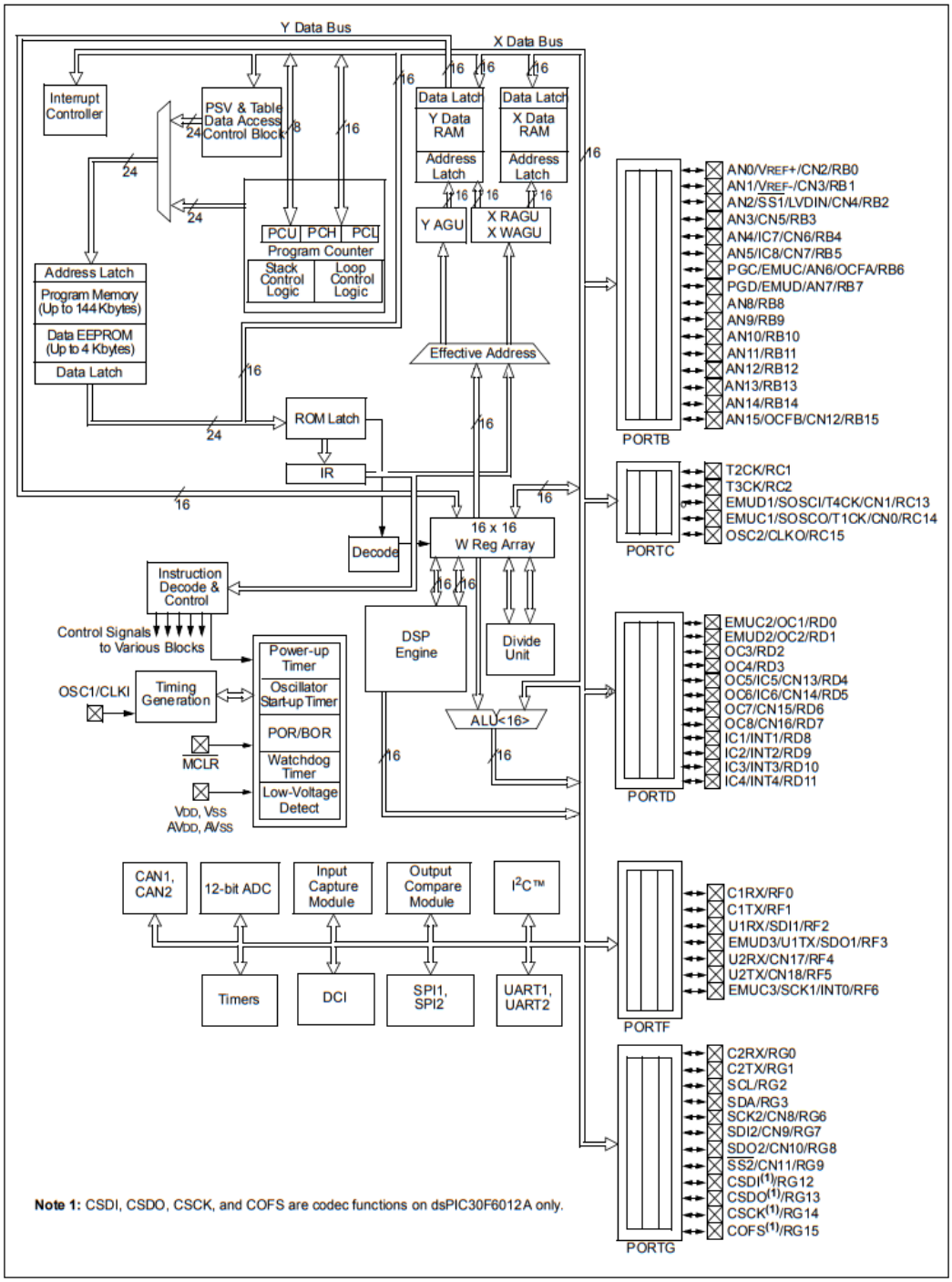


Figura 23: Arquitectura del dsPIC30F6011A. Extraído de hoja de datos anexada.

- Posee una arquitectura de 16 bits con juego de instrucciones DSP (capacidad de hasta 30MIPS). (Ver figura 23).
- Memoria Flash de 48KB.
- Memoria RAM de 2KB.

- Tensión de operación de 2.5 a 5.5 V.
- 9 canales de convertidores analógico/digital de 10 bits de resolución (hasta 500000 muestras por segundo). No posee convertidores digitales/analógicos (ver figura 24).
- Conectividad UART, SPI, I2C y CAN.
- Velocidad de reloj de hasta 30 MHz (en la maqueta se añade una extensión a un reloj de cuarzo externo para aumentar la tasa de velocidad).
- Es especialmente útil para control de motores, análisis de señales en tiempo real y su consumo es relativamente bajo.
- Necesita herramientas de Microchip para su programación (MPLAB, XC16).

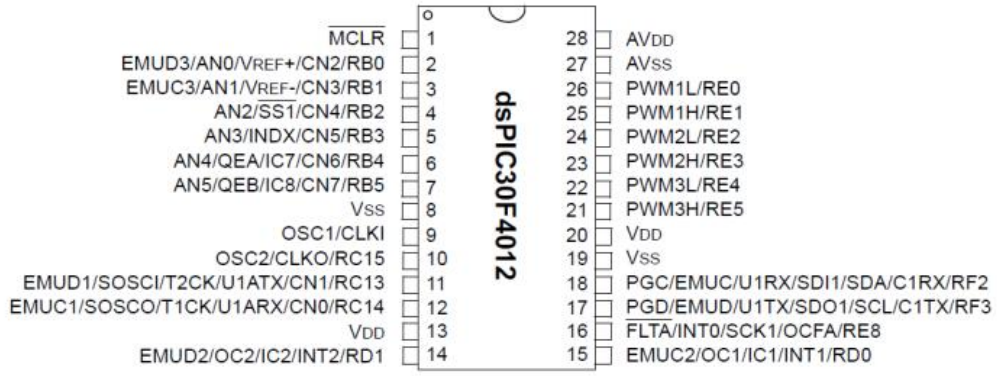


Figura 24: Disposición de pines dsPIC. Extraído de hoja de datos.

Este microcontrolador está especializado en el control de motores DC y PID en tiempo real (posee un PWM avanzado con 3 pares complementarios, deadtime y sincronización), realiza lecturas ADC rápidas y utiliza el conjunto de instrucciones DSP que lleva integrado. Se propone entonces la implementación de un nuevo microcontrolador, con el objetivo de aumentar las capacidades que la maqueta ofrece y establecer una compatibilidad con MATLAB. Se trata del microcontrolador ESP32 Devkit V1.

4.1.2. MÓDULO ESP32-WROOM-32

El ESP32-WROOM-32 es un módulo compacto (18x25.5x3.1 mm) que integra el chip ESP32-D0WDQ6 junto con una antena y memoria flash. Certificado para Wi-Fi, Bluetooth y diseñado para tareas desde sensores hasta procesamiento de audio (entre otras) [22]. Sus características técnicas son las siguientes:

- Chip: dual-core Xtensa LX6 (80-240MHz), 520kiB SRAM, además de 448KiB ROM y 16KiB de SRAM en RTC (Real Time Clock).
- Conectividad inalámbrica: Wi-Fi 802.11 b/g/n (hasta 150Mbps), Bluetooth v4.2 BR/EDR y BLE (Bluetooth Low Energy).
- Periféricos: convertidores AD/DA, comunicaciones SPI, I^2C, UART, I^2S, SDIO, CAN, interrupciones (timer), sensores táctiles y de efecto Hall.
- Alimentación: 2.7 – 3.6 V: consumo típico de 80mA funcionamiento, menos de 5 – 10 uA en modo reposo (bajo consumo).
- Flash: 4MB QSPI integrada. Más de 100 000 ciclos de lectura/escritura de programas, 20 años de retención de datos.
- Interfaz física: 38 pines, excluyendo 6 de ellos, utilizados directamente por la memoria flash.

Gracias a su coprocesador de ultra bajo consumo (ULP), puede permanecer en alerta con menos de 5 uA de corriente, despertando ante cambios en sensores o temporizadores. Incluye aceleradores hardware (AES, SHA, RSA, ECC) y soporte para un arranque seguro, cifrado de flash y OTP de hasta 1024 bits [20].

Para este proyecto, se empleará el microcontrolador ESP32-DevkitV1, el cual integra el módulo ESP32-WROOM-32 (ver figura 25).

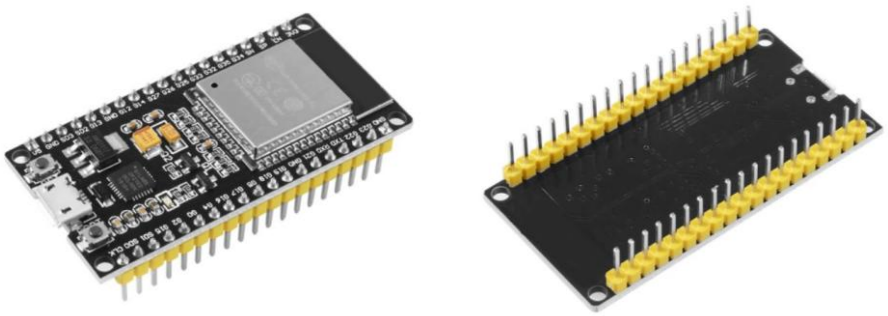


Figura 25: Microcontrolador ESP32 DevKit V1 de 38 pines. Extraído de [14].

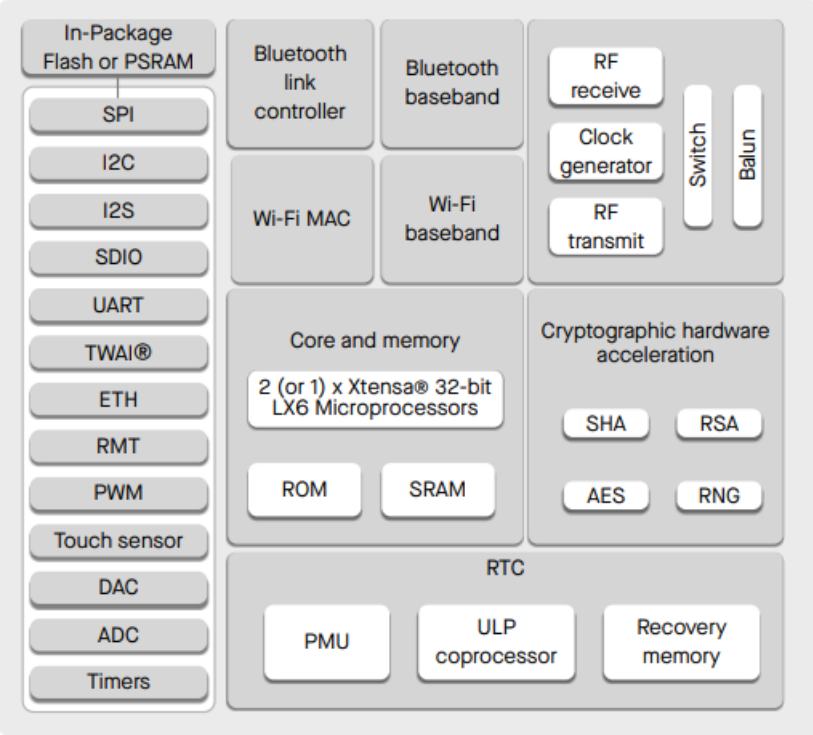


Figura 26: Diagrama funcional de la ESP32 DevKit V1. Extraído de [14].

Este nuevo dispositivo se basa en el ESP32 – WROOM – 32 y posee hasta 38 pines de salida, de los cuales 6 de ellos son utilizados directamente por la memoria SPI Flash [24]. Sus características son las siguientes (ver figura 26):

- Posee un procesador de doble núcleo (que operan hasta 160 – 240 MHz) basado en el Tensilica LX6 de 32 bits.
- Un coprocesador de bajo consumo (ULP o Ultra Low Power) para tareas de baja exigencia energética.
- Memoria RAM de 520 KB (incluyendo la RAM interna y la caché).
- Memoria Flash interna del módulo ESP32 - WROOM – 32 de hasta 4 MB.
- Conectividad WiFi integrada 802.11 b/g/n.
- Conectividad Bluetooth v4.2 / BLE (Bluetooth Low Energy).
- Aceleración por Hardware para criptogradía (AES – Advanced Encryption Standard - , SHA – Secure Hash Algorithm).
- Sensores internos (de Efecto Hall y sensor de temperatura).
- Un chip integrado conversor de UART a USB y viceversa.
- Convertidor Digital Analógico (ADC) con una resolución de 12 bits y hasta 18 canales.
- Dos convertidores Analógico Digital (DAC) integrados.

- Conectividad UART, SPI, I2C, I2S, CAN (con adaptador) y Ethernet (con PHY).

Este microcontrolador es popularmente conocido debido a sus amplias capacidades para múltiples proyectos, además de la facilidad de desarrollo que ofrece, pudiendo ser programado mediante Frameworks sencillos como Arduino IDE, ESP-IDF, MicroPython o PlatformIO [21].

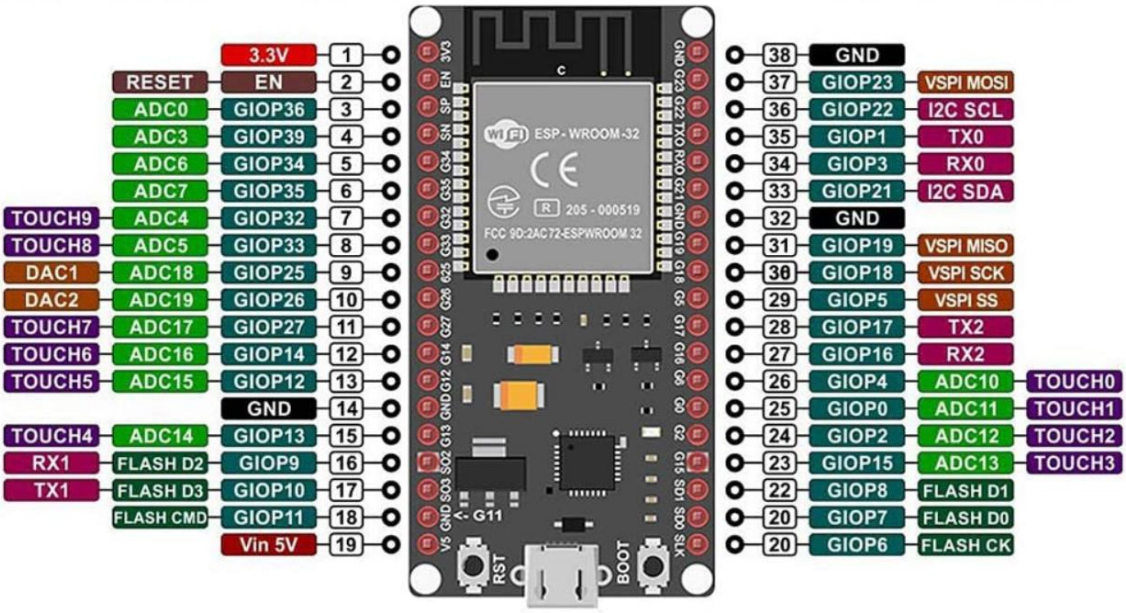


Figura 27: Pinout del ESP32 DevKit V1. Extraído de [14].

La configuración de sus pines es la siguiente (ver figura 22):

- 6 pines empleados por la memoria flash SPI
- 4 pines sólo funcionan como entradas (carecen de resistencia interna pull-up y pull-down).
- 10 pines son capacitivos (sensor efecto Hall).
- 18 pines funcionan como canales del convertidor analógico/digital (ADC).
- 2 pines funcionan como convertidor digital/analógico (DAC).
- 16 pines pueden generar señal PWM (aquellos capaces de actuar como salida).
- 2 pines para la comunicación I2C, 2 para la SPI y 6 pines (3 pares) para comunicación UART.

A continuación, se resume la comparación de ambos microcontroladores para determinar la sustitución de este:

Tabla 4: Comparativa de la actualización del microcontrolador en la maquetra de prácticas.

Parámetros	dsPIC30F4012	ESP32 DevKit v1 (38 pines)
Arquitectura	16-bits DSP (Digital Signal Controller)	32-bits doble núcleo (Xtensa LX6)
Velocidad de CPU	Hasta 30 MIPS (~30 MHz)	Hasta 160–240 MHz (doble núcleo)
Memoria RAM	2 KB	~520 KB
Memoria Flash	48 KB	~4 MB (incluye memoria para programas y datos)
Voltaje de Operación	2.5–5.5 V	3.3 V
Entradas ADC	10-bits, 9 canales, hasta 500 ksps	12-bits, ~18 canales (no tan lineales para control crítico)
Control de Motor	PWM avanzado para control de motor, periféricos DSP integrados	PWM general, menos específico para motores, pero configurable
Conectividad	UART, SPI, I2C, CAN (no WiFi/Bluetooth)	WiFi, Bluetooth, UART, SPI, I2C
Consumo de Energía	Moderado, pensado para entornos industriales	Moderado-alto, pero con opciones de bajo consumo para IoT
Robustez en Aplicaciones de Control	Alta para control de motor en tiempo real	Media (puede requerir programación cuidadosa para cumplir requisitos de tiempo real)
Facilidad de Desarrollo	Necesita MPLAB X e IDE de Microchip (más técnico)	Compatible con Arduino, ESP-IDF, MicroPython (más comunidad, soporte y documentación)
Ideal para	Aplicaciones críticas de control de motor e industriales	Aplicaciones de control menos críticas, donde la conectividad e interfaces gráficas son clave
Coste y Disponibilidad	Moderado, menos utilizado en proyectos actuales	Bajo coste y gran disponibilidad en el mercado actual

Otros Atributos	Especializado en control de motor, DSP integrado	Ideal para IoT, control básico de motores, dashboards, comunicación remota

4.1.3. ACTUALIZACIÓN DEL PUENTE H EN LA MAQUETA

Para el correcto funcionamiento y control del motor, las maquetas hasta ahora han empleado un modelo de puente H “HIP4081” (ver figura 28). Se trata de un controlador MOSFET de 4 canales utilizado para construir estos puentes de alta potencia.

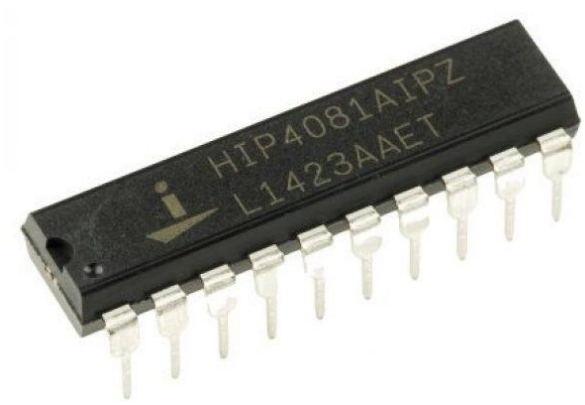


Figura 28: Puente H HIP4081. Extraído de la hoja de datos anexada.

En las maquetas previas, su diseño emplea 4 transistores de tipo MOSFET para formar un puente completo junto con el controlador HIP4081. El motor de corriente continua es alimentado a través de la fuente conmutada de 12V, cuya velocidad y sentido de giro lo determinan la lógica del controlador sobre los 4 transistores. Además, incorpora una configuración de pines para la medición de corriente en el motor mediante un amplificador operacional. La sugerencia de diseño del fabricante es la representada a continuación (ver figura 29):

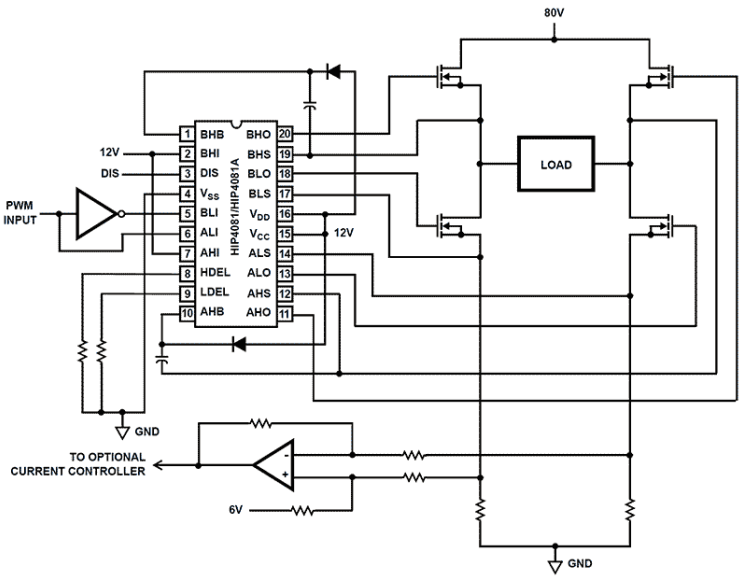


Figura 29: Diseño de control para velocidad y sentido de giro con medición de corriente para HIP4081, empleado en la maqueta. Extraído de la hoja de datos anexada.

El nuevo diseño planteado para el control del motor con el ESP32 parte de los mismos principios que la maqueta previa, mediante el uso de un puente H. En esta versión, se ha optado por el controlador L298N. Se basa en un doble puente capaz de realizar el control sobre dos motores de forma independiente, ofreciendo como mínimo las mismas prestaciones que el controlador HIP4081, con pines específicos para medir la corriente en ambos motores (ver figura 30).

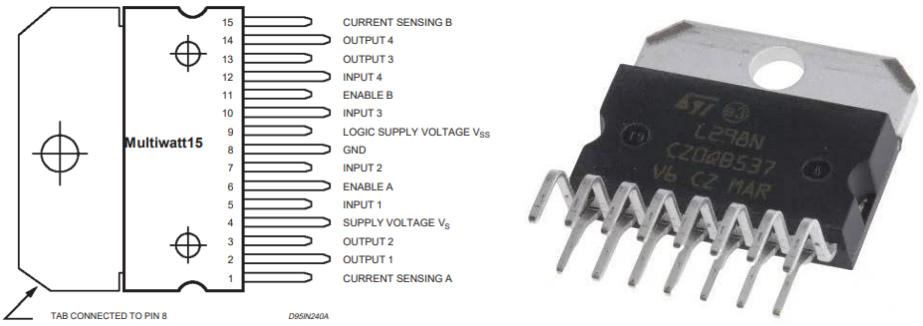


Figura 30: Controlador L298N ZIP-15. Extraído de la hoja de datos anexada.

Se trata de un circuito integrado compuesto por 15 pines para controlar motores de corriente continua y otros tipos de cargas inductivas. Es muy conocido en electrónica de potencia para proyectos de robótica y automatización. Suele fabricarse en un formato de módulo para ser utilizado directamente por el usuario (ver figura 31). Su diseño es simple y cómodo de integrar en placas electrónicas. Las características principales son las siguientes:

- Es un controlador de puente H dual integrado (posee dos canales para el control independiente de hasta dos motores).
- Soporta una tensión máxima de alimentación de 46V (necesita 5V para la tensión de lógica).
- Puede proporcionar una corriente de salida de hasta 2A por canal (con disipador de calor) y superar corrientes de pico de 3A por breves periodos de tiempo.
- Su lógica de control es compatible con TTL (Tiempo de vida) 5V.
- Posee 4 pines de entrada para el control de los 2 puentes completos (IN1, IN2, IN3, IN4).
- Dos salidas para cada motor (OUT1-2 para el motor A, OUT3-4 para el motor B).
- Su diseño requiere diodos de protección para las cargas inductivas.
- El consumo de lógica es alrededor de 30~60 mA.
- Destina dos pines específicos para medir la corriente que atraviesa la carga.
- Se vende como módulo completo para control de motores DC (el módulo inhabilita la función de medir corriente en ambos motores).

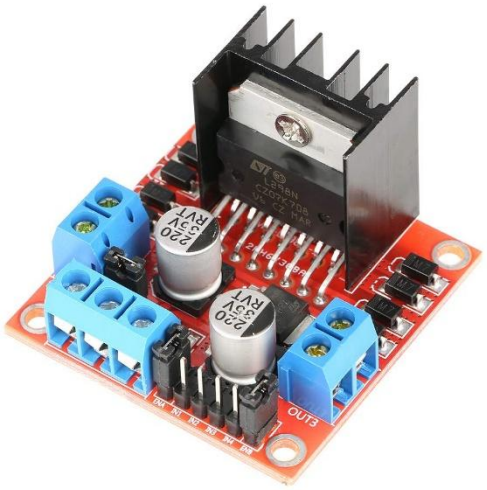


Figura 31: L298N en formato módulo. Extraído de la hoja de datos anexada.

En resumen, el L298N garantiza que pueda alimentar y controlar motores de corriente continua de pequeña a mediana potencia (que son los que se encuentran en las maquetas de prácticas). Además, dispone de entradas compatible con la lógica TTL, la cual facilita la conexión directa con el microcontrolador ESP32, sin la necesidad de adaptar los niveles de tensión (ver figura 32). Al contar con dos puentes H internos, es

posible controlar tanto la velocidad como el sentido de giro de uno o dos motores, incluso el de un paso a paso empleando un único dispositivo.

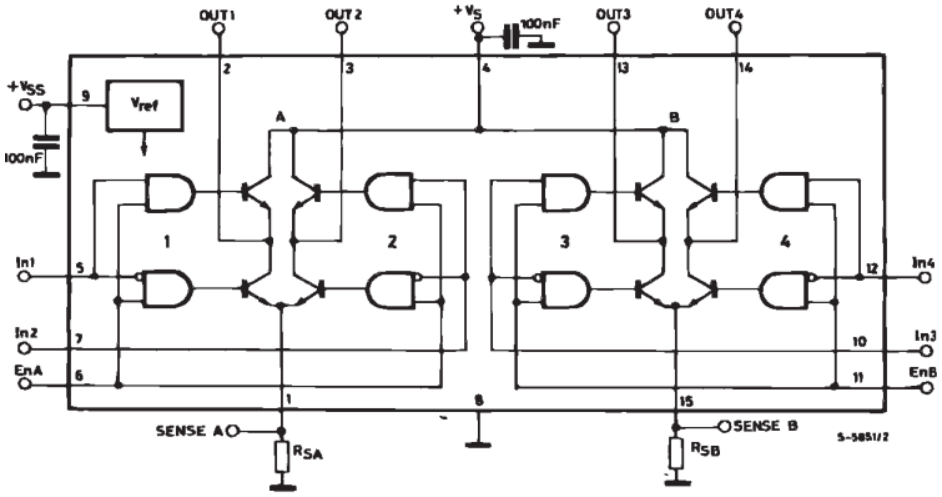


Figura 32: Diagrama de bloques del L298N para el control de dos motores (A y B). Extraído de la hoja de datos anexada.

A continuación, se muestra una breve comparativa de ambos puentes H:

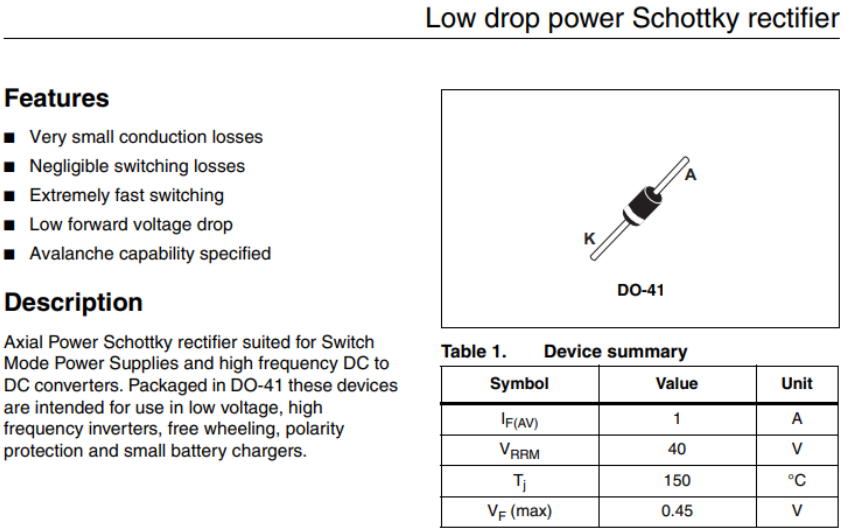
Tabla 5: Comparativa de la actualización del puente H en la maqueta de prácticas.

Parámetro	HIP4081	L298N
Tipo de Dispositivo	Controlador de MOSFET de puente H (no incluye los MOSFET de potencia)	Controlador de puente H integrado (incluye transistores de potencia)
Rango de Voltaje Motor	Hasta 80 V (usando MOSFET externos)	Hasta 46 V
Corriente de Salida	En función del MOSFET externo (pueden alcanzar decenas de amperios)	Hasta 2 A constantes por canal, 3 A en pico
Control de Puerta (Gate)	Salida de puerta para MOSFET de alta y baja, soporta conmutación rápida	Salidas integradas para motor, no requiere MOSFET externos
Protección	UVLO (bloqueo por baja tensión), no incluye diodos para motor	Incluye diodos de protección para cargas inductivas (si se adquiere como módulo)
Encapsulado	DIP, SOIC (necesita PCB con MOSFET externos)	ZIP-15, integrado y fácil de montar

Diseño	Mayor: requiere selección y diseño de MOSFET, disipación, drivers, PCB específico	Menor: solución compacta e integrada, ideal para pruebas y enseñanza
Ámbito educativo	Avanzado para entornos industriales donde la corriente y la eficiencia son críticos	Adecuado para prácticas con requerimientos moderados de potencia
Aplicaciones	Alta potencia y frecuencia donde la disipación y selección de MOSFET es crítica	Baja y media potencia donde la simplificación y facilidad de implementación son una ventaja

Con objeto de aprovechar al máximo el circuito integrado L298N, en el diseño de la placa electrónica final se plantean dos borneras para el control de dos motores. Debido a la finalidad de la maqueta, una de las tomas queda libre, por lo que puede ser utilizada para añadir ventilación forzada o el control de un segundo motor si fuera necesario.

Para que el doble puente H funcione de forma segura, es necesario diseñar una protección para las cargas inductivas. El motor de corriente continua de la maqueta genera unos picos de tensión muy elevados al cambiar el sentido de giro de forma brusca, entre otros estados (inicio del movimiento, introducción de un par externo, etc.). Esto se conoce como corriente de retorno y puede dañar tanto al propio circuito integrado L298N como a toda la placa electrónica. Se propone por ello introducir al diseño unos diodos de protección. El modelo seleccionado es el 1N5819, un diodo de potencia Schottky rectificador con una caída de tensión muy baja (ver figura 33), normalmente empleado en fuentes conmutadas y en convertidores continua/continua de alta frecuencia.



Low drop power Schottky rectifier

Features

- Very small conduction losses
- Negligible switching losses
- Extremely fast switching
- Low forward voltage drop
- Avalanche capability specified

Description

Axial Power Schottky rectifier suited for Switch Mode Power Supplies and high frequency DC to DC converters. Packaged in DO-41 these devices are intended for use in low voltage, high frequency inverters, free wheeling, polarity protection and small battery chargers.

DO-41

Table 1. Device summary

Symbol	Value	Unit
$I_{F(AV)}$	1	A
V_{RRM}	40	V
T_j	150	°C
V_F (max)	0.45	V

Figura 33: Breve resumen de la hoja de datos del diodo 1N5819. Extraído de la hoja de datos anexada.

La disposición de estos diodos se realiza mediante ingeniería inversa al módulo L298N. Tras el análisis del circuito de protección, se concluye que son necesarios ocho diodos Schottky (cuatro por cada puente H). Para la lógica del dispositivo, se debe alimentar con una tensión de 5 voltios, los cuales serán generados a partir de la fuente conmutada de 12 voltios presente en la maqueta, mediante un regulador de tensión. El componente seleccionado es el L7805CV con un encapsulado TO-220, por su popularidad en el mercado (fácil de conseguir). Este regulador admite una tensión de hasta 35 voltios en su entrada y los reduce a 5 voltios estables en su salida (ver figuras 34 y 35).

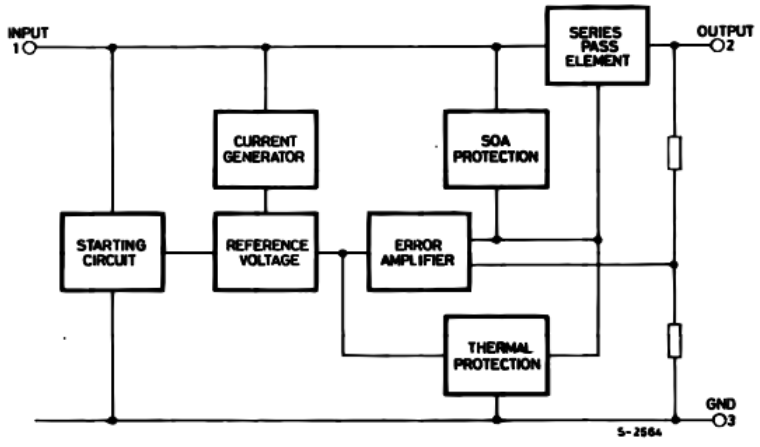


Figura 34: Diagrama de bloques del regulador L7805CV. Extraído de la hoja de datos anexada.

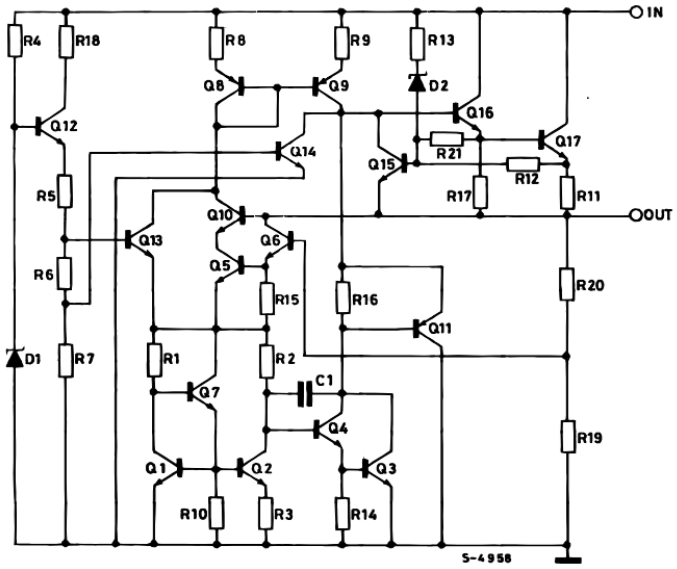


Figura 35: Conexiones internas del L7805CV. Extraído de la hoja de datos anexada.

Para lograr una alimentación estable con el menor ruido posible, el regulador contará con dos condensadores electrolíticos de desacoplo (uno en la entrada y otro en la salida), ambos con los siguientes valores: 35V 220uF.

4.1.4. MOTOR DE CORRIENTE CONTINUA

El motor JGA25-370 es un motor de corriente continua equipado con una caja reductora de engranajes metálicos. El modelo seleccionado para la maqueta funciona con una tensión nominal de 12V, cuya velocidad máxima a esa tensión es de 170 rpm (La velocidad máxima puede variar ligeramente entre motores de igual modelo). Gracias a su combinación de pequeño tamaño y una buena relación par/velocidad, es una opción muy utilizada en proyectos de robótica, pequeños vehículos, mecanismos eléctricos, etc. La caja reductora no solo reduce la velocidad de salida, sino que también incrementa el par disponible, permitiendo mover cargas mayores. Además, incorpora en la parte inferior del eje un codificador incremental que hace posible realizar lecturas de posición y velocidad. (ver figura 36).

Figura 36: Motor de corriente continua JGA25 – 370 con reductora y codificador incremental en cuadratura. Extraído de [15].

El codificador incremental que integra este motor se basa en el efecto Hall y su disposición de pines es la siguiente (ver figura 37):

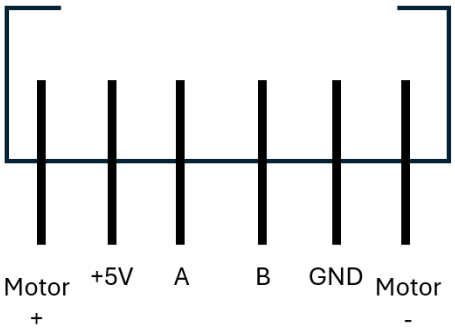


Figura 37: Disposición de pines codificador incremental. Elaboración propia.

Los terminales externos corresponden a la alimentación del motor, mientras que los pines +5V y GND dan tensión a los sensores de efecto Hall, los cuales permanecen activos hasta que el disco magnético atraviesa una de las pistas A, o B. Al realizar una lectura de estas dos pistas, es posible determinar el sentido de giro del motor, la posición y velocidad.

Tabla 6: Hoja de datos del motor de corriente continua JGA25-370, 12VDC 170RPM con reductora y codificador incremental de efecto Hall.

Parámetro	Valor
Modelo del motor	JGA25-370
Tensión nominal	12 VDC
Velocidad sin carga (motor sin reductora)	~5800 RPM
Velocidad sin carga (reductora 1:34)	~170 RPM
Corriente sin carga	~50–100 mA
Corriente nominal (bajo carga)	~400–800 mA (depende del par aplicado)
Corriente de arranque/bloqueo	~3–4 A (pico)
Par nominal (a 12V)	~1.2 kg·cm (0.12 N·m)
Par máximo (bloqueo)	~4.0 kg·cm (0.4 N·m)
Reducción de engranajes	1:34
Diámetro del eje de salida	6 mm (cilíndrico con chavetero)
Tipo de motor	Motor DC cepillado
Rodamientos	Casquillos metálicos autolubricados
Codificador	Incremental de efecto Hall de 2 canales
Resolución del encoder	13 pulsos/rev en eje del motor (~442 pulsos/rev salida)
Tensión de alimentación del encoder	3.3–5V
Corriente del encoder	~10 mA por canal
Señales del encoder	Salida digital cuadratura A/B
Temperatura de operación	-10 °C a +60 °C
Dimensiones motor + reductora	Longitud total ~65 mm; diámetro ~25 mm

Tabla 7: Características mecánicas y eléctricas del motor de corriente continua.

Parámetro	Símbolo / Unidad	Valor Aproximado
Tensión nominal	V [V]	12
Resistencia del devanado	R [Ω]	3.2
Inductancia del devanado	L [mH]	0.6
Constante de par	Kt [N·m/A]	0.017
Constante de fuerza contraelectromotriz	Ke [V·s/rad]	0.017
Inercia rotor sin reductora	Jm [kg·m²]	1.2×10^-6
Inercia equivalente en salida	Jeq [kg·m²]	~1.4×10^-4 (considerando reductora 1:34)
Coeficiente de fricción viscosa	B [N·m·s/rad]	~1.5×10^-5
Corriente sin carga	I_0 [A]	0.05–0.1
Velocidad sin carga	ω_0 [RPM]	~170 (salida)

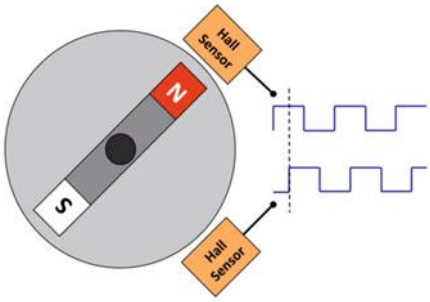


Figura 38: Principio de funcionamiento de las pistas A y B en el codificador incremental. Extraído de [16].

Al realizar la lectura digital de ambas pistas (ver figura 38), mientras se realiza una vuelta completa al eje (sin reductora), arroja un resultado de 22 pulsos por vuelta. En el modelo seleccionado para la maqueta, la reductora que presenta es de 34 vueltas a 1 (1:34). Esto quiere decir que para completar una vuelta completa del motor con reductora son necesarios 22·34 pulsos, siendo esto un total de 748 pulsos por vuelta.

4.1.5. MEDICIÓN DE CORRIENTE

La ventaja principal de hacer uso del integrado ZIP-15 L298N sin su módulo, es la posibilidad de aprovechar dos de los pines para medición de corriente (Sense A y Sense B, tal y como detalla su hoja de datos). El primer pin, corresponde al motor DC (pin Sense A) mientras que el otro pin de medición (Sense B) no se emplea en ningún caso, así que se lleva directamente a GND. Para realizar la medición, es necesario incorporar una resistencia de tipo shunt cuyo valor sea lo suficientemente bajo como para no afectar al control del motor, pero fácil de leer a través del microcontrolador. Esta resistencia (cerámica) en la maqueta se denomina "R_{SA}" y el valor elegido para su diseño es de 1 ohmio, con una disipación de hasta 5 watios (ver figura 39).

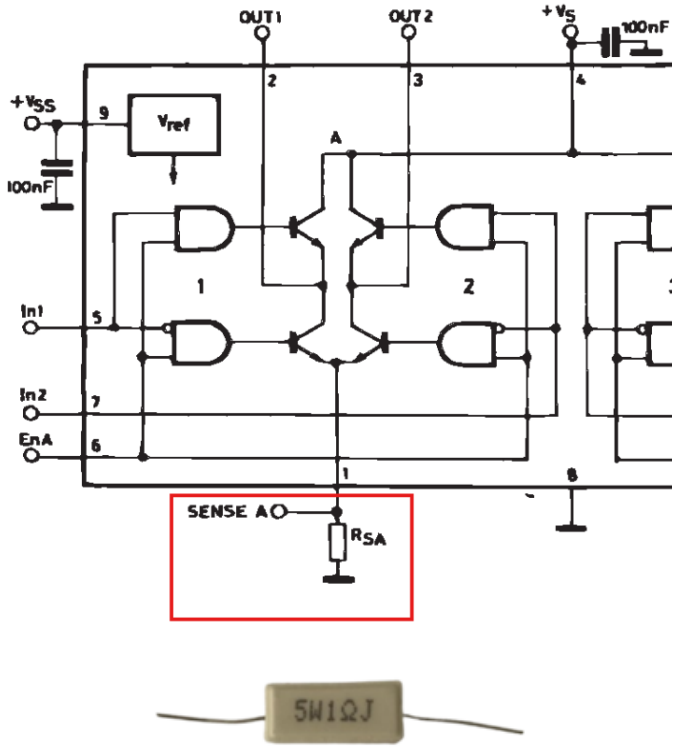


Figura 39: Ubicación de la resistencia Rsa para medición de corriente. Extraído de la hoja de datos anexada y editado.

La resistencia shunt tiene un valor aproximado de 1 ohmio. Se ha determinado así para que la caída de tensión en la resistencia coincida con el valor de corriente aproximado que circula por ella. De esta manera, al leer el valor analógico de la tensión entre el pin Sense A y GND, se obtendrá la corriente del motor. De forma teórica, es razonable este procedimiento. Sin embargo, en la realidad, el motor se somete a tensiones entre 12 y -12 voltios, por lo que el consumo de corriente es del orden de miliamperios si no se aplica un par externo. Esto equivale a una caída de milivoltios en la resistencia shunt. Por ello, la corriente consumida por el motor en su zona muerta y en los primeros valores de

tensión aplicada, no son leídos por el microcontrolador (son del orden de 0.001 voltios). El ESP32 no es capaz de interpretar los datos de un valor de tensión tan pequeño. Se sugieren varias alternativas para solucionar el problema, modificando las características de entrada de los convertidores ADC:

- **Hacer uso de un amplificador operacional** (tal y como se sugería en el controlador PIC de la antigua maqueta).
- **Aumentar la resolución de los pines de entrada** del microcontrolador (hasta 12 bits, equivalente a 4096 estados).

Tabla 8: Resolución de los pines de lectura analógicos del conversor analógico digital.

Resolución	Estados
9 bits	512
10 bits	1024
11 bits	2048
12 bits	4096 (máxima precisión)

Como se indica en la tabla, la máxima resolución que permite el ESP32 DevKit V1 es de 12 bits. No influye en ningún aspecto importante, debido a que el problema reside en la tensión tan pequeña de medición que existe. Por mucha resolución que el microcontrolador sea capaz de ofrecer, no interpreta valores de tensión tan cercanos al cero.

- **Disminuir la atenuación de los pines** (aumentando la posibilidad de introducir ruido en la señal). Este método resulta más práctico y logra abordar un rango más amplio de valores de tensión pequeños. Sin embargo, la señal no es clara y se genera mucho ruido. En la siguiente tabla se muestran las configuraciones desde software que pueden realizarse para establecer una atenuación concreta en cada pin. Incluso con una atenuación de 0 dB, las tensiones del orden de milivoltios cercanas a 0 no son legibles por el ESP32.

Tabla 9: Rango de tensiones permitidas para lectura del conversor analógico digital. Atenuaciones.

Atenuación	Rango de tensiones aprox.	Descripción
0 dB	0 – 1.1 V	Más precisión en tensiones pequeñas
2.5 dB	0 – 1.5 V	Útil en sensores de rango medio
6 dB	0 – 2.2 V	Compromiso ideal para muchos usos
11 dB	0 – 3.9 V	Para señales de hasta ~3.3V (por defecto)

Como conclusión, se determina que la mejor opción es realizar una amplificación de la señal medida a través de un amplificador operacional. En la maqueta previa se sugería realizar esta tarea con el amplificador operacional LM358P. Debido a que es un circuito integrado bastante común y versátil, ha sido seleccionado para la nueva placa electrónica.

El LM358P es un pequeño dispositivo con un encapsulado DIP-8 que integra dos amplificadores operacionales (ver figura 40). Ambos amplificadores comparten las tensiones de saturación, pero sus salidas y entradas son independientes.

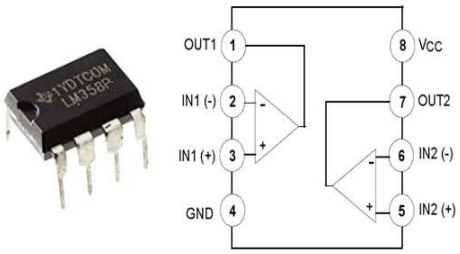


Figura 40: Diseño del OpAmp LM358P. Extraído de la hoja de datos anexada.

Se configuran así dos de los pines del microcontrolador ESP32 para recibir las señales. Uno de los pines recibe el valor de tensión “en crudo”, mientras que el segundo pin recibirá la señal amplificada. En resumen, se utiliza uno de los dos amplificadores disponibles en el LM358P.

Para poder realizar un circuito capaz de amplificar la señal de entrada, se debe seguir la configuración de un amplificador no inversor (ver figura 41). Este diseño es el más básico para el proyecto, además de determinar la ganancia de la señal a partir del valor fijo en las resistencias:

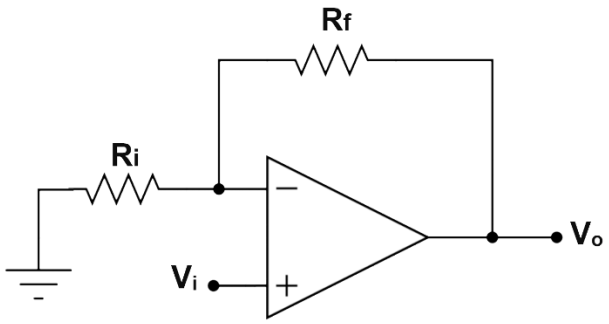


Figura 41: Configuración de OpAmp para amplificador no inversor. Extraído de [17].

La fórmula matemática que rige el comportamiento del esquema de la figura 26 es la siguiente:

$$V_o = \left(\frac{R_f}{R_i} + 1\right) \cdot V_i$$

Donde "*Vo*" representa la tensión amplificada a la salida, "*Vi*" la tensión de entrada sin amplificar y "*Rf*" junto con "*Ri*" representan la ganancia de la señal de salida "*Vo*". Tras analizar los mejores valores de "*Rf*" y "*Ri*" para una buena amplificación sin llegar a saturar el operacional, se concluye que dichos valores son:

$$R_i = 1K\Omega$$

$$R_f = 44K\Omega$$

De forma teórica, se determina que la señal de salida es amplificada 45 veces respecto a la señal de entrada. Realizando pruebas de tensión aplicada al motor, la siguiente tabla recoge las lecturas de corriente y tensión con ayuda de un multímetro.

Tabla 10: Tensiones y corriente que atraviesa el motor DC para la adecuación de los datos. Valores amplificados por el operacional.

Tensión motor (V)	Corriente motor real (mA)	Tensión Rshunt (mV)	Tensión Amplificada (V)
12	78.2	78.3	3.23
11	75.2	72.5	2.72
10	73.3	70.2	2.51
9	68.8	68.7	2.29
8	65.4	65.4	2.06
7	60.4	61.6	1.84
6	57.2	57.4	1.60
5	51.6	51.8	1.35
4	46.4	44.1	1.10
3	34.3	33.2	0.81
2	21.3	20.3	0.53
1	6.2	6.0	0.23
0	0	0	0
-1	-6.6	5.8	0.23
-2	-24.9	25.2	0.55
-3	-35.5	-34.2	0.82
-4	-46.9	-46.9	1.10
-5	-54.9	-54.8	1.36
-6	-61.8	62.2	1.62
-7	-65.2	65.7	1.86
-8	-69.4	69.2	2.10
-9	-71.1	71.1	2.32
-10	-71.8	72.6	2.49
-11	-74.5	74.7	2.63
-12	-77.9	78.1	3.08

Los valores marcados en rojo son aquellos que no puede interpretar el microcontrolador porque es confundido con señal de ruido externo. Con una amplificación como la determinada por las resistencias seleccionadas, se obtiene un rango

de tensiones bastante elevado, llegando a los 3.3V cuando la corriente es de 80mA aproximadamente. Manteniendo una atenuación de 11 dB y una resolución de 12 bits, es posible abarcar un amplio rango de valores de corriente cercanos al cero. Se ha de tener en cuenta que la tensión de saturación positiva teórica son 5V, pero debido a pérdidas, la saturación real es de 3.675V. La tensión de saturación negativa es de 0V. La corriente que circula por el motor emplea los 12 bits en el rango positivo de su valor, debido a que la corriente que atraviesa la resistencia shunt no cambia de signo por la configuración de sus transistores en el puente H. Por ello, la resolución no se reduce a la mitad, sino que el signo de su valor se introduce de forma posterior a través de software, en función del sentido de giro que el motor lleve en un instante dado.

$$Resolución = \frac{80\ mA - 0\ mA}{4096\ pasos} = 0.01953mA = 19.5uA$$

Se concluye así que la resolución de corriente para valores de tensión normales en el motor es muy precisa. Sin embargo, también es necesario realizar la lectura de la corriente de bloqueo del motor (en inglés *stall current* o I_{stall} en las hojas de datos). Esta corriente es muy superior a la que trabaja en condiciones normales el motor DC y se origina al aplicar un par externo al motor mientras es alimentado. Esta corriente es capaz de llegar a los 2 amperios. Como ya se ha demostrado, el amplificador satura si mide corrientes mayores a 80 mA, por lo que es necesario encontrar una solución para valores superiores a este. Como el integrado LM358 incorpora dos amplificadores operacionales, es posible asignar este segundo operacional para medir corrientes más grandes. De esta manera, se realizan dos mediciones. La etapa de medición fina (corrientes en condiciones de trabajo normales para el motor, suficiente resolución) y la etapa de medición en bruto. Esta segunda etapa produce caídas de tensión en la resistencia shunt lo suficientemente grandes como para ser medidas directamente por el microcontrolador. Como resultado, no es necesario utilizar el segundo amplificador, sino que, en su lugar, para corrientes entre 0 y 80mA se utilizará el primer operacional, mientras que, para valores superiores, se tomará la medida de forma directa de la resistencia. Es por este motivo que la señal sin amplificar pasa antes por uno de los pines del microcontrolador y luego es amplificada para ser leída en otro de los pines.

4.1.6. OTROS ELEMENTOS DE LA PLACA A DESARROLLAR

Se han ubicado los encapsulados TO-220 y ZIP-15 uno cerca del otro para añadir un disipador de calor común (ver figura 42). De esta manera, se refrigeran ambos circuitos integrados, tal y como sus hojas de especificaciones lo requieren. De no ser así, el regulador de tensión se calienta de forma excesiva y el doble puente H pierde eficiencia.

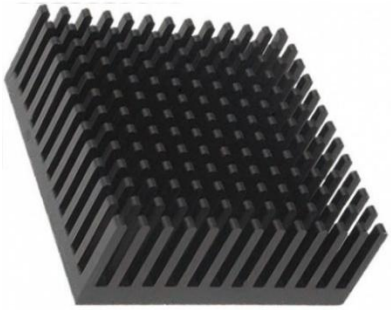


Figura 42: Disipador de calor empleado en la placa electrónica. Extraído de [18].

Para facilitar el reemplazo de algunos componentes que suelen presentar problemas o estropearse en la placa electrónica (microcontrolador ESP32, amplificador LM358P) se han soldado conectores hembra y un socket DIP-8 para acceder de forma rápida a su sustitución, tal y como se muestran en la figura 43.

Figura 43: Socket DIP-8 para amplificador y pines hembra para ESP32. Elaboración propia,

A su vez, se han instalado tres borneras (una para alimentación y dos para la salida de los puentes H) y varias hileras de pines de tipo macho para acceder de forma sencilla a los pines del microcontrolador, además de añadir a la placa grupos específicos de conexión para la pantalla I2C LCD1602 y el codificador incremental rotativo KY-040 (ver figura 44). Finalmente, unos indicadores LED para notificar al usuario de qué tipo de control se encuentra operando la maqueta.

Figura 44: Codificador rotativo KY-040 (izquierda), pantalla I2C LCD 1602 (centro) y pines macho (derecha). Elaboración propia,

El programa de diseño de PCB que se ha empleado es KiCAD (ver figura 45), un software gratuito especializado en el diseño de circuitos electrónicos y esquemas eléctricos, entre otras capacidades. Se desarrolló en 1992 por Jean – Pierre Charras como software de código abierto, y desde entonces ha sido sustentado por una comunidad de desarrolladores.

Figura 45: Logotipo del software libre KiCAD. Extraído de [19].

Es una herramienta muy popular debido a que no es necesario tener una licencia, además de ser multiplataforma (macOS, Linux, Windows). Sus principales características son las siguientes:

- Editor de esquemas: diseño de circuitos electrónicos de forma visual, a través de bibliotecas de símbolos y definición de reglas eléctricas.
- Editor de PCB: Se trata de una herramienta para diseñar placas de circuito impreso en dos o más capas, con la posibilidad de añadir extensiones como rutado automático y generador de archivos gerber (necesarios para su fabricación).
- Gestor de bibliotecas: Lugar donde se almacenan todas las huellas de los componentes electrónicos. Permite editar o crear símbolos y componentes nuevos para proyectos.

- Visualizador en 3D: Es una herramienta muy útil para previsualizar el diseño final de la placa electrónica. Se trata de un renderizado tridimensional de esta.
- Herramientas de simulación y verificación: KiCAD integra un simulador como LTSPICE para verificar las conexiones eléctricas e integridad del diseño en general.

A continuación, se procede a mostrar el esquema eléctrico realizado con los componentes y distribución mencionadas anteriormente para el diseño de la placa electrónica (ver figura 46), así como su renderizado en 3D (ver figuras 47 y 48). Se trata de una placa compuesta por dos capas, con componentes de tecnología de agujeros pasantes, o THT en inglés (Through - Hole Technology). Se ha decidido de esta manera para evitar la complejidad en su diseño (capa superior y capa inferior), además de la facilidad de fabricación de esta a la hora de integrar los componentes a la placa. Realizar una soldadura con componentes THT es mucho más sencillo que hacerlo con dispositivos de montaje en superficie (SMD).

Tabla 11: Función de los pines correspondientes al microcontrolador ESP32

Pin ESP32	Nombre de la señal	Descripción
3V3	3.3V	Alimentación lógica de 3.3 V
GND	GND	Masa común
IO34	Fase_A	Señal codificador motor canal A
IO35	Fase_B	Señal codificador motor canal B
IO33	EN_B	Habilitación puente H canal B
IO26	EN_A	Habilitación puente H canal A
IO25	Dir_Mot+	Dirección positiva motor
IO27	Dir_Mot-	Dirección negativa motor
IO14	SensorCorriente	Medida tensión sensor de corriente en bruto
IO12	OpAmpCorriente	Medida salida OpAmp corriente (amplificada)
IO21	Display_SDA	Datos I2C display LCD
IO23	Display_SCL	Reloj I2C display LCD

IO19	Encoder_CLK	Señal CLK codificador rotativo KY-040
IO18	Encoder_DT	Señal DT codificador rotativo KY-040
IO17	LED_LOCAL	LED indicador modo local
IO16	LED_SIMULINK	LED indicador modo Simulink
IO15	Encoder_SW	Botón encoder rotativo KY-040
5V	5V	Alimentación externa de 5 V (puede alimentar LCD)

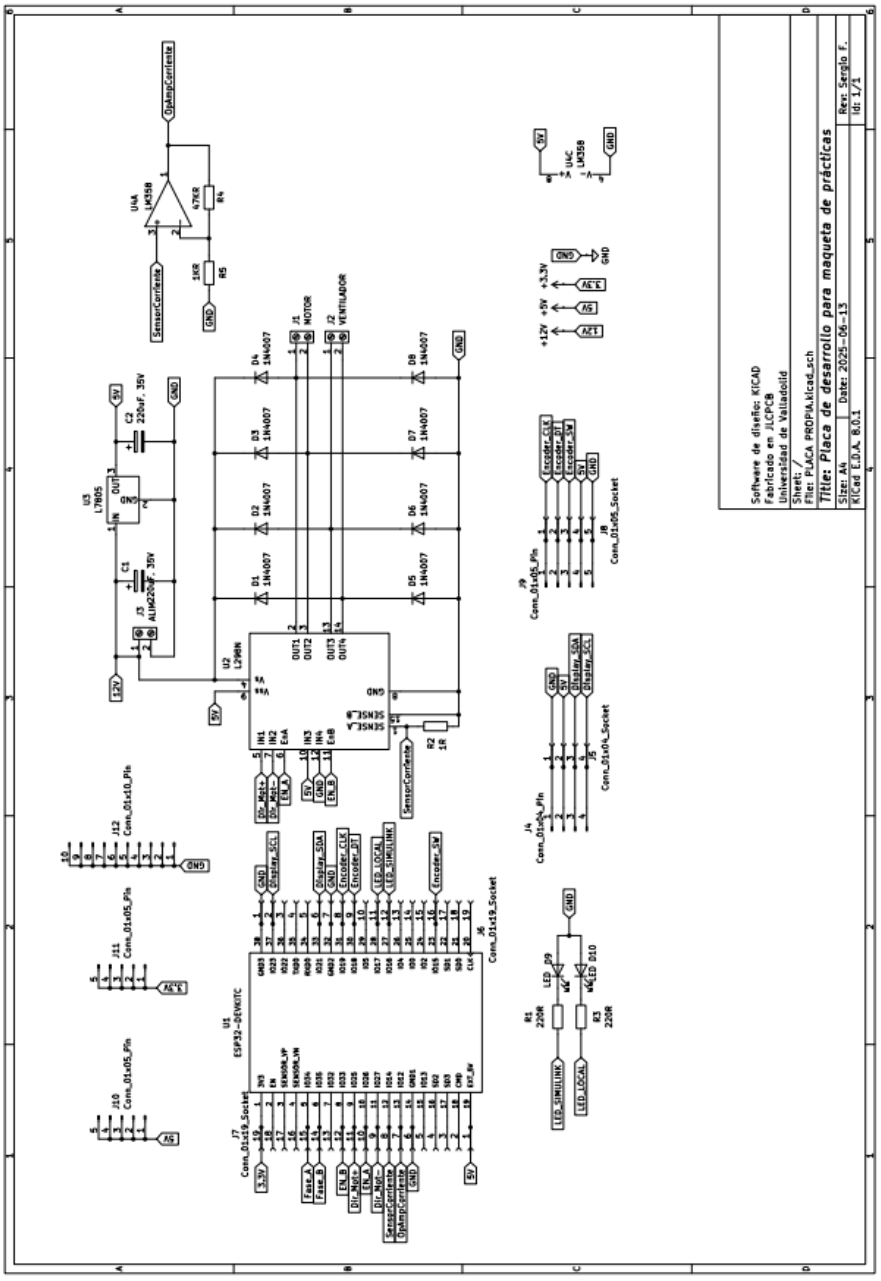


Figura 46: Esquema de conexiones eléctricas de la placa. Elaboración propia.

El resto de esquemáticos y capas de fabricación se encuentran adjuntas en el anexo del documento, así como en el capítulo "Anexos". Por otro lado, se han diseñado unos modelos 3D para la sujeción del motor de corriente continua, la pantalla LCD y el codificador KY040 para el interfaz humano – máquina, también anexados al documento. Estos diseños han sido realizados con el software Fusion 360 e impresos en 3D.

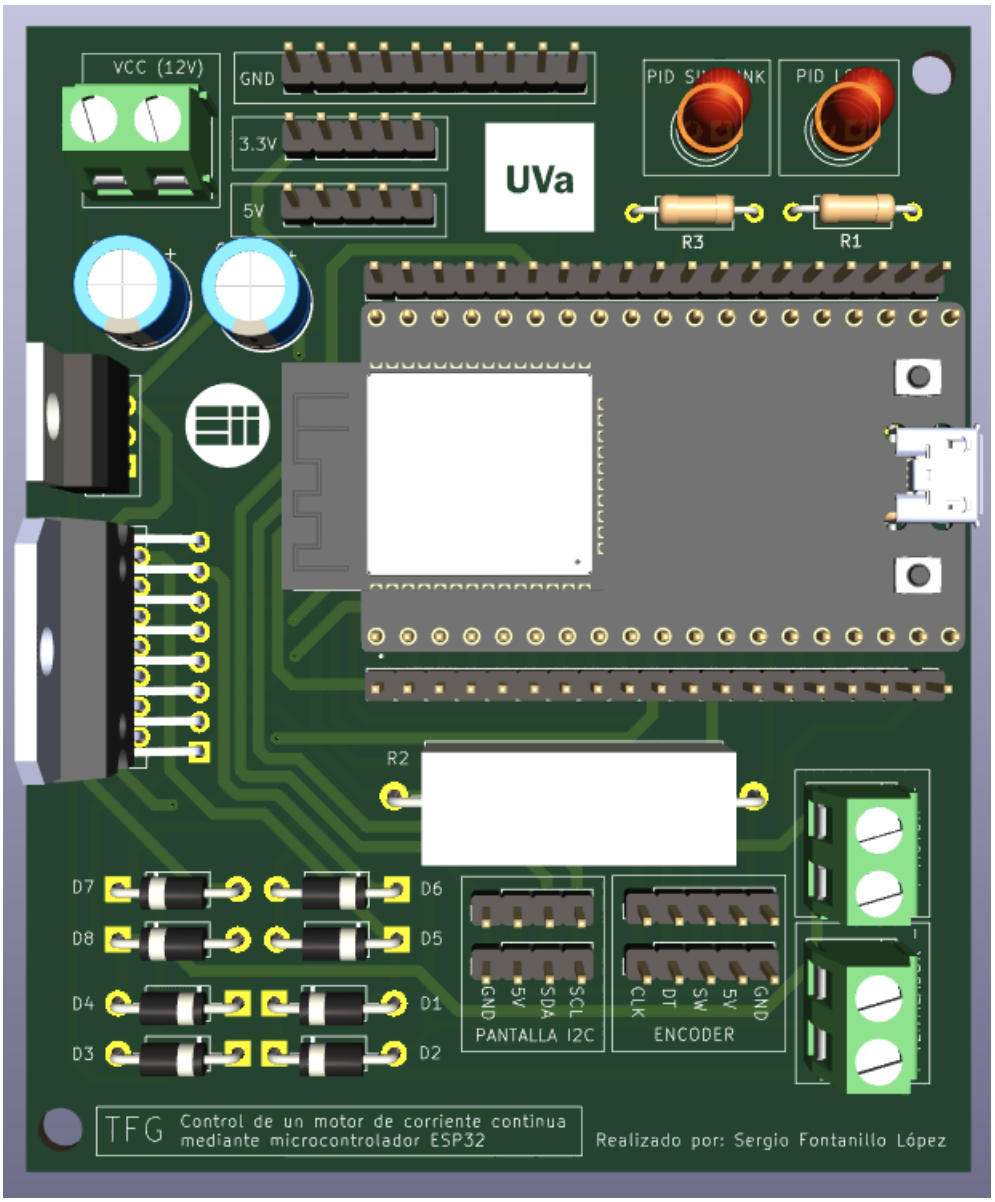


Figura 47: Vista superior de la placa en el visor 3D de KiCAD. Elaboración propia.

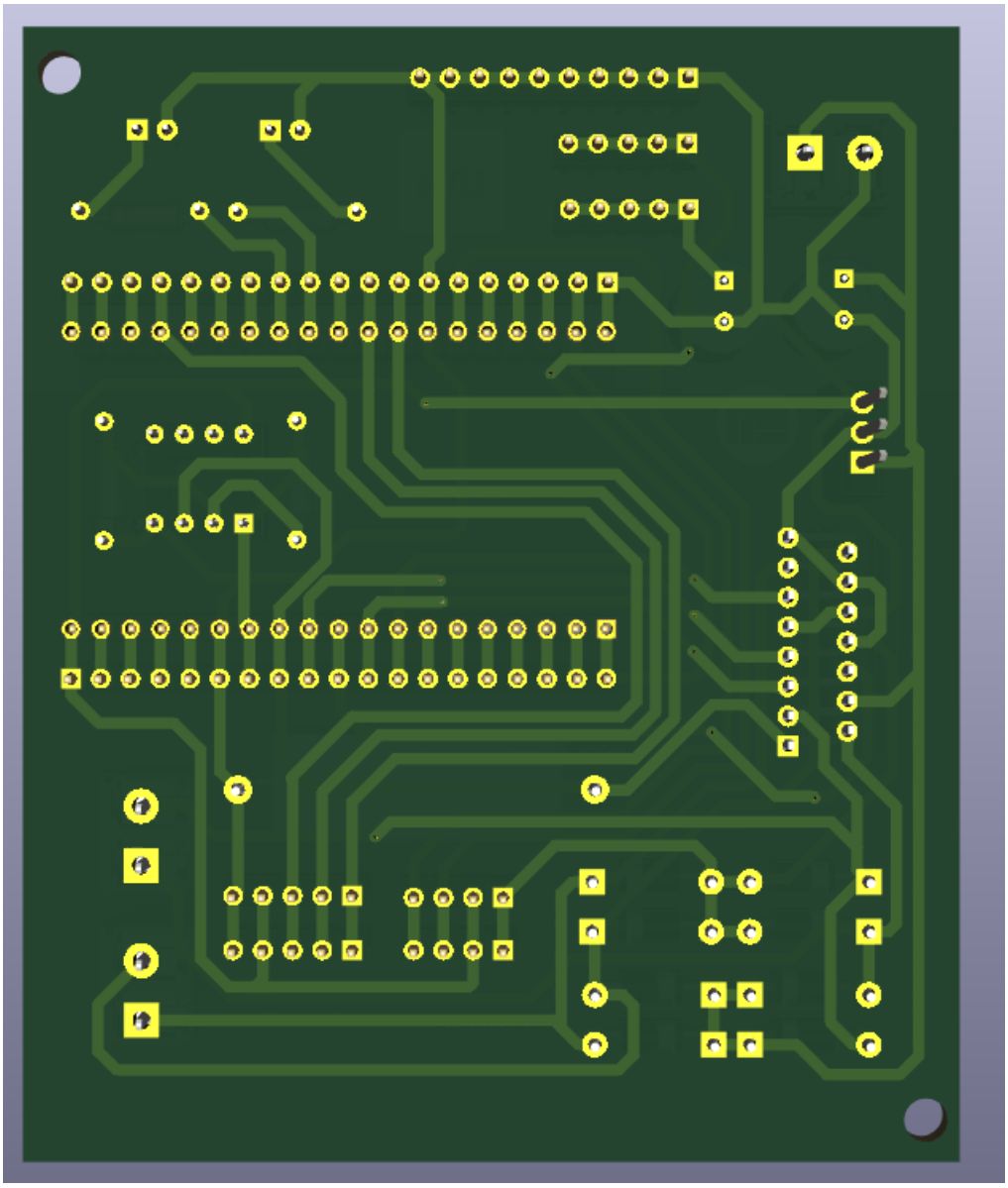


Figura 48: Vista inferior de la placa desde el visor 3D de KiCAD. Elaboración propia.

La placa electrónica fue fabricada por la empresa JLCPCB, la cual completó su envío al cabo de veinte días. Se realizó una realimentación en todo momento de su proceso de fabricación, así como la calidad del envío final (ver figura 49).

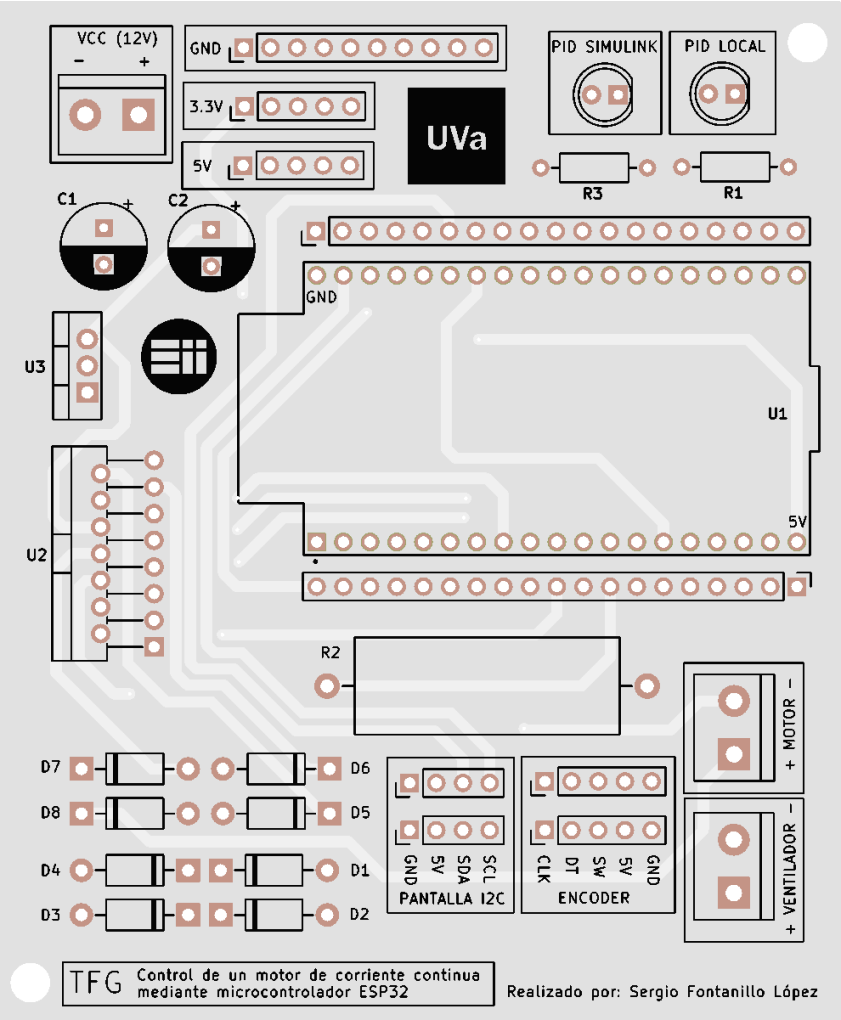


Figura 49: Resultado de la fabricación de la placa electrónica (sin el amplificador, debido a una versión anterior de esta). Elaboración propia.

Una vez verificado y soldado todos los componentes, realizado pruebas de continuidad, valores de tensión adecuados y conectividad con los periféricos, se procede al montaje de la maqueta (figura 50):

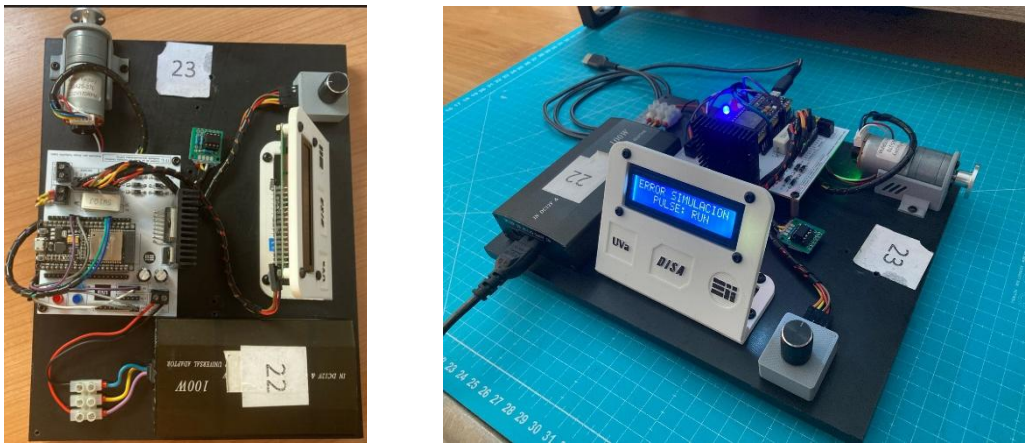


Figura 50: Maqueta ensamblada (vista en alzado y principal). Elaboración propia.

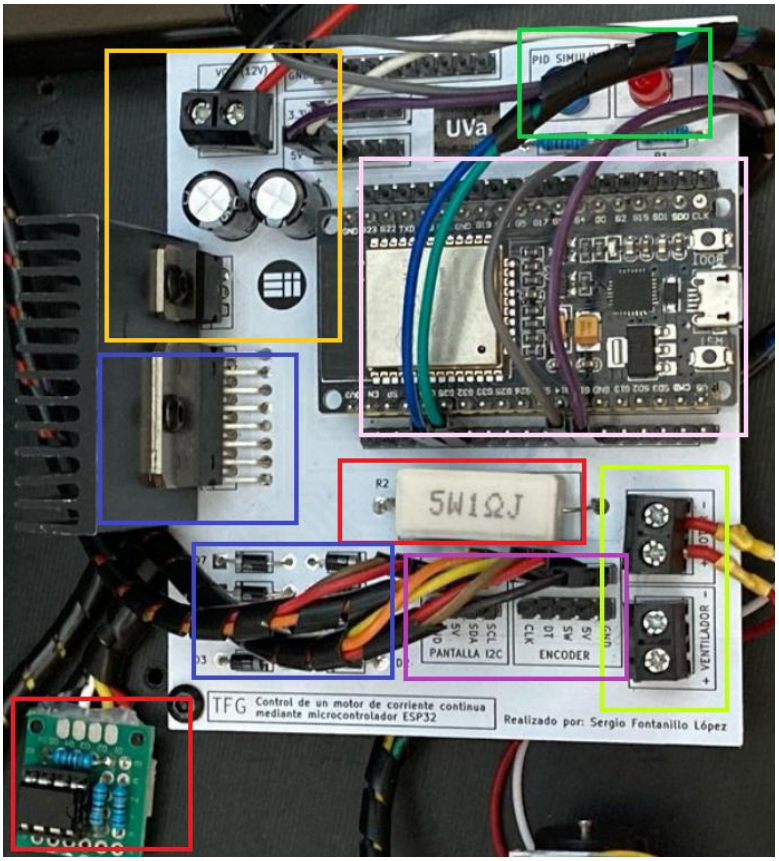


Figura 51: Etapas de la maqueta. Elaboración propia.

Como se observa en la figura 51, la placa electrónica se divide en diferentes etapas. La primera de todas, comenzando desde arriba a la izquierda, es la etapa de alimentación (marcada en naranja). Se introducen 12 voltios en la entrada y se obtienen 5V de forma regulada. En el recuadro verde situado arriba a la derecha se muestran dos indicadores LED que establecen el tipo de control al que está sometido la maqueta (Identificación, Local, o Simulink, representado por "apagados", "rojo" y "azul"

respectivamente). Más abajo se encuentra la etapa de control coloreada en rosa, correspondiente al microcontrolador ESP32. En azul oscuro, se muestra la etapa de potencia del motor, compuesta por el doble puente rectificador y sus correspondientes diodos de protección. En rojo, se detalla la etapa para medición de corriente. En este caso, la placa es una versión anterior a la actual, por lo que el amplificador no se encuentra embebido debajo del microcontrolador. En el resto de las maquetas, será de esta manera. En morado se distinguen dos "puertos" compuestos por dos filas de hembra y machos para la conexión de la pantalla LCD I2C y el codificador incremental KY - 040 del menú. Finalmente, en verde lima se muestran los puertos de salida para los motores, concretamente para el JGA25-370 y para agregar una ventilación de forma opcional en caso de no utilizar esta segunda bornera.

Figura 52: Maqueta en funcionamiento. Modo de identificación sin conexión. Elaboración propia.

En la figura 52 se observa cómo la interfaz HMI indica el estado en el que se encuentra la maqueta (paro), la variable de control que se quiere controlar (velocidad) y el tipo de entrada que se le va a aplicar al sistema (en este caso, escalón). Se detallará más sobre su funcionamiento en siguiente capítulo de firmware. Las siguientes imágenes muestran los tres tipos de modos que puede tomar la maqueta: el primero se trata de identificación sin conexión (figura 51). En este modo, no se realiza ningún lazo de control. Únicamente se introduce una entrada de tensión y la interfaz arroja los valores de velocidad y corriente correspondientes a dicha tensión. Para una mejor visualización, es posible conectar la maqueta vía USB al "Plot Serial" de Arduino IDE, donde se mostrarán los datos correspondientes en la gráfica. Ambos LEDs indicadores permanecen apagados.

Figura 53: Modo de control Simulink. Conexión directa con MATLAB/Simulink. Elaboración propia.

Para el modo de control de la figura 53, se establece la configuración necesaria para el envío de tramas de forma serial al entorno de Simulink. De esta manera, es posible modificar los valores de entrada de tensión tanto del software como de la maqueta, así como realizar lazos de control desde el propio entorno de Simulink. Para ello, se ha realizado un bloque específico para la comunicación con la maqueta. El indicador LED azul permanece encendido. El procedimiento necesario para seguir es el siguiente:

- Iniciar la simulación desde Simulink con los valores de entrada deseados (es posible que la primera ejecución y compilación del programa muestre un error de desconexión, debido al tiempo excesivo inicial que tiene MATLAB al compilar por primera vez. La maqueta se desconecta por seguridad).
- Durante la simulación, es posible modificar los valores de entrada en tiempo real, así como visualizar las variables en la gráfica.
- Al finalizar la simulación, se ejecuta un script de forma automática que recoge en vectores los valores de cada variable y los organiza en una gráfica con el tiempo real. Dentro del script existe un panel de control para el usuario en caso de querer almacenar los datos en un archivo Excel.

Todas estas características serán analizadas paso por paso en el siguiente capítulo (Firmware), donde se explicará el código empleado y las interfaces disponibles para el usuario.

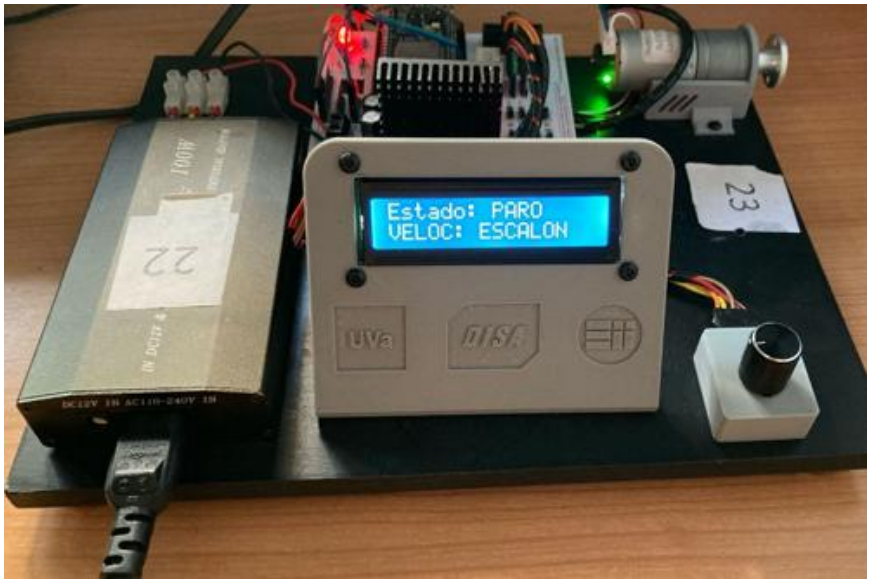


Figura 54: Modo de control local. Elaboración propia.

En la figura 54, se muestra el último modo de control de la maqueta. Se trata de la implementación de un PID interno en el microcontrolador, cuyos parámetros pueden ser modificados a través del menú dispuesto en la maqueta, así como los tipos de entrada (en este caso, referencia de velocidad en revoluciones por minuto) deseadas. En caso de necesitar visualizar las variables de forma gráfica, es posible conectarlo vía microUSB al "Plot Serial" del sotfware Arduino IDE.

CAPÍTULO 5:

DESARROLLO DEL FIRMWARE

5. DESARROLLO DEL FIRMWARE

A lo largo del capítulo, se realizará el estudio del código que integra el microcontrolador, así como de las funcionalidades que aporta a la maqueta de prácticas y que permiten la interacción con el motor de corriente continua, la adquisición de señales, la visualización por pantalla de la maqueta y las configuraciones de comunicación necesarias para lograr la compatibilidad con Arduino IDE y MATLAB/Simulink.

El firmware se estructura en varios módulos separados por la lectura de los codificadores rotativos, el control interno PID, la memoria EEPROM simulada, la gestión de la interfaz HMI, el control del motor y el código principal (que integra los tipos de entrada y conectividad con los softwares mencionados, así como los diferentes modos de operación según la práctica que se desee desarrollar).

5.1. ESTRUCTURA DEL CÓDIGO

Como se ha descrito anteriormente, el código está organizado en varios archivos que encapsulan un total de 1920 líneas de código, donde cada sección almacena las funcionalidades principales de la maqueta. Estos archivos son los siguientes:

- Código principal (CODIGO_MAQUETA.ino): contiene la inicialización del sistema, el bucle principal, las lecturas de entradas, conectividad por puerto serie y la lógica de control de alto nivel. En general, se encarga de declarar todos los pines de entradas y salidas existentes en la maqueta, además de los valores de estado, control, entradas, y variable (velocidad o posición) mostradas en la pantalla LCD. Además, alberga funciones específicas como la transmisión por puerto serie al configurar el control con Simulink, o la medición de corriente amplificada por el operacional. Por último, se encarga de inicializar le control de velocidad o de posición. Para este proyecto, se ha trabajado exclusivamente en el control por velocidad, pudiendo ser el de posición una futura actualización de la maqueta asignada a un nuevo proyecto de fin de carrera.

Por otro lado, se encuentran los módulos auxiliares que desempeñan funciones específicas dentro del código principal. Estos módulos son:

- PID_ESP32: es la implementación de la lógica del control PID interno en el microcontrolador.
- Menu: reúne todos los aspectos a nivel de código para mostrar y navegar por el menú interactivo a través de la interfaz HMI (pantalla I2C y encoder KY – 040).
- Movement: sección que se encarga de gestionar la dirección y velocidad del motor en función de la tensión de entrada, así como de otras funciones (por ejemplo, la simulación de una carga en el motor).
- EEPROM: elaboración de código que aprovecha las capacidades de la memoria flash del ESP32 y emula una memoria EEPROM, desde la que se almacenan parámetros específicos configurables por el usuario a través del menú.
- Librerías exclusivas para el uso del codificador KY-040 y el codificador incremental situado en el motor de corriente continua.

5.1.1. LECTURA DE SEÑALES

El sistema recopila una serie de señales para supervisar el estado del motor y realizar su control. Tales señales son generadas por el codificador incremental integrado en el motor (sensor de efecto Hall), las cuales consisten en un tren de pulsos que se contabilizan mediante interrupciones. A partir de las interrupciones, es posible calcular la velocidad angular en revoluciones por minuto, teniendo en cuenta la relación entre el número de pulsos y el tiempo de muestreo, además de la relación de transmisión existente en la reductora. Por otro lado, se determina el sentido de giro en función del patrón de ambas señales del codificador en cuadratura (dos sensores Hall dispuestos de forma ortogonal).

Para la lectura de la corriente se emplea un amplificador operacional, debido a que la tensión que cae en la resistencia shunt es demasiado pequeña como para obtener una resolución adecuada (ya se explicó en la medición de corriente). Esta señal, antes de ser amplificada, atraviesa uno de los pines del microcontrolador. A la salida del amplificador, se emplea otro de los pines para la medición del valor. De esta forma, se obtienen dos mediciones. El motivo de esta configuración se debe a la búsqueda de una resolución aceptable. El microcontrolador permite tensiones de lectura desde los 0V a los 3.3V. El problema reside en la zona muerta del motor, donde la corriente al aplicar tensiones pequeñas es prácticamente despreciable, y el lector analógico digital no hacía distinción entre esos valores y el 0, de manera que a partir de una tensión umbral, en la cual el motor comenzaba a moverse, se originaba un valor de corriente espontáneo (sin tener una

retroalimentación de los valores de corriente menores al valor de tensión umbral) concluyendo en una medición de corriente poco precisa.

Al añadir un amplificador operacional a una resistencia cuyo valor es de 1ohmio, el valor de tensión que caía coincidía con el valor de corriente multiplicado por una ganancia (determinada por las resistencias del operacional). El problema residía en buscar un valor lo suficientemente grande como para reconocer corrientes en la zona muerta del motor, pero lo necesariamente pequeño para no saturar el amplificador. Se llegó a la conclusión de emplear un valor de 47KΩ y 1KΩ, obteniendo una resolución de 19.5 uA. Al ser valores tan pequeños, el error aumenta en gran medida, por lo que es necesario realizar una media móvil en tiempo real para adecuar este valor. Otra consideración es la alimentación del amplificador. Al introducir 5V en la tensión de saturación, en la realidad el valor máximo capaz de alcanzar es de 3.6V.

5.1.2. CONTROL DEL MOTOR

El control del motor se realiza generando una señal PWM proporcional a la tensión de entrada (con una frecuencia de 5KHz) y los pines de dirección del puente H L298N. Se opta por utilizar una resolución de 8 bits en los pulsos de ancho modulado, correspondiendo 12 voltios de entrada al motor a un valor de 255 en la señal generada. El resto de los valores se encuentran interpolados en la recta que une el punto mencionado y el valor nulo de ambas señales (0V de entrada y 0 como valor de PWM). En función del signo empleado por la tensión de entrada, se determina el sentido de giro del motor. En el modo de control “IDENTIFICACION” y “SIMULINK”, las señales de entrada vienen determinadas por el propio microcontrolador, el cual se encarga de generarlas. Los valores como el periodo o la amplitud son configurables por el usuario y pueden ser posteriormente almacenados en la memoria EEPROM. Además, el control por Simulink también permite tomar de sus bibliotecas las fuentes “escalón” o “rampa” como entradas, e incluso realizar lazos de control (para ello, es necesario marcar una casilla dentro del bloque de la planta en Simulink para elegir el origen de las entradas a priorizar). En el modo de control “LOCAL”, se realiza un lazo de control PID cuyos parámetros están descritos en la memoria EEPROM y son editables. Al seleccionar este modo, el usuario puede modificar el valor de consigna (Setpoint) deseado y el control PID actuará en consecuencia a dicho cambio.

5.1.3. INTERFAZ HMI: MENÚ Y NAVEGACIÓN

La interfaz de usuario está basada en una pantalla LCD 1602 I2C, donde se muestra la información de estado, las variables en tiempo real y las opciones del menú. Con ayuda del encoder KY – 040, es usuario puede navegar entre las opciones, modificar parámetros, el tipo de entrada o el tipo de control. En encoder detecta rotación (incremento/decremento) y pulsación (selección).

El firmware gestiona un sistema de menús jerárquicos, con las siguientes funciones:

- Visualización de valores (velocidad dada en RPM, el estado marcha/paro, tipo de entrada, tensión y corriente).
- Selección del modo de operación (tipo de control / entradas)
- Ajuste de parámetros PID para posición o velocidad. Cada tipo de variable posee seis parámetros PID. Tres de ellos corresponden a las constantes P, I, D, y los otros tres son parámetros aplicados en el rango de zona muerta del motor (donde al aplicar una tensión, el motor no es capaz de generar movimiento).
- Almacenamiento en memoria EEPROM. Al reiniciar la maqueta, inicializa con los parámetros grabados por el usuario.
- Selección de opciones / modificar valores con encoder.

La tasa de refresco de la pantalla se realiza mediante interrupciones, permitiendo mostrar de forma fluida el contenido del menú sin “intermitencias” al actualizar los valores cada cierto tiempo.

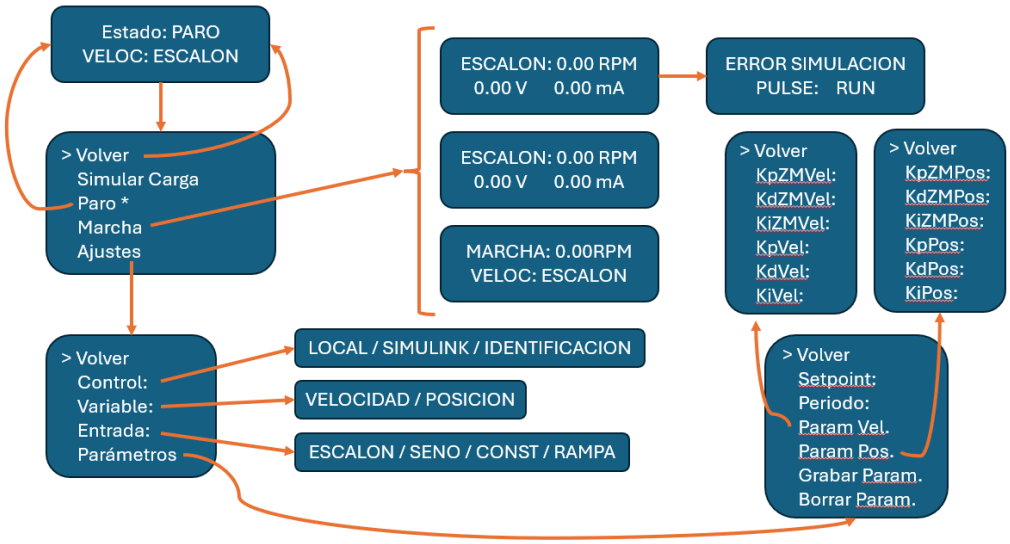


Figura 55: Diagrama de opciones del menú HMI. Elaboración propia.

La tasa de refresco de la pantalla se realiza mediante interrupciones, permitiendo mostrar de forma fluida el contenido del menú sin "intermitencias" al actualizar los valores cada cierto tiempo. Para acceder al menú de opciones, es suficiente con realizar una pulsación en el codificador incremental KY – 040. A continuación, se muestra una lista de opciones, entre las cuales se encuentra el establecimiento del estado a marcha o paro (seleccionado con un asterisco "*"), el apartado de ajustes, la simulación de la carga y la opción de retroceder al menú principal. En función del modo de control que se encuentre seleccionado, la marcha se mostrará de una manera u otra. En la figura 55, se visualizan los tres tipos de marcha existentes (correspondientes a cada modo de control).

En el modo Simulink, las variables se organizan formando una cadena serial de bytes denominada "trama". Esta trama tiene una longitud fija. Para establecer una comunicación con el ordenador, el microcontrolador envía tramas con un carácter ASCII como cabecera (en este caso, "A" mayúscula, correspondiente al carácter 65). Como respuesta, Simulink envía otra trama de datos con una cabecera diferente ("B" mayúscula, o número 66 en carácter). Se aprovecha la cabecera que envía Simulink al microcontrolador para determinar el tiempo de conexión de la maqueta con el ordenador. De esta manera, si ocurre algún problema con la simulación y esta deja de enviar tramas, la maqueta se desconecta de forma automática, realizando el paro del sistema y mostrando el mensaje de error en la simulación. Las cabeceras también ayudan a distinguir a qué destinatario se dirige cada trama.

En el modo de identificación, al realizar la marcha, la pantalla muestra el tipo de entrada que se está empleando, la velocidad del eje en revoluciones por minuto, la tensión aplicada y la corriente consumida.

Por último, en el modo local, al ser un lazo de control PID interno, no se muestran todas las variables por pantalla, sino que, en su lugar, aparece el estado del sistema (marcha/paro), la velocidad del motor en revoluciones por minuto, el tipo de variable a controlar (velocidad) y el tipo de consigna (escalón). Existe otra opción en este menú denominada "Simular carga". Se trata de la aplicación de un torque externo al eje del motor de forma simulada para no dañar el motor. Esta aplicación de carga dura aproximadamente 5 segundos. Con esta opción, es posible analizar la respuesta que ofrece el lazo de control para compensar la carga.

El modo de control, la variable a controlar y las entradas se pueden modificar en el menú de ajustes. Basta con presionar sobre la sección deseada para ir alternando entre las distintas opciones.

Finalmente, hay un menú de parámetros donde se establecen los valores PID necesarios para el lazo de control interno en el ESP32, así como el valor de velocidad deseado (Setpoint) y el periodo de esta señal. El periodo también rige la velocidad de cambio en las señales de entrada. Al ser valores analógicos, es necesario presentar el cursor sobre el parámetro deseado, presionar la perilla hacia adentro y mover el codificador en sentido horario/antihorario para modificar los valores. Una vez obtenido el valor deseado por el usuario, se debe presionar nuevamente la perilla.

5.2. MODOS DE CONTROL

El firmware desarrollado permite que la maqueta funcione con distintos modos de operación, orientados a las prácticas de la asignatura Fundamentos de Automática y al aprendizaje progresivo de los conceptos de identificación y control. Como se ha descrito previamente, la selección de estos modos se realiza con la interfaz HMI, que facilita al usuario a navegar por un menú de opciones, donde puede:

- Realizar ensayos de identificación experimental del motor DC sin la necesidad de intervención de ningún control automático.

- Ejecutar un control en lazo cerrado interno al microcontrolador mediante un algoritmo PID parametrizable.

- Transferir la lógica de control a un entorno de simulación avanzado como es MATLAB/Simulink, permitiendo el diseño y validación de controladores externos en tiempo real.

Cada modo define cómo se generan las consignas de control, qué variables se utilizan internamente y cómo se realiza la comunicación con el exterior.

5.2.1. MODO IDENTIFICACIÓN

Este modo está pensado para que el usuario pueda caracterizar dinámicamente el motor y recopilar datos que permitan identificar su comportamiento. El ESP32 se encarga exclusivamente de aplicar las consignas predefinidas de excitación (entrada de tensión, adecuación a la señal PWM) y de registrar las variables de respuesta (velocidad y corriente). A su vez, imprime las variables en un formato legible por el puerto serial del software Arduino IDE, de manera que, al inicializar el graficador, las variables se muestran correctamente.

- Generación de la consigna: En modo identificación, la consigna de entrada aplicada al motor corresponde a la tensión almacenada en la variable "sys.tension". Esta tensión puede definirse como un perfil de prueba (escalón, rampa, secuencia periódica) programado en el firmware del microcontrolador a través del HMI.

- Transmisión de datos: Para que el usuario pueda visualizar y registrar las señales en tiempo real, el microcontrolador envía periódicamente los valores medidos al puerto serie. La transmisión se realiza en formato texto, separando los campos con punto y coma ";". Este formato es compatible con el Serial Plotter del entorno Arduino IDE, de forma que las variables aparecen representadas de manera gráfica y sincronizada.

- Variables transmitidas: Normalmente se envían la consigna de tensión aplicada, la velocidad del motor y la corriente consumida. De este modo, el usuario puede observar la respuesta del sistema frente a la excitación definida.

En particular, el modo de identificación permite generar los datos experimentales que luego se emplean para ajustar un modelo matemático del motor, por ejemplo, mediante la identificación de la constante de tiempo, la ganancia y el retardo (todo ello en caso de no emplear el sotfware de Simulink para la identificación). Durante este modo, los pilotos LED permanecen apagados. Ver figura 56.

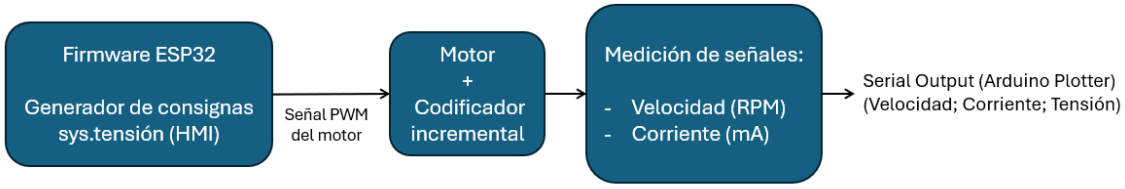


Figura 56: Diagrama de bloques modo Identificación. Elaboración propia.

5.2.2. MODO PID LOCAL

La maqueta pasa a operar en lazo cerrado, aplicando un control automático de la velocidad directamente en el ESP32 sin depender de un ordenador externo. El indicador LED de la maqueta se enciende en rojo, indicando que se está realizando un lazo de control de forma local (sin ordenador).

- Generación de la consigna: La referencia de velocidad que el motor debe seguir se obtiene de la variable “setpoint”, que el usuario define a través del menú de la interfaz HMI (pantalla LCD y codificador rotativo).

- Cálculo del control: El microcontrolador calcula continuamente el error entre el “setpoint” y la velocidad medida. A partir de este error, el firmware ejecuta el algoritmo PID configurado con los valores de “Kp”, “Ki” y “Kd” (y los valores PID establecidos para la zona muerta del motor) definidos por el usuario.

- Transmisión de datos: Igual que en el modo “Identificación”, las variables principales (consigna, velocidad real y corriente) se envían en formato texto con campos separados por punto y coma, facilitando su visualización con el Serial Plotter de Arduino IDE.

- Uso didáctico: Este modo permite que el alumno observe directamente cómo varía la respuesta del motor al modificar en tiempo real los parámetros PID, visualizando el efecto sobre la estabilidad, el tiempo de establecimiento, la recuperación del “setpoint” al aplicar una carga externa o los valores de integración acumulados (windup). Ver figura 57.

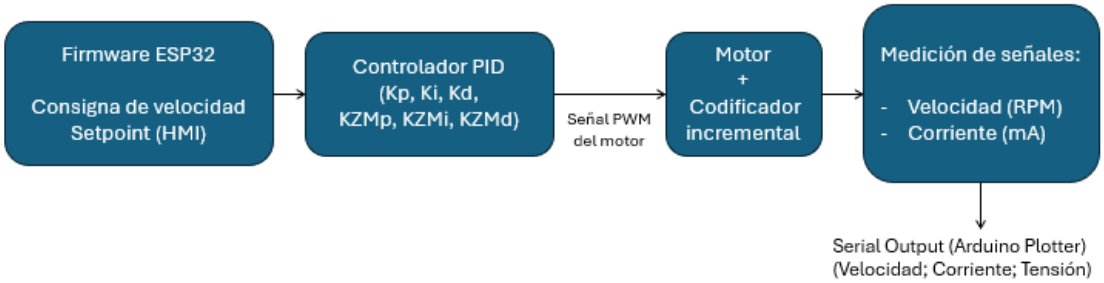


Figura 57: Diagrama de bloques para el modo PID local. Elaboración propia.

5.2.3. MODO SIMULINK

Este es el modo más avanzado, pensado para el uso conjunto de la maqueta con MATLAB/Simulink. Aquí, el microcontrolador se comporta como un esclavo de comunicación que permite integrar la maqueta en un entorno de simulación en tiempo real. El indicador LED se enciende en azul.

- Selección de la entrada: La tensión de consigna puede proceder de dos fuentes distintas: La variable sys.tension definida en el firmware (igual que en modo Identificación), o los valores de tensión generados en tiempo real por Simulink y enviados por el ordenador. La elección de una u otra fuente se controla mediante la variable booleana “elegirentradas”. Según su valor, el microcontrolador decide si aplicar su propia secuencia de tensión o seguir la consigna externa recibida.

- Protocolo de comunicación: La transmisión y recepción de datos entre el ESP32 y Simulink se hace mediante dos tramas diferenciadas: Trama con cabecera A

(enviada por el ESP32 hacia el ordenador), y trama con cabecera B (enviada desde el ordenador al ESP32.Cada trama incluye varias variables empaquetadas en formato binario.

- Variables transmitidas y recibidas: El microcontrolador envía un total de 7 datos, que incluyen la velocidad real, la corriente, la tensión efectivamente aplicada, el tiempo actual y otras variables de estado necesarias para la monitorización. Por otra parte, el ordenador (Simulink) envía 3 valores que contienen principalmente la consigna de tensión y posibles parámetros auxiliares.

- Funciones de empaquetado y decodificación: Para gestionar estos datos, el firmware utiliza las funciones unioin() y getfloat(), que permiten convertir los valores flotantes en secuencias de bytes y reconstruirlos al otro extremo de la comunicación.

- Estructura en Simulink: El bloque desarrollado en Simulink permite al usuario definir la señal de tensión a aplicar al motor. La salida del bloque devuelve cuatro señales principales: La velocidad real medida en la maqueta, la corriente consumida por el motor, la tensión final que el microcontrolador decide aplicar (en función del modo) y el tiempo real, que facilita el registro de las series temporales.

Este modo es necesario para comprobar la validez del modelo teórico obtenido en la fase de identificación y comparar simulaciones con la respuesta real del sistema. Estará establecido por defecto al inicializar la maqueta.

Tabla 12: Número de bits por cada formato compatible con MATLAB.

Parámetro	Valor
Número de bits de char	8
Número de bits de short	16
Número de bits de int	32
Número de bits de long	32
Número de bits de long long	64
Número de bits de float	32
Número de bits de double	64
Número de bits de native	32
Número de bits de pointer	32
Número de bits de size_t	32
Número de bits de ptrdiff_t	32
Mayor tamaño atómico (entero)	Long
Mayor tamaño atómico (coma flotante)	Double
Orden de bytes	Little Endian

La tabla anterior muestra la configuración de tipos de datos y representación interna utilizada en MATLAB. En esta configuración, los tipos enteros (int long) ocupan 32 bits, mientras que los enteros largos (long long) se representan en 64 bits. Los tipos de punto flotante siguen el estándar IEEE-754, con 32 bits para float y 64 bits para double. El sistema emplea un orden de bytes “Little Endian”, donde el byte menos significativo se almacena en la dirección de memoria más baja. Esto es importante al interpretar datos binarios y al intercambiar información con otros sistemas. La división de números enteros con signo se configura para redondeo hacia cero, truncando el resultado en lugar de aproximar al entero inferior. Por otro lado, el tamaño máximo definido para operaciones con enteros es de tipo “long” y, en caso de números con coma flotante, de tipo “double”. Esta configuración determina cómo se gestionan las operaciones aritméticas, la alineación de memoria y el comportamiento de los algoritmos implementados en el microcontrolador. Por ello, es necesario seguir la misma estructura para poder realizar una comunicación satisfactoria con MATLAB/Simulink. El código definido para la comunicación serial en el modo Simulink de la maqueta es el siguiente:

```
7   system_t sys;   //Estructura definida en Globales.h que contiene los variables del p
8
9   typedef union //Unión para conversión de float a byte
10  {
11    float number;
12    uint8_t bytes[4];
13  } FLOATUNION_t;
14
15  //Variables a recibir de simulink
16  FLOATUNION_t ValorPIDSimulink;
17  FLOATUNION_t ReferenciaTension;
18  FLOATUNION_t ElegirEntradas; //Elegir las entradas de simulink o las de la maqueta
19
20  //Variables enviadas por la ESP32
21  FLOATUNION_t SIM_Tiempo;
22  FLOATUNION_t SIM_TipoControl;
23  FLOATUNION_t SIM_TipoPID;
24  FLOATUNION_t SIM_Encoder;
25  //FLOATUNION_t SIM_PIDSalida;
26  FLOATUNION_t SIM_Corriente;
27  FLOATUNION_t SIM_Tension;
28  FLOATUNION_t SIM_TorqueCarga;
```

Figura 58: Declaración de la unión para convertir los valores de las variables de tipo float a formato en bytes. Elaboración propia.

Como se observa en la Figura 58, existe una unión para dar diferente formato a una variable. Cada variable declarada con la unión tiene la capacidad de ser expresada en forma numérica tipo "*float*", o de forma binaria en "*bytes*". Esta última será muy útil para construir las tramas de datos que serán enviadas por el puerto serial. Se trata de un total de 10 variables, de las cuales 7 de ellas son enviadas desde el microcontrolador, mientras que el resto las transmite Simulink. Para simplificar la comunicación, se ha decidido utilizar formatos de tipo *float* e *int* en las variables, debido a que emplean el mismo espacio (32 bits). Cada variable se fragmenta en 4 paquetes de 8 bits, o lo que es igual, un total de 4 paquetes de 1 byte por variable. Las tramas entonces se organizan de la siguiente manera: 1 byte para la cabecera (carácter ASCII) y un número de bytes consecuentes a la cantidad de datos. En el caso de las tramas enviadas por el microcontrolador, son tramas de 29 bytes, donde se emplea 4 bytes por variable (7 variables) y un byte de cabecera.

Tabla 13: trama de datos correspondiente al microcontrolador ESP32.

A (1byte)	Dato 1 (4 bytes)	Dato 2 (4 bytes)	Dato 3 (4 bytes)	Dato 4 (4 bytes)	Dato 5 (4 bytes)	Dato 6 (4 bytes)	Dato 7 (4 bytes)

Por otro lado, las tramas enviadas por Simulink, emplean un byte de cabecera y 12 bytes de datos (3 variables de 4 bytes), haciendo un total de 13 bytes.

Tabla 14: trama de datos correspondiente a Simulink.

B (1byte)	Dato 8 (4 bytes)	Dato 9 (4 bytes)	Dato 10 (4 bytes)

```
bool getFloats()
{
  // Esperar a que haya al menos 13 bytes disponibles
  if (Serial.available() >= 13)
  {
    if (Serial.read() != 'B')//Comprobar que se están recibiendo datos de Simulink
    {
      return false;
    }
    else
    {
      tanterior = tactual;
    }
    // Leer 4 bytes del primer float
    for (int i = 0; i < 4; i++)
    {
      ReferenciaTension.bytes[i] = Serial.read();
    }
    // Leer 4 bytes del segundo float
    for (int i = 0; i < 4; i++)
    {
      ValorPIDSimulink.bytes[i] = Serial.read();
    }
    // Leer 4 bytes del tercer float
    for (int i = 0; i < 4; i++)
    {
      ElegirEntradas.bytes[i] = Serial.read();
    }
    return true;
  }
  return false;  //No había suficientes datos
}
```

Figura 59: Función para la lectura de datos desde Simulink. Elaboración propia.

Tal y como se muestra en la figura 59, la función “getFloats()” es booleana, y se emplea para recibir los datos de Simulink al microcontrolador, además de garantizar el tiempo de conexión con la maqueta (true/false). Esta función permanece a la espera de recibir un paquete cuyo tamaño sea superior o igual a 13 bytes. Si este caso se da, identifica el primer byte correspondiente a la cabecera. En caso de coincidir con el carácter ASCII 66 (letra “B”), no devuelve condición, garantizando que hay una conexión

con el software de simulación. Posteriormente, realiza de forma serial la lectura de todas las variables en paquetes de 4 bytes, finalizando con la condición “true”. Si el microcontrolador deja de recibir el carácter ASCII de la cabecera, o el tamaño de la trama es menor a 13 bytes, la maqueta se desconecta del software a partir de 2 segundos, mostrando un mensaje de error en la pantalla y solicitando al usuario reiniciar la simulación (ver figura 60).

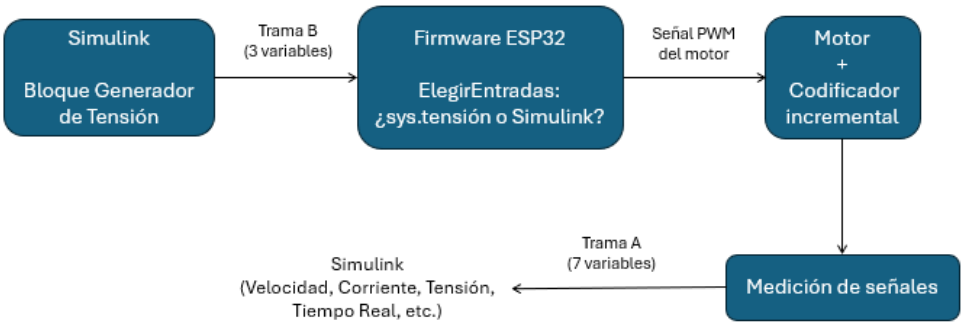


Figura 60: Diagrama de bloques para modo Simulink. Elaboración propia.

El diagrama de bloques representativo con los tres modos de control para la maqueta es el siguiente (ver figura 61):

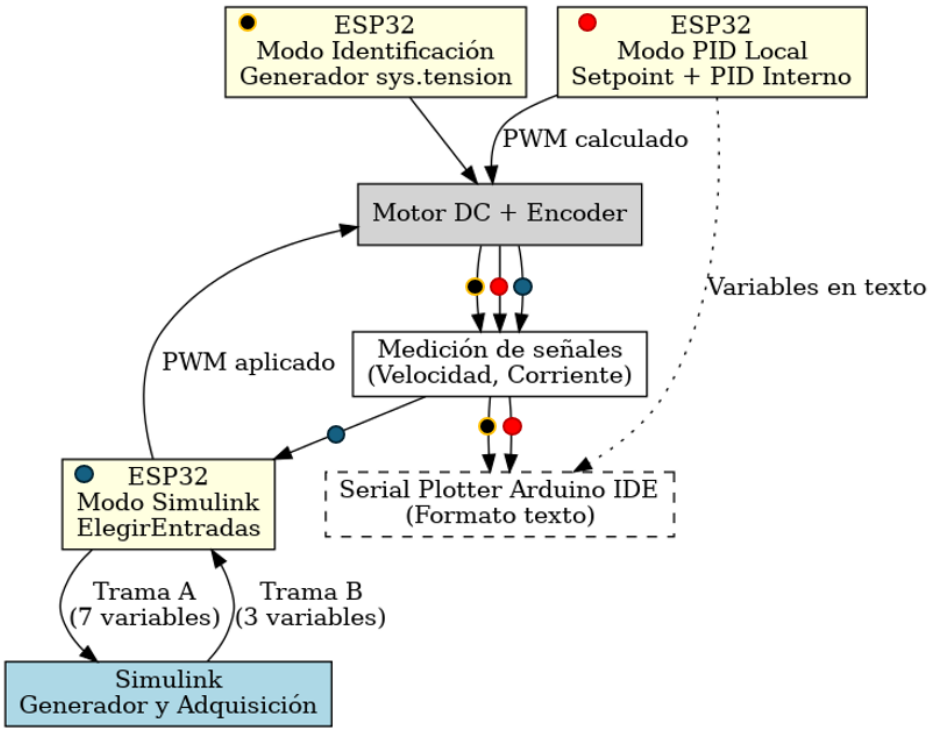


Figura 61: Diagrama de los modos de control de la maqueta. Elaboración propia.

Por razones de extensión, no se detallarán en este documento todos los aspectos del código fuente. No obstante, cada función implementada está debidamente comentada en el propio sketch de Arduino, donde se especifica su propósito y funcionamiento. En este trabajo se describen únicamente las funciones principales que estructuran la lógica de la maqueta.

CAPÍTULO 6:

DESARROLLO DEL MODELO EN MATLAB/SIMULINK

6. DESARROLLO DEL MODELO EN MATLAB/SIMULINK

En este capítulo se describe el proceso de creación y validación del modelo de comportamiento dinámico del motor de corriente continua mediante el entorno de simulación MATLAB/Simulink. El objetivo principal es disponer de un modelo ajustado a la respuesta real del sistema, que permita comparar la simulación con el comportamiento de la maqueta y diseñar estrategias de control apropiadas (ver figura 62).

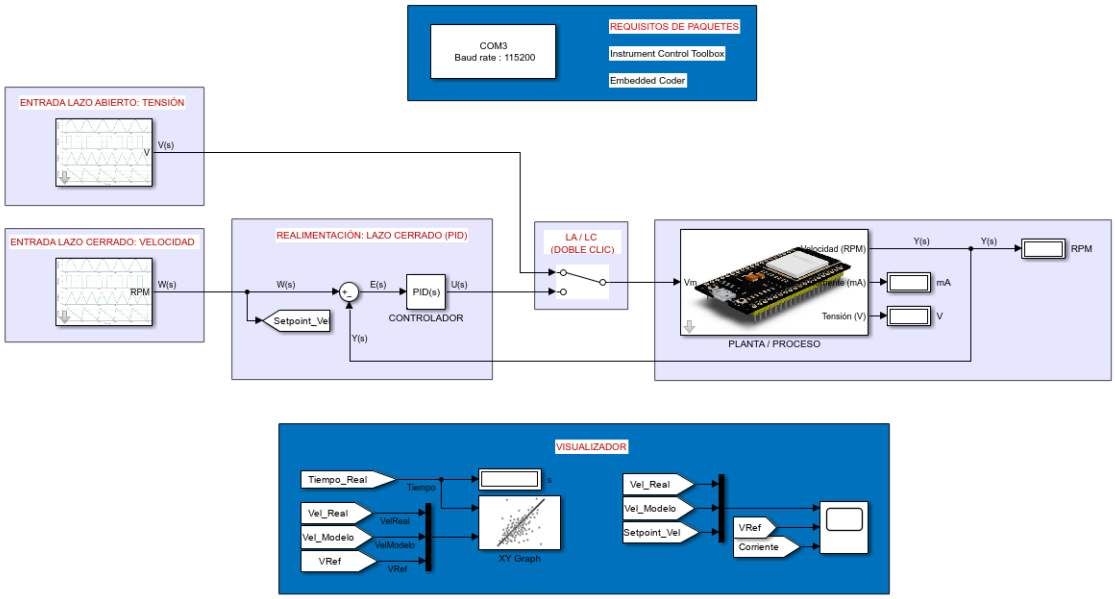


Figura 62: Archivo interactivo MATLAB/Simulink. Elaboración propia.

6.1. MODELADO MATEMÁTICO DEL MOTOR

El comportamiento dinámico del motor de corriente continua puede describirse mediante un sistema de ecuaciones diferenciales que relaciona la tensión de entrada con la corriente eléctrica y la velocidad angular. Estas ecuaciones son las siguientes:

- Ecuación eléctrica:

$$V(t) = L\frac{di(t)}{dt} + R \cdot i(t) + K_e \cdot \omega(t)$$

Donde V(t) es la tensión aplicada, L es la inductancia del devanado, R es la resistencia del devanado, Ke es la constante de fuerza contraelectromotriz, i(t) es la corriente instantánea y w(t) es la velocidad angular del rotor.

- Ecuación mecánica:

$$J\frac{d\omega(t)}{dt} + B \cdot \omega(t) = K_t \cdot i(t) - T_c$$

Donde J es la inercia total del sistema (motor y reductora, debido a que se encuentra integrada), B es el coeficiente de fricción viscosa, K_t es la constante de par y Tc es el par de carga (en este caso despreciado en vacío).

Los valores iniciales aproximados empleados en el modelo fueron extraídos de las hojas de datos y afinados posteriormente mediante la identificación experimental:

Tabla 15: Parámetros de partida motor JGA25-370.

Parámetro	Símbolo	Valor aproximado
Resistencia del devanado	R	2.4 Ω
Inductancia del devanado	L	1.8 mH
Constante de par	K_t	0.017 N·m/A
Constante de CEM	K_e	0.017 V·s/rad
Inercia equivalente	J	4.14×10^{-5} kg·m²
Coeficiente de fricción viscosa	B	1.5×10^{-5} N·m·s/rad

Estos parámetros constituyen un punto de partida que posteriormente se ajustó con los datos reales obtenidos en la maqueta.

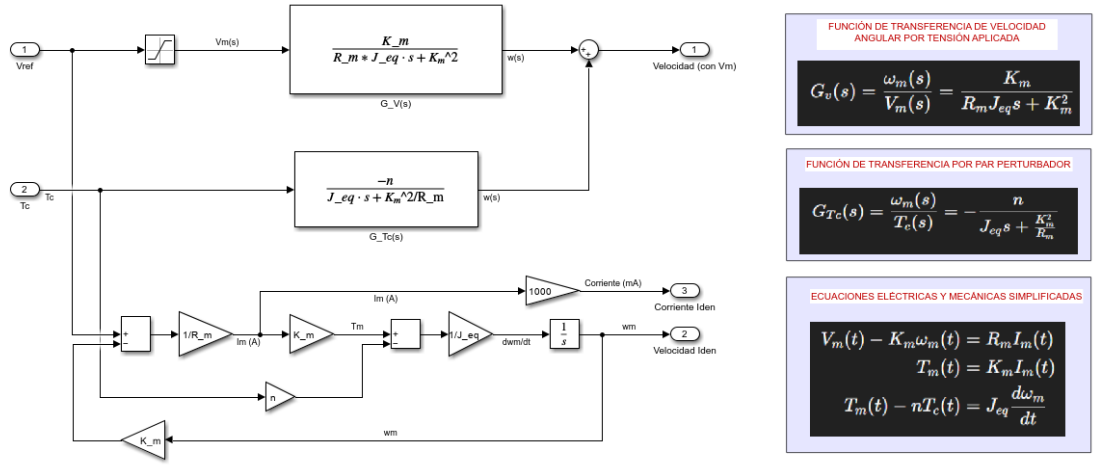


Figura 63: Modelo realizado en Simulink. En la parte superior, se emplean funciones de transferencia, mientras que en la parte inferior se ha realizado el diseño con el modelo no linealizado del sistema. Elaboración propia.

Como se observa en la figura 63, el modelo se implementó en Simulink mediante bloques básicos (integradores, sumadores, ganancias y funciones de transferencia), de manera que la entrada es la tensión aplicada y la salida la velocidad angular estimada y la corriente consumida.

6.2. CONFIGURACIÓN DE LA INTERFAZ HARDWARE IN THE LOOP (HIL)

La conexión entre la maqueta y Simulink fue uno de los aspectos más críticos del proyecto. Inicialmente se planteó utilizar el soporte oficial de placas Arduino en Simulink, que incluye bloques específicos para adquisición y envío de señales. Sin embargo, la placa ESP32 DevKit v1 no es compatible con el soporte hardware de Arduino en Simulink, ya que no dispone de integración directa ni bloques preconfigurados que permitan la comunicación y generación automática de código con esta familia de microcontroladores(el software de Simulink no reconocía el microcontrolador).

Debido a esta limitación, fue necesario diseñar una solución alternativa basada en la transmisión mediante el puerto serie, implementando un protocolo propio de intercambio de datos en paquetes de bytes. Este enfoque implicó dedicar un tiempo considerable a:

- La definición de la estructura de las tramas (cabeceras, número de variables, separación de campos).

- La codificación de variables flotantes en secuencias de bytes mediante funciones unioin(), que permiten expresar las variables en formato float o en bytes.

- La reconstrucción de datos recibidos en Simulink mediante bloques de Serial Receive, combinados con scripts de procesado en MATLAB.

En la comunicación, cada ciclo de muestreo, el microcontrolador empaqueta una trama con la cabecera “A” y 7 variables: El tiempo real del sistema (millis()), la variable a la que se realiza el control (velocidad o posición), el modo de control de la maqueta (identificación, local o simulink), el valor del encoder (velocidad real en rpm), corriente consumida (en mA), la tensión aplicada (en V) y un parámetro relativo al par instantáneo del motor. Estas variables se transmiten en formato binario a través de UART, donde Simulink se prepara para la recepción de los datos. Posteriormente, genera una trama con la cabecera “B” y 3 variables: la consigna de tensión y dos parámetros auxiliares (el selector de entradas para priorizar Simulink o HMI y el valor de consigna del lazo cerrado en Simulink, en caso de ser necesario). El ESP32 descodifica esta trama y actualiza la señal de control.

Un aspecto importante fue la medición precisa del tiempo. En una simulación típica en Simulink, el tiempo transcurrido se calcula como la suma de las duraciones de cada ciclo de simulación, que no necesariamente coinciden con el tiempo real de ejecución debido a retardos ocasionados por el procesamiento del ordenador, latencia variable en la recepción de datos, o la velocidad de interpretación de los bloques de Simulink. Por este motivo, el tiempo que figura en los datos registrados por Simulink no refleja el tiempo real transcurrido en la maqueta.

Para solucionar este problema, se decidió incluir como variable adicional la marca temporal real medida con la función “millis()” en el ESP32. De este modo, en cada trama enviada se incorpora el valor del tiempo absoluto en milisegundos desde el arranque del microcontrolador. Esto permitió alinear correctamente las curvas registradas con el tiempo real y garantizar la coherencia de los datos.

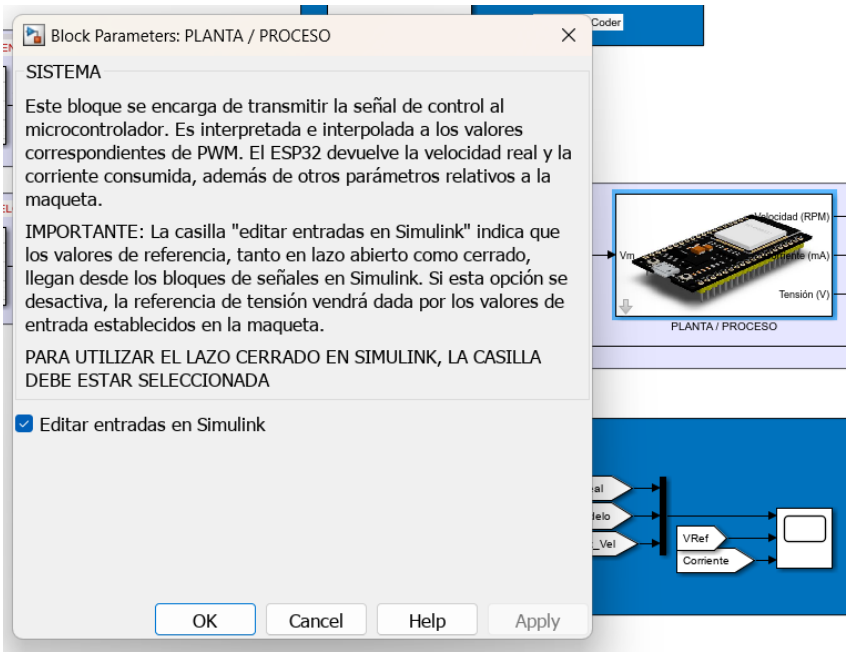


Figura 64: Casilla para elegir la preferencia de los datos de entrada en Simulink en lugar de emplear el HMI. Elaboración propia.

En aspectos de la práctica, se trata de una simulación finita, establecida por el parámetro “Stop Time” en Simulink, cuyo valor será la suma de todos los periodos de muestreo de las tramas enviadas. Es decir, si se establece un valor de tres segundos, la simulación permanecerá en ejecución hasta que el total de los tiempos de muestreo sumen tres segundos. Como ya se ha mencionado, este valor no tiene por qué coincidir estrictamente con tres segundos reales de reloj, sino que son tres segundos equivalentes a la suma de los periodos de muestreo en Simulink.

Al presionar el botón “RUN” en el entorno, se inicia la simulación y la maqueta procede a inicializar los parámetros necesarios del sistema. La duración de la simulación está determinada por el valor configurado en el parámetro “Stop Time”, que corresponde a la suma de todos los periodos de muestreo que se ejecutan de manera secuencial hasta alcanzar el tiempo total especificado (no es tiempo real).

Durante el proceso de adquisición, se genera en tiempo real una gráfica que representa los datos medidos, donde el eje X corresponde al tiempo de simulación (incrementado en cada ciclo de muestreo) y el eje Y muestra las distintas variables adquiridas, como la velocidad, la corriente y la tensión aplicada. Según el modo operativo seleccionado, el usuario puede modificar dinámicamente la tensión de consigna (en lazo abierto es posible realizar cambios desde la interfaz HMI presionando la casilla en

Simulink, mostrada en la figura 65) o la velocidad de referencia (en lazo cerrado), permitiendo así evaluar la respuesta del sistema ante cambios de consigna en tiempo real.

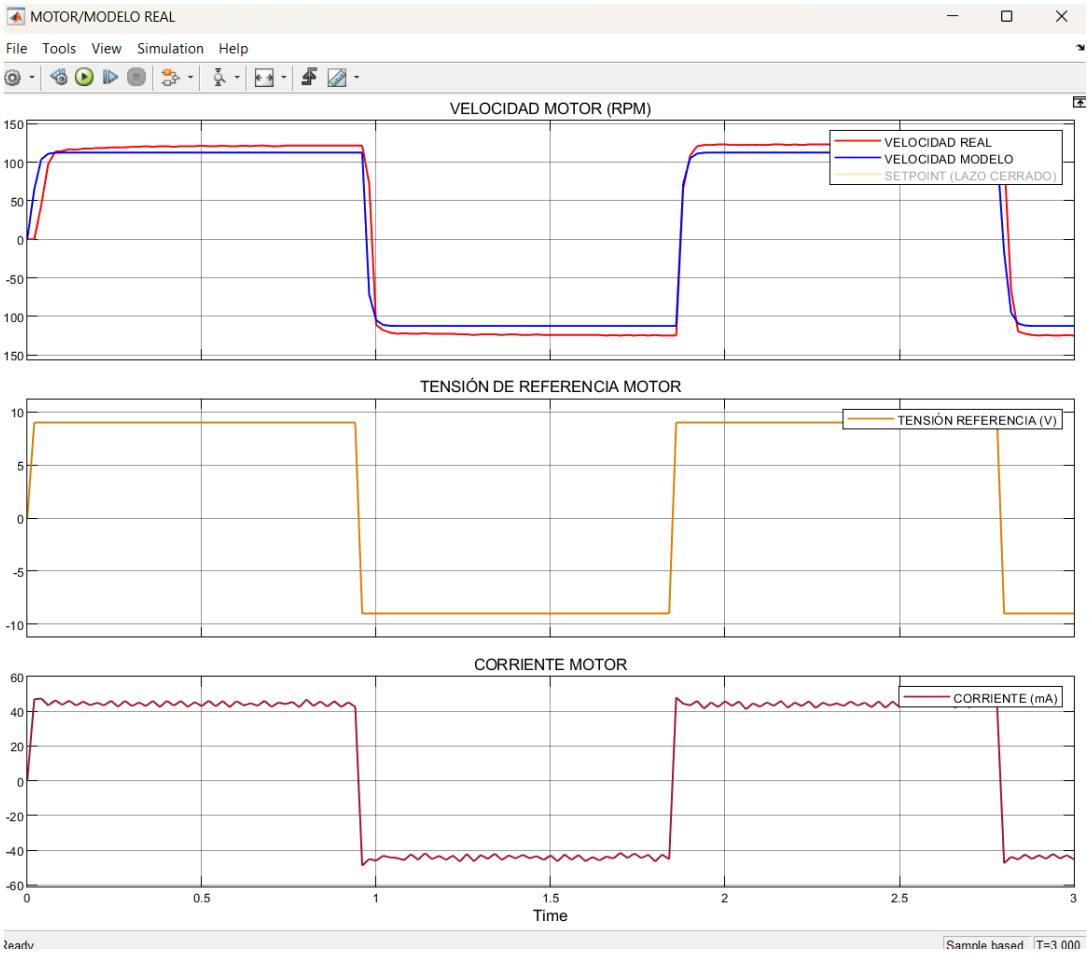


Figura 65: Visualización durante el periodo de simulación. Elaboración propia.

Para facilitar el análisis de los ensayos y la posterior elaboración de los informes de prácticas, se ha diseñado un script específico en MATLAB denominado postSimAnalysis.m. Este programa tiene como finalidad representar de manera visual y estructurada las señales capturadas durante la ejecución del modo Hardware-in-the-Loop.

El script comienza extrayendo y adaptando las dimensiones de los datos procedentes del objeto de salida (out) de la simulación. Las variables extraídas incluyen la corriente consumida por el motor, la tensión aplicada, el tiempo real proporcionado por el microcontrolador (medido mediante la función millis()), la velocidad real registrada por el encoder, la velocidad estimada por el modelo matemático y el setpoint de consigna utilizado durante el experimento.

Una vez procesados los datos, el programa genera automáticamente una ventana gráfica en la que se representan las curvas correspondientes a estas señales. La visualización se estructura en dos ejes verticales: el eje izquierdo muestra las velocidades (modelo, real y referencia), mientras que el eje derecho representa la tensión aplicada al motor. Para facilitar la interpretación, cada curva se inicializa con un color predefinido y se acompaña de una leyenda descriptiva. Además, se habilita la visualización en la cuadrícula, que contribuye a una lectura más precisa de los valores.

Junto a la representación gráfica, se ha integrado un panel de controles lateral que permite al usuario modificar en tiempo real la visibilidad de cada variable y cambiar sus colores de forma interactiva mediante menús desplegables. Esta funcionalidad es especialmente útil para destacar determinadas señales y comparar visualmente los resultados en distintas condiciones de operación.

Adicionalmente, el panel dispone de botones para realizar zoom tanto en el eje temporal como en el eje vertical, así como un apartado específico para ajustar el rango de tiempo visualizado introduciendo los valores de inicio y fin. Esta característica facilita el estudio de tramos concretos del ensayo sin necesidad de recurrir a herramientas externas.

Para complementar la fase de análisis, el script incluye opciones de exportación de los datos experimentales a un fichero Excel, permitiendo su posterior procesado o incorporación en los informes de prácticas. También es posible guardar la figura en distintos formatos gráficos (imagen o archivo .fig de MATLAB), así como abrir una tabla resumen de la corriente registrada a lo largo del tiempo.

En conjunto, este desarrollo constituye una herramienta de gran valor didáctico, pues simplifica la interpretación de los resultados y aporta un entorno de trabajo más intuitivo y accesible para los estudiantes. La integración de estas utilidades reduce la necesidad de procesar manualmente los datos y contribuye a una experiencia de aprendizaje más completa.

Una vez finalizada la simulación, se ejecuta de forma automática un script de MATLAB que procesa los datos almacenados. Este script organiza los valores de cada variable en vectores y los representa frente al tiempo real proporcionado por la función

millis() del microcontrolador, garantizando que la escala temporal refleje con precisión el tiempo transcurrido físicamente, independientemente de la velocidad de ejecución de Simulink. Adicionalmente, el script dispone de un panel de opciones que permite al usuario cambiar el color de las gráficas, realizar operaciones de zoom y seleccionar las variables que se desean visualizar.

Por último, el procesamiento incluye la generación automática de una tabla en formato Excel con los datos registrados, así como la creación de un informe específico de los valores de corriente en forma de tabla, facilitando el análisis posterior y la documentación de los ensayos realizados (ver figura 66).

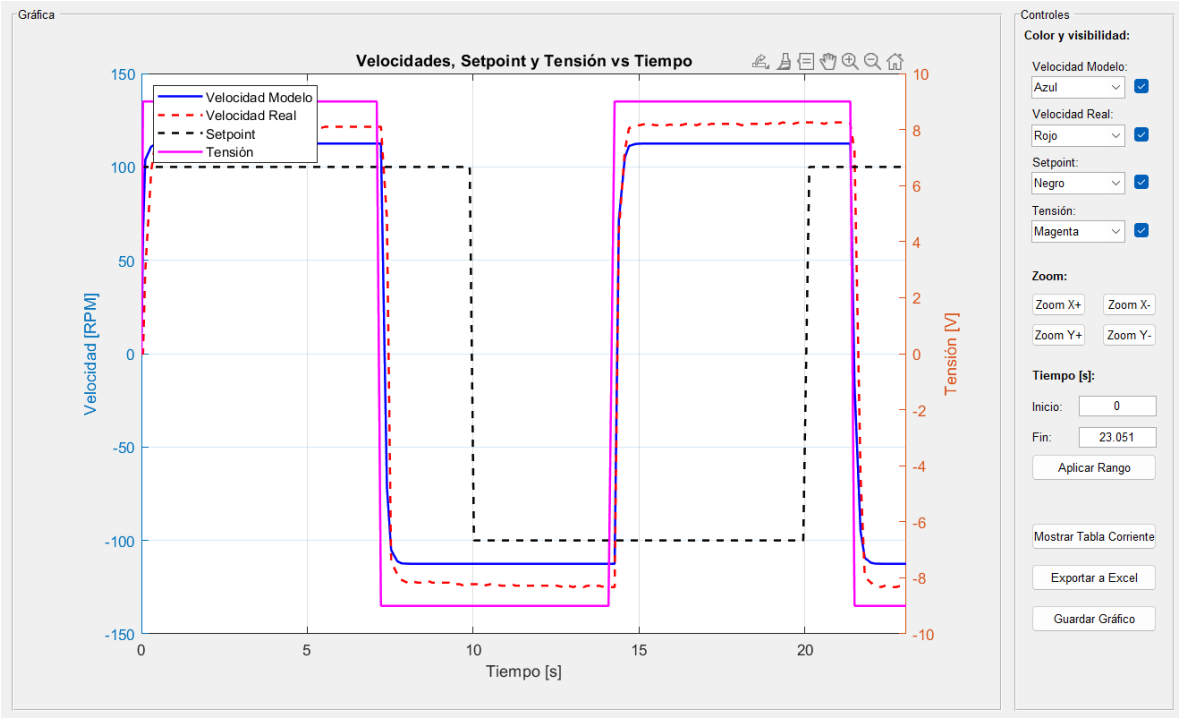


Figura 66: Ejecución del script al finalizar la simulación. Elaboración propia.

6.3. ESTRATEGIAS DE IDENTIFICACIÓN EXPERIMENTAL

Con el objetivo de obtener un modelo matemático que represente de forma rigurosa la dinámica del sistema real, se ha llevado a cabo un procedimiento de identificación experimental que permite estimar los parámetros característicos del motor de corriente continua y del conjunto mecánico. Este proceso es necesario tanto para validar la fidelidad del modelo como para diseñar posteriormente los controladores PID.

La identificación se ha realizado empleando MATLAB/Simulink (R2023b) en combinación con el microcontrolador ESP32 DevKit V1. Se definieron ensayos en los que se aplicaron escalones de tensión con amplitudes de 2 V, 4 V, 6 V, 8V, 10V, 12V, observando la respuesta temporal en velocidad y corriente. Cada escalón tuvo una duración de 6 segundos, con pausas de reposo de 6 segundos entre ellos para permitir el asentamiento de las variables.

Durante estos ensayos, el microcontrolador funcionó en modo lazo abierto, aplicando directamente las consignas generadas desde Simulink. Esta aproximación asegura que la respuesta medida dependa únicamente de la dinámica interna del sistema sin la intervención del controlador. Para reducir el ruido de medida, la señal de corriente se amplificó mediante un LM358P configurado con una ganancia de 48 y se aplicó un filtrado digital.

A partir de los datos experimentales, se estimaron los siguientes parámetros mediante ajuste por mínimos cuadrados y comparación de la respuesta simulada frente a la real:

Tabla 16: Actualización de los valores del modelo motor DC.

Parámetro	Símbolo	Valor estimado
Resistencia del devanado	R	2.42 Ω
Inductancia de armadura	L	1,3 mH
Momento de inercia equivalente	J	$3.6 \cdot 10^{-6}$ kg·m²
Coeficiente de fricción viscosa	b	$1{,}0 \cdot 10^{-4}$ N·m·s/rad
Ganancia estática tensión-velocidad	K	13.309 rpm/V

Los valores se incorporaron en el bloque matemático del modelo Simulink, consiguiendo una coincidencia de la curva simulada con la experimental con un error

cuadrático medio inferior al 4 %. Esta precisión es suficiente para los objetivos docentes y de verificación de control. Finalmente, se completa con la exportación de los datos a Excel y su representación con el script postSimAnalysis.m, facilitando al alumno la comparación sistemática de resultados y el ajuste posterior de los controladores.

6.4. IMPLEMENTACIÓN DEL CONTROL PID

Una vez identificado el modelo matemático del motor y definida la arquitectura Hardware-in-the-Loop, se procedió a implementar el controlador PID desde Simulink, que constituye el núcleo del lazo de regulación automática. Este controlador permite regular la velocidad de giro del motor en función de una consigna establecida por el usuario, asegurando una respuesta dinámica adecuada y un error estacionario mínimo.

El diseño del PID se realizó en el entorno MATLAB/Simulink mediante el bloque PID Controller, que ofrece una interfaz intuitiva para ajustar de forma independiente los tres parámetros principales: la ganancia proporcional (*Kp*), la ganancia integral (*Ki*) y la ganancia derivativa (*Kd*). De este modo, se facilita que los alumnos puedan experimentar con diferentes combinaciones de estos valores y observar en tiempo real su efecto sobre la respuesta del sistema.

La implementación del PID se diseñó para operar en dos modos principales: el modo Simulink, donde el cálculo del PID se ejecuta íntegramente en el propio modelo de Simulink. El valor de consigna, el error y las acciones de control se procesan en el ordenador, que envía al ESP32 la tensión de salida en cada ciclo de muestreo. Este enfoque permite realizar simulaciones en tiempo real y comparar de manera inmediata la respuesta del sistema real con la respuesta del modelo matemático implementado en paralelo. Por otra parte, también está el modo PID Local, donde el microcontrolador ESP32 calcula directamente la acción de control a partir de la consigna introducida mediante el potenciómetro o el codificador rotativo de la maqueta. El firmware programado en el entorno Arduino IDE incluye una rutina que actualiza la salida de control en función del error entre la referencia y la medida de velocidad. Este enfoque permite el funcionamiento autónomo sin necesidad de un ordenador conectado.

En ambos modos se estableció un tiempo de muestreo de 20 ms, lo que garantiza un compromiso adecuado entre la capacidad de procesamiento del microcontrolador, la latencia de la comunicación serie y la estabilidad del lazo de control.

Para evaluar la eficacia del controlador, se realizaron ensayos de seguimiento de escalones de velocidad y de rechazo de perturbaciones aplicadas mecánicamente sobre el eje del motor. Los resultados obtenidos muestran que el PID implementado es capaz de mantener la consigna con un error estacionario muy reducido y un tiempo de recuperación ante perturbaciones inferior a 2 segundos, validando así la correcta sintonización del sistema.

La posibilidad de alternar entre el modo local y el modo Simulink, junto con la visualización de las variables en tiempo real mediante el script desarrollado en MATLAB, proporciona al alumno una experiencia completa y didáctica, permitiéndole comprender en profundidad la influencia de cada componente del controlador y las limitaciones prácticas derivadas del retardo, la cuantificación y la no linealidad del sistema real.

6.5. ANÁLISIS DE RESULTADOS

Con objeto de evaluar la fidelidad del modelo matemático identificado y la eficacia de los controladores diseñados, se llevaron a cabo múltiples ensayos de validación en condiciones controladas. Dichos ensayos se realizaron tanto en modo lazo abierto como en modo lazo cerrado con control PID.

En primer lugar, se comparó la respuesta del modelo matemático implementado en Simulink con los datos medidos por el microcontrolador durante la aplicación de escalones de tensión. La superposición de las curvas mostró una coincidencia satisfactoria en las fases de aceleración y régimen estacionario, con un error inferior al 10 %. Esta diferencia se puede atribuir principalmente al efecto de la fricción estática y la zona muerta del motor, fenómenos que no se incluyen en la linealización del modelo pero que forman parte de la realidad física de la maqueta. A continuación, se procedió a la validación del control PID en dos modalidades:

- Modo Simulink: en este caso, el cálculo de la acción de control se realiza en tiempo real en el entorno Simulink, que transmite la consigna de tensión al ESP32. Las pruebas de seguimiento de escalones de referencia de 100 rpm y 150 rpm mostraron tiempos de establecimiento del orden de 1,5 segundos, con un sobreimpulso inferior al 12 % y ausencia de oscilaciones prolongadas. Asimismo, el error estacionario fue prácticamente nulo gracias a la acción integral del controlador.

- Modo PID Local: en esta modalidad, la ejecución del algoritmo PID se realiza directamente en el microcontrolador. Aunque el rendimiento global es similar, se observaron ligeras diferencias en la rapidez de respuesta debido al periodo de muestreo fijo y la resolución limitada de la señal de consigna introducida mediante el potenciómetro. A pesar de ello, el sistema mantuvo un error estacionario inferior al 3 % en todos los ensayos y un comportamiento estable ante perturbaciones mecánicas aplicadas manualmente sobre el eje del motor.

Durante todos los ensayos se utilizó el script postSimAnalysis.m para registrar y visualizar las variables principales en tiempo real, generando automáticamente gráficas comparativas y exportando los datos a ficheros Excel. Esta herramienta permitió identificar con precisión las diferencias entre el modelo y el comportamiento real, así como ajustar de manera iterativa los parámetros del controlador PID hasta alcanzar la respuesta deseada.

En conjunto, los resultados obtenidos demuestran que la maqueta renovada cumple los objetivos planteados en términos de robustez, precisión y valor didáctico. El modelo matemático identificado permite predecir con suficiente exactitud la dinámica del sistema, mientras que el control PID proporciona un lazo de regulación efectivo y fácilmente configurable por el alumno. La posibilidad de alternar entre modos de control, así como la integración con Simulink, aporta una gran flexibilidad para la realización de prácticas y el aprendizaje de los conceptos fundamentales de la asignatura.

CAPÍTULO 7:

CONCLUSIONES, LÍNEAS DE FUTURO Y PROPUESTAS DE MEJORA

7. CONCLUSIONES, LÍNEAS DE FUTURO Y PROPUESTAS DE MEJORA

El proyecto ha logrado con éxito la actualización integral de la maqueta didáctica de control de un motor de corriente continua con ESP32, cumpliendo con los objetivos generales y específicos planteados al inicio. Se ha desarrollado una nueva maqueta funcional que comunica bidireccionalmente con MATLAB/Simulink en tiempo real y opera de forma autónoma con varios modos de control, alcanzando así el objetivo principal propuesto. Cada objetivo específico ha sido satisfecho: se reemplazó la antigua arquitectura basada en PIC por un ESP32 de última generación, mejorando el rendimiento de cálculo y eliminando la necesidad de módulos adicionales (como el reloj externo y el convertidor RS-232 a USB de la maqueta previa). Asimismo, se diseñó y fabricó una nueva placa electrónica en KiCAD incorporando el ESP32, un puente H L298N, un encoder rotativo KY-040, pantalla LCD I2C y demás periféricos, todo con tecnología through-hole para facilitar el montaje y mantenimiento (los componentes críticos como el microcontrolador y el amplificador operacional se montaron en zócalos, permitiendo su reemplazo sencillo en caso de fallo).

A su vez, se implementó el firmware de control en el entorno Arduino IDE, integrando la lectura de los sensores (codificador incremental de posición, potenciómetro/codificador rotativo), el accionamiento del motor mediante PWM y la gestión de la interfaz HMI. Esta programación flexible permitió añadir una selección de modos de funcionamiento configurables – modo identificación, control PID local y control externo en Simulink – que cumple el objetivo de facilitar el análisis comparativo entre controladores internos y externos. También se construyeron e integraron piezas estructurales impresas en 3D para ensamblar la maqueta, obteniendo un dispositivo robusto y portátil. Además, se evaluó el comportamiento del sistema en lazo abierto y cerrado, recopilando datos de rendimiento clave; las pruebas confirmaron la estabilidad y precisión del control implementado, manteniendo por ejemplo un error estacionario inferior al 3% incluso ante perturbaciones mecánicas aplicadas al eje.

Todo el proceso de diseño electrónico, programación, simulación y validación fue documentado, incluyendo esquemas, código fuente y un manual de usuario, lo que no solo cumple con los objetivos de documentación, sino que también facilita la

reproducibilidad y futuras ampliaciones del proyecto. Cabe destacar que incluso se exploró una configuración alternativa basada en módulos comerciales (utilizando placas y componentes preensamblados) en comparación con la placa diseñada, demostrando la versatilidad y viabilidad económica de la solución propuesta en distintos escenarios de implementación. En cuanto a las mejoras respecto a la maqueta anterior, la nueva plataforma resuelve las limitaciones tecnológicas detectadas en el modelo previo: el antiguo microcontrolador dsPIC presentaba baja velocidad de procesamiento, lo que obligaba a emplear un reloj externo para cálculos de control y un conversor UART-USB externo (vía RS-232) para la comunicación con el PC. Estas complejidades han sido eliminadas gracias al ESP32, que integra de fábrica la interfaz USB-UART y opera a una frecuencia de reloj muy superior, incrementando notablemente la capacidad de cómputo y la velocidad de respuesta del sistema. También se ha optimizado la disposición física y el consumo de espacio al integrar todos los componentes en una sola placa compacta.

El resultado es una maqueta más sencilla de ensamblar y mantener, apoyada por la tecnología por agujero pasante (THT) que emplea componentes fácilmente soldables y sustituibles. Desde el punto de vista técnico, la solución actual no solo es más potente sino también más conectable: el ESP32 aporta conectividad inalámbrica (WiFi/Bluetooth) y amplias posibilidades de comunicación, sentando las bases para futuras mejoras como monitorización remota o proyectos de IoT. En definitiva, la actualización tecnológica ha hecho al sistema más limpio en hardware, más rápido en su respuesta y más preparado para adaptarse a retos adicionales.

En términos de desempeño, la maqueta renovada demostró ser robusta y precisa, cumpliendo los criterios de diseño en cuanto a estabilidad y exactitud. Los resultados experimentales indican que se han alcanzado los objetivos de robustez, precisión y valor educativo previstos. El modelo matemático identificado del motor permite predecir con fidelidad la dinámica real, y el controlador PID implementado ofrece un lazo de regulación efectivo, manteniendo el sistema con respuesta rápida y un error estacionario prácticamente nulo gracias a la acción integral. Además, el control es fácilmente ajustable por el alumno a través de parámetros accesibles, fomentando la experimentación. Desde la perspectiva didáctica, la maqueta supone un salto cualitativo: ahora los estudiantes pueden experimentar directamente con un

sistema real de control. La integración directa con herramientas como MATLAB y Simulink (ampliamente utilizadas en la formación en automática) aporta un valor pedagógico añadido, pues los alumnos se familiarizan aún más con estos entornos al interactuar con la planta física. Gracias a la comunicación bidireccional con Simulink, es posible realizar identificación de parámetros y aplicar controladores avanzados en tiempo real, comparando de forma inmediata las respuestas simuladas con las medidas en la maqueta. Esta característica refuerza la comprensión teórica mediante la práctica, permitiendo apreciar fenómenos reales que trascienden los modelos ideales. El sistema actualizado permite observar efectos prácticos como el desgaste mecánico, la zona muerta o la saturación de señales, factores que no suelen contemplarse en simulaciones puramente teóricas y que ahora pueden ser estudiados por los alumnos en un entorno seguro. Todo ello incrementa significativamente el valor didáctico del dispositivo, al puentear la brecha entre la teoría y la práctica.

La versatilidad de los modos de funcionamiento implementados es otro punto fuerte para resaltar. El diseño ofrece tres modalidades principales de uso: un modo identificación para determinar los parámetros del modelo del motor, un modo PID local donde el ESP32 ejecuta el algoritmo de control internamente, y un modo Simulink en el que la maqueta sigue las consignas de un controlador externo en MATLAB/Simulink en tiempo real. Esta capacidad de alternar entre controladores internos y externos proporciona una gran flexibilidad para las prácticas de laboratorio y proyectos académicos. Mediante la interfaz HMI (pantalla LCD y codificador rotativo) integrada en la maqueta, el usuario puede seleccionar de forma intuitiva el modo deseado y ajustar parámetros básicos, todo ello sin necesidad de recurrir directamente al código ni depender exclusivamente de un ordenador. Por ejemplo, un estudiante puede operar el sistema de forma autónoma con el controlador PID embarcado para entender su dinámica básica, y luego con unos simples pasos conmutar al modo conectado a Simulink para probar estrategias de control más avanzadas diseñadas en el aula. Importa subrayar que la maqueta puede funcionar de manera completamente autónoma, sin un ordenador, lo que facilita su uso incluso en entornos donde no se disponga de MATLAB; a la vez, ofrece la posibilidad de integrar un ordenador cuando se requiera un análisis más profundo o el desarrollo de controladores complejos. Esta dualidad hace al equipo muy versátil en contextos educativos diversos. De hecho, gracias a su independencia de una plataforma

específica, el sistema puede emplearse no solo en la asignatura original de Fundamentos de Automática sino también en áreas afines como Mecatrónica, Robótica o Automatización Industrial, ampliando su utilidad y alcance didáctico.

En resumen, los múltiples modos de operación incrementan las oportunidades de aprendizaje, adaptándose a distintos objetivos pedagógicos y niveles de complejidad. La calidad de la solución propuesta queda patente en la conjunción de su solidez técnica con su orientación formativa. Se ha obtenido una maqueta de alta calidad tanto en construcción hardware como en desempeño, que satisface plenamente las expectativas iniciales. Durante el desarrollo y las pruebas, todos los módulos funcionaron de manera integrada y estable, evidenciando una arquitectura bien diseñada. El sistema resultante es fiable, con capacidad de mantener un control preciso del motor y soportar perturbaciones sin comprometer la estabilidad. Al mismo tiempo, cada mejora introducida estuvo alineada con objetivos educativos: por ejemplo, la implementación de la pantalla y el menú interactivo no solo añade funcionalidad, sino que facilita al estudiante la observación y modificación de parámetros en tiempo real, promoviendo una comprensión más interactiva. También se incorporaron herramientas para el análisis de resultados post-ensayo, como la exportación automática de datos y generación de gráficos comparativos en Excel mediante scripts dedicados, lo que agiliza la evaluación de diferentes configuraciones de control y refuerza el aprendizaje autónomo del alumno. La solución resultante es, por tanto, un recurso docente de calidad, donde la experiencia de usuario ha sido cuidada para potenciar el proceso formativo.

Por otro lado, la elección de componentes estandarizados y la documentación correspondiente garantizan la replicabilidad del sistema; cualquier laboratorio académico podría reproducir la maqueta con costes accesibles, gracias a que se trata de un montaje económico, escalable y fácilmente duplicable. Este carácter abierto y reproducible añade valor a la propuesta, ya que puede ser adoptada o adaptada por otros centros educativos interesados en disponer de herramientas similares. Por último, hay que destacar la capacidad de ampliación y la posibilidad de mejoras futuras. Aunque el proyecto en su estado actual ya cumple su propósito educativo, su diseño prevé posibilidades de crecimiento. Por ejemplo, se ha dejado un puerto libre en la placa de control para integrar un segundo motor DC, lo que permitiría extender

la maqueta a experimentos de control múltiple de ejes o sistemas más complejos (como la simulación de un sistema ABS en automoción). Del mismo modo, las funcionalidades inalámbricas del ESP32 (WiFi/Bluetooth), no explotadas en este trabajo, abren la puerta a desarrollos posteriores en ámbitos como el IoT o el control remoto avanzado. Estas perspectivas de ampliación reafirman la versatilidad intrínseca de la plataforma y aseguran que pueda seguir evolucionando de acuerdo con nuevas necesidades o iniciativas de investigación docente. No obstante, es importante señalar que incluso sin añadidos futuros, la maqueta ya ofrece todo lo necesario para el fin con el que fue concebida: servir de introducción práctica a los lazos de control y a la automatización para estudiantes de ingeniería, en un entorno seguro y manejable. De hecho, su carácter educativo define hasta dónde complicar el sistema, priorizando siempre la claridad didáctica sobre la sofisticación innecesaria.

En conclusión, todos los objetivos planteados al inicio del proyecto han sido alcanzados con creces, dando como resultado una maqueta de control de motor DC moderna, versátil y pedagógicamente enriquecedora. La solución implementada mejora significativamente a su predecesora en rendimiento tecnológico y valor educativo, aportando un recurso fiable, intuitivo y formativo para las prácticas de automatización. Por último, el desarrollo de este Trabajo Fin de Grado ha supuesto una experiencia de aprendizaje integral para el autor, al combinar disciplinas de electrónica, programación embebida, control automático y herramientas de simulación. Este recorrido ha permitido consolidar conocimientos y adquirir nuevas competencias en ingeniería, desde la resolución de retos de diseño hasta la integración exitosa de múltiples sistemas. En definitiva, se presenta una maqueta de alta calidad técnica y didáctica, capaz de potenciar la formación de futuros ingenieros y susceptible de seguir creciendo con nuevos aportes, lo que certifica el éxito y la solidez de la solución propuesta en el contexto académico.

CAPÍTULO 8:

BIBLIOGRAFÍA

8. BIBLIOGRAFÍA

[1]: Brunete, A. (s.f.). Sistemas de control. En Introducción a la automática. Bookdown. Recuperado el 12 de julio de 2025, de https://bookdown.org/alberto_brunete/intro_automatica/sistemas-de-control.html

[2]: Makinando Vélez. (2018, 15 de febrero). Sistemas de control de lazo cerrado. Makinando Vélez. Recuperado el 12 de julio de 2025, de https://makinandovelez.wordpress.com/2018/02/15/sistemas-de-control-de-lazo-cerrado/

[3]: iStock. (s.f.). Fotos de pedal de coche [Galería]. iStock. Recuperado el 12 de julio de 2025, de https://www.istockphoto.com/es/fotos/pedal-coche?page=2

[4]: Automatismo Industrial. (s.f.). 1.3.5.2 Principios de funcionamiento. En *Curso carnet instalador baja tensión: Motores de corriente continua*. Recuperado el 12 de julio de 2025, de https://automatismoindustrial.com/curso-carnet-instalador-baja-tension/motores/1-3-5-motores-de-corriente-continua/1-3-5-2-principios-de-funcionamiento/

[5]: Pepper. (2016, abril). Modelo simple de un motor DC. Pepper Blogspot. Recuperado el 12 de julio de 2025, de https://stg-pepper.blogspot.com/2016/04/modelo-simple-de-un-motor-dc.html

[6]: Shoptronica. (s.f.). Qué es PWM y cómo funciona. Shoptronica. Recuperado el 12 de julio de 2025, de https://www.shoptronica.com/curiosidades-tutoriales-y-gadgets/4517-que-es-pwm-y-como-funciona-0689593953254.html

[7]: Wikipedia. (s.f.). Archivo: Arduino Logo.svg. Wikipedia, la enciclopedia libre. Recuperado el 12 de julio de 2025, de https://es.wikipedia.org/wiki/Archivo:Arduino_Logo.svg

[8]: FPGA Key. (s.f.). dsPIC30F4012 - High-performance digital signal controller. FPGA Key. Recuperado el 12 de julio de 2025, de https://www.fpgakey.com/wiki/details/22?srsltid=AfmBOooO8H2V68-37OApAgPueP5WrIO3zgoFYPZRR2T4q4bP3Ae47Zgc

[9]: MathWorks. (s.f.). Simulink – Simulation and Model-Based Design. MathWorks. Recuperado el 12 de julio de 2025, de https://es.mathworks.com/products/simulink.html
[10]: https://tritekbattery.com/es/uart-communication-protocol/
[11]:https://riverdi.com/es/blog/comprender-la-velocidad-en-baudios-una-guia-completa?srsltid=AfmBOopgXC4cFUHaMWb8c9Ge52Wd7ivT-HdFJ11Jd5GpZDtp8wFmfTq
[12]: FMUSER. (s.f.). ¿Qué es la modulación por ancho de pulso (PWM)?. FMUSER. Recuperado el 12 de julio de 2025, de https://es.fmuser.net/content/?11027.html
[13]: Aprendiendo Arduino. (2017, 9 de julio). I2C. Aprendiendo Arduino. Recuperado el 12 de julio de 2025, de https://aprendiendoarduino.wordpress.com/2017/07/09/i2c/
[14]: Grobotronics. (s.f.). ESP32 Development Board – DEVKIT V1. Grobotronics. Recuperado el 12 de julio de 2025, de https://grobotronics.com/esp32-development-board-devkit-v1.html?sl=en&srsltid=AfmBOoqI67YpyWz2Bkv5TFqZXWbPXAJAopAKxQli_Ds3BpHqsjOnh89-
[15]: Sando Robotics. (s.f.). Motor DC con encoder HS1017-170. Sando Robotics. Recuperado el 12 de julio de 2025, de https://sandorobotics.com.mx/producto/hs1017-170/
[16]: Arduino. (s.f.). Concepts – Arduino Education Kit. Arduino. Recuperado el 12 de julio de 2025, de https://aek.arduino.cc/chapter/concepts
[17]:Solución Ingenieril. (s.f.). Amplificador no inversor. Solución Ingenieril. Recuperado el 12 de julio de 2025, de https://solucioningenieril.com/amplificadores_operacionales/amplificador_no_inversor
[18]: A3D Prints. (s.f.). Disipador motor NEMA. A3D Prints. Recuperado el 12 de julio de 2025, de https://www.a3dprints.com/producto/disipador-motor-nema/
[19]: Wikipedia. (s.f.). Archivo: KiCad-Logo.svg. Wikipedia, la enciclopedia libre. Recuperado el 12 de julio de 2025, de https://es.wikipedia.org/wiki/Archivo:KiCad-Logo.svg

[20]: Espressif Systems. (2025). *ESP32 Series Datasheet v4.9*.

[21]: Espressif Systems. (2025). *ESP32-WROOM-32 Datasheet v3.5*.

[22]: Olimex. (2025). *ESP32-WROOM-32 technical overview*.

[23]: Grobotronics. (2025). *ESP32 Development Board – DEVKIT V1 specs*.

[24]: Einstronic. (2025). *ESP-DevKit V1 specifications*.

[25]: Random Nerd Tutorials. (2024). *Getting Started with the ESP32 DevKit V1*.
[26]: Maker Advisor. (2025). *DevKit V1 Wi-Fi and Bluetooth details*.
[27]: The Engineering Projects. (2020). *ESP32 pinout and features*.

[28]: García, P., García, C., & Martín, F. (2007). *Control de velocidad de un motor de corriente continua mediante controladores PID implementados en FPGA*. Revista Iberoamericana de Automática e Informática Industrial RIAI, 4(1), 63–71.
[29]: Banzi, M., & Shiloh, M. (2014). *Getting Started with Arduino* (3rd ed.). Maker Media, Inc.
[30]: Arduino. (s. f.). *What is Arduino?*. Recuperado de https://www.arduino.cc/en/Guide/Introduction
[31]: Arduino. (s. f.). *Arduino Documentation*. Recuperado de https://docs.arduino.cc/
[32]: Monk, S. (2017). *Programming Arduino: Getting Started with Sketches*. McGraw-Hill Education.

[33]:MathWorks. (s. f.). *What Is MATLAB?*. Recuperado de https://www.mathworks.com/products/matlab.html
[34]: CIMSS, University of Wisconsin–Madison. (s. f.). *What is MATLAB?* Recuperado de https://cimss.ssec.wisc.edu/wxwise/class/aos340/spr00/whatismatlab.htm

[35]: Tutorials Point. (s. f.). *MATLAB – Overview*. Recuperado de https://www.tutorialspoint.com/matlab/index.htm

[36]: GeeksforGeeks. (2024, marzo 1). *Introduction to MATLAB*. Recuperado de https://www.geeksforgeeks.org/introduction-to-matlab/

[37]: MathWorks. (s. f.). *What Is Simulink?*.

Recuperado de https://www.mathworks.com/products/simulink.html

[38]: MathWorks. (s. f.). *Simulink Documentation*.
Recuperado de https://www.mathworks.com/help/simulink/

[39]: MathWorks. (s. f.). *Simscape Documentation*.
Recuperado de https://www.mathworks.com/help/physmod/simscape/

[40]: Ogata, K. (2010). *Modern Control Engineering* (5th ed.). Prentice Hall.

[41]: Ogata, K. (2010). *Ingeniería de control moderna*. Pearson Educación.

[42]: Dorf, R. C., & Bishop, R. H. (2011). *Sistemas de control automático*. Addison-Wesley.

[43]: MATLAB. (2023). *System Identification Toolbox™ User's Guide*. MathWorks.

[44]: Isermann, R. (2006). *Identification of Dynamic Systems*. Springer.

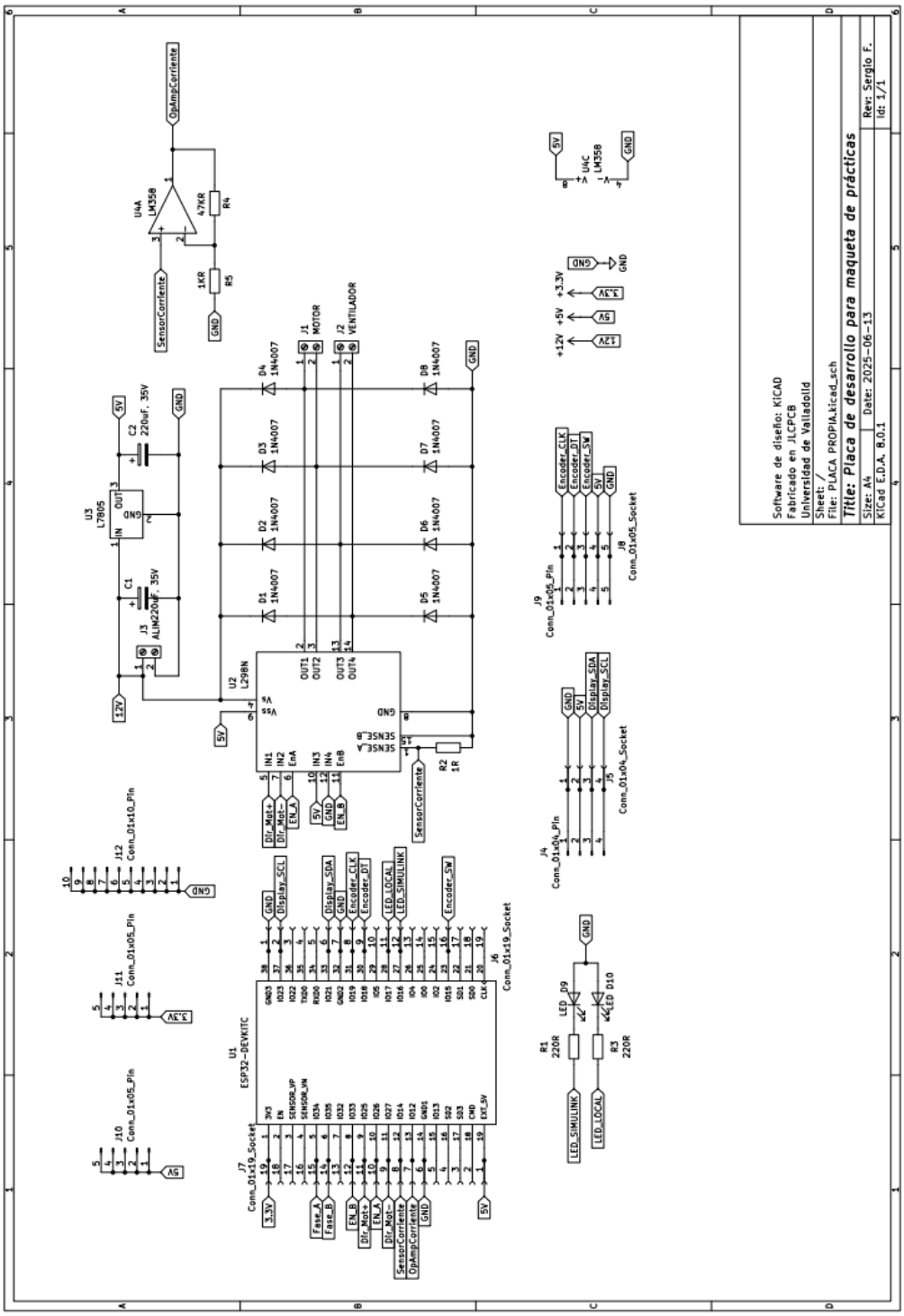
[45]: Arduino. (2020). *ESP32 DevKit V1 Technical Reference Manual*.

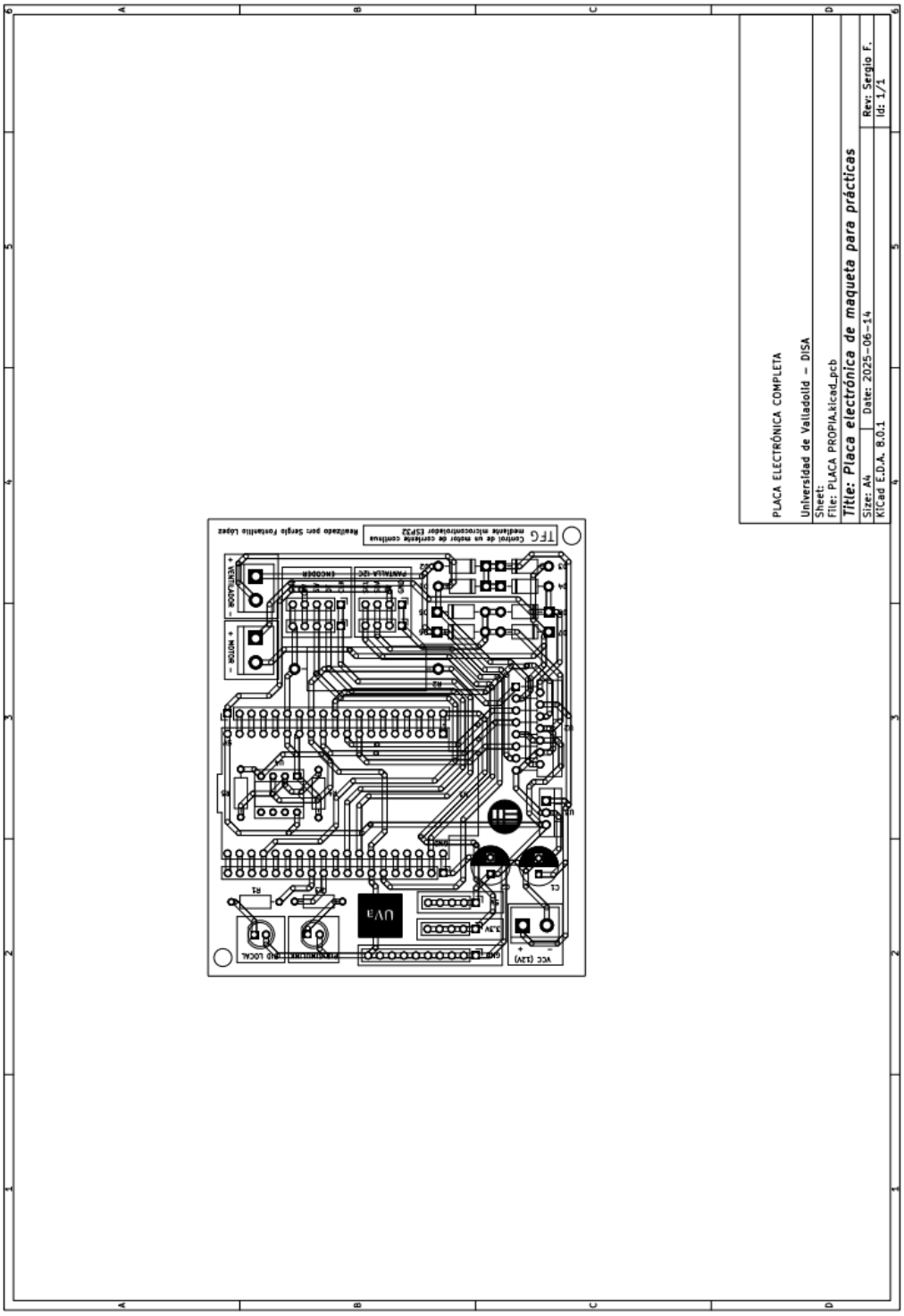
ANEXOS

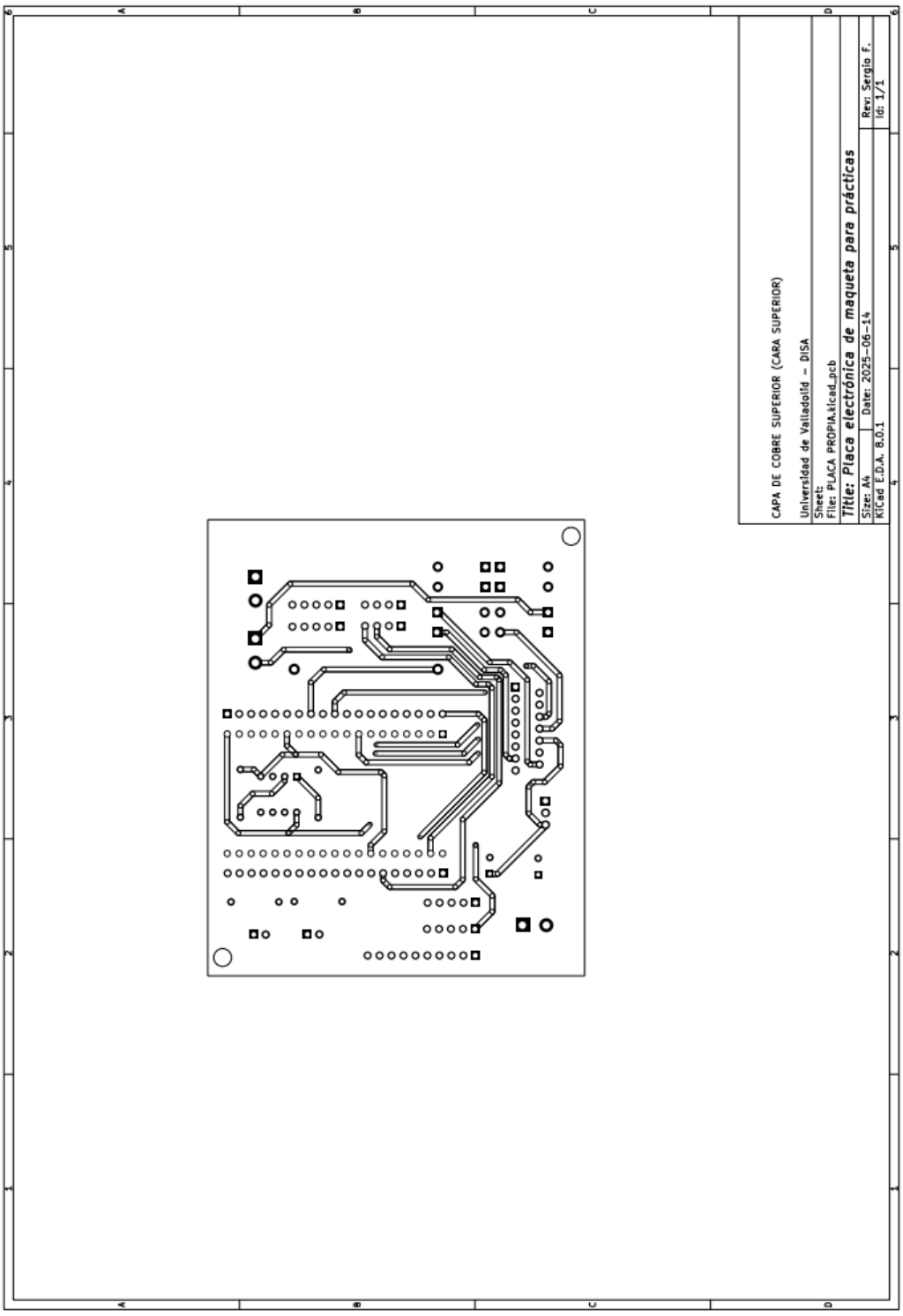
ANEXOS

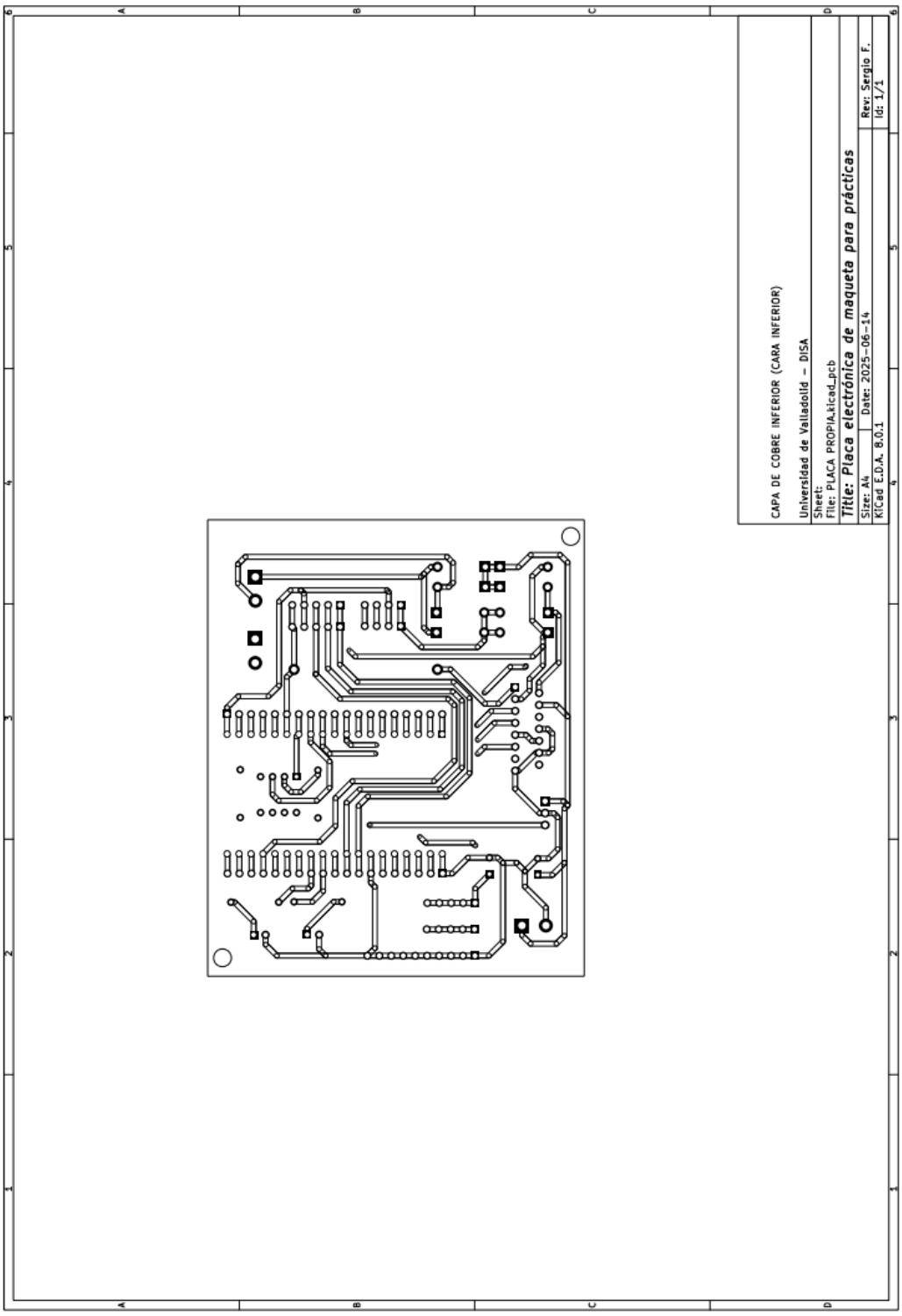
A. ESQUEMAS ELÉCTRICOS DETALLADOS

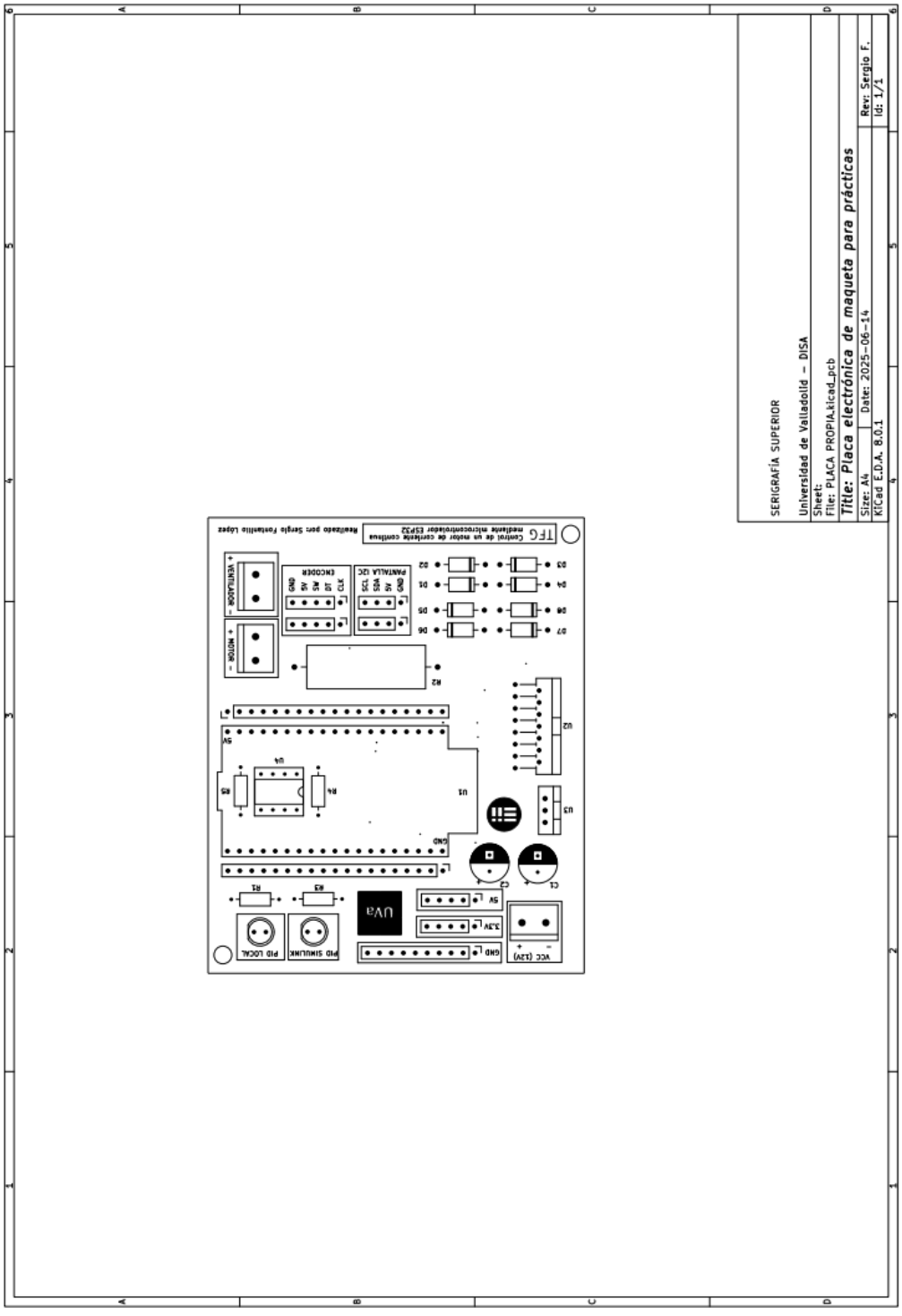
A continuación, se muestra el desglose de capas necesarias para la fabricación de la placa electrónica:

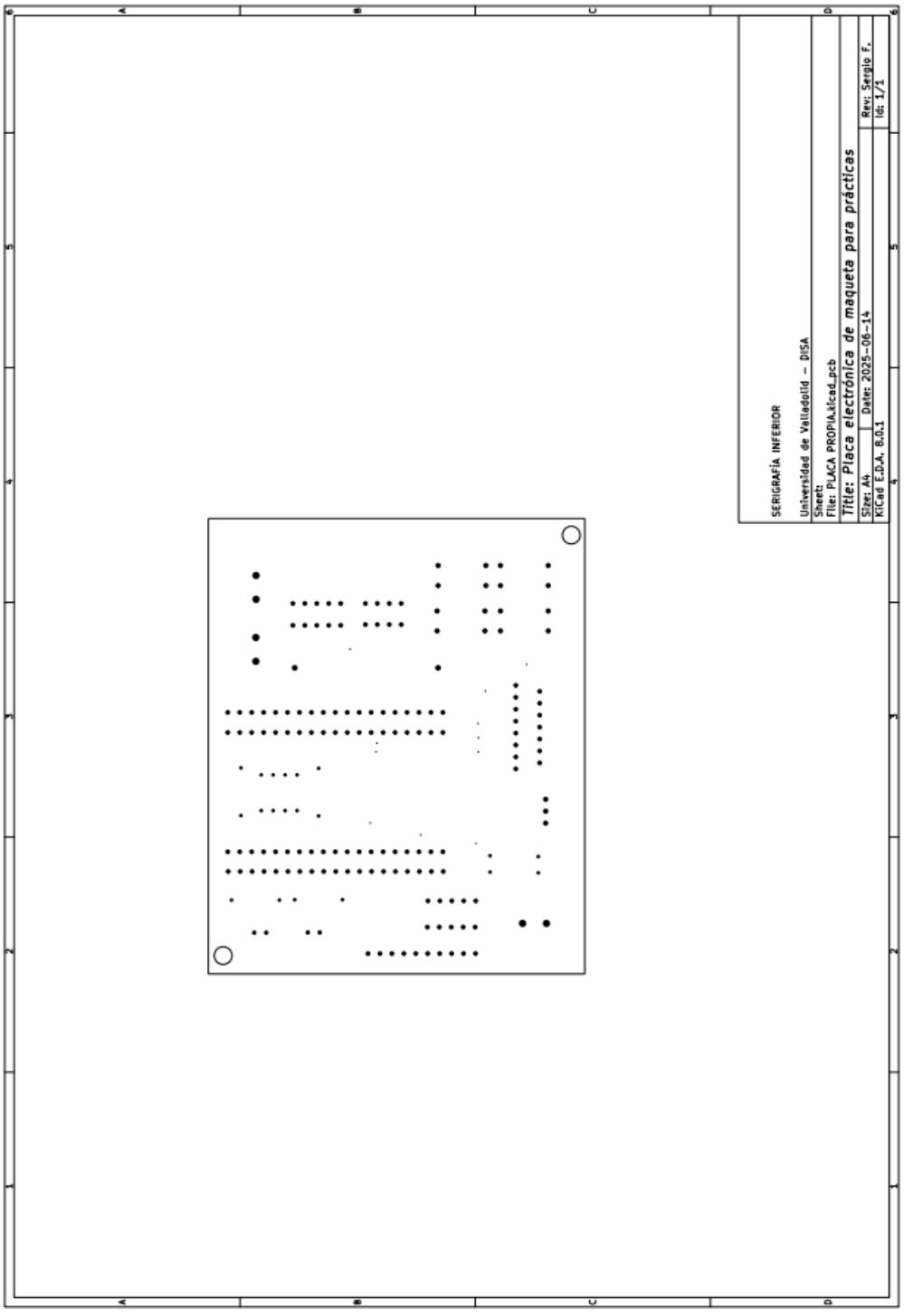



PLACA ELECTRÓNICA COMPLETA
Universidad de Valladolid – DISA
Sheet:
File: PLACA PROPIA.kicad_pcb
Title: Placa electrónica de maqueta para prácticas
Size: A4
Date: 2025-06-14
Rev: Sergio F.
KiCad E.D.A. 8.0.1
Id: 1/1
TFG
Control de un motor de corriente continua mediante microcontrolador ESP32
Realizado por Sergio Fontanillo Lopez
MOTOR
ENCODER
MATRIZ I2C
VCC (12V)
3.3V
GND
UVa
FOTOINTERRUPTOR LOCAL

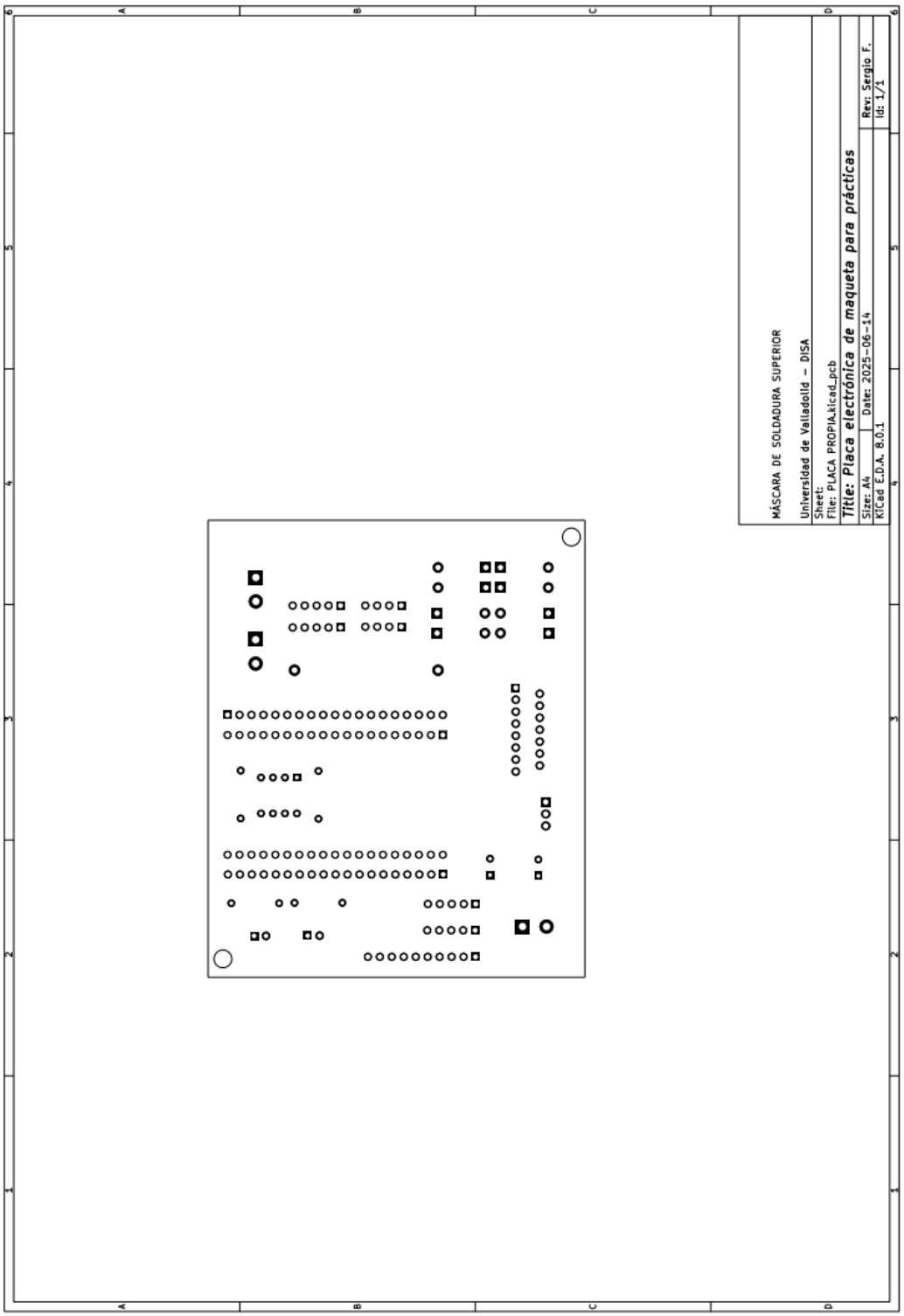

CAPA DE COBRE SUPERIOR (CARA SUPERIOR)
Universidad de Valladolid – DISA
Sheet:
File: PLACA PROPIA.kicad_pcb
Title: Placa electrónica de maqueta para prácticas
Size: A4
Date: 2025-06-14
Rev: Sergio F.
KiCad E.D.A. 8.0.1
Id: 1/1

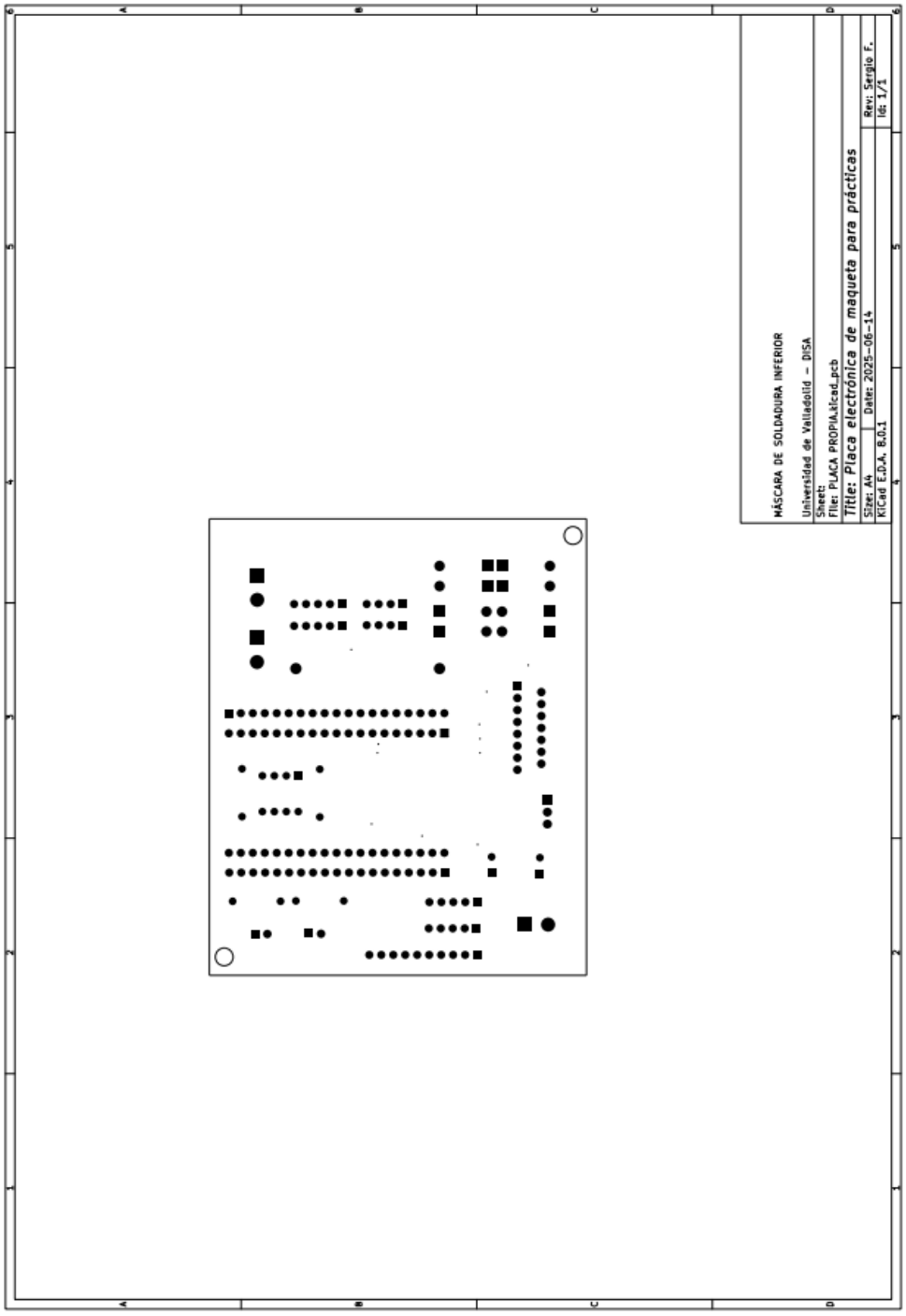

CAPA DE COBRE INFERIOR (CARA INFERIOR)
Universidad de Valladolid – DISA
Sheet:
File: PLACA PROPIA.kicad_pcb
Title: Placa electrónica de maqueta para prácticas
Size: A4
Date: 2025-06-14
Rev: Sergio F.
KiCad E.D.A. 8.0.1
Id: 1/1


SERIGRAFÍA SUPERIOR
Universidad de Valladolid – DISA
Sheet:
File: PLACA PROPIA.kicad_pcb
Title: Placa electrónica de maqueta para prácticas
Size: A4
Date: 2025-06-14
KiCad E.D.A. 8.0.1
Rev: Sergio F.
Id: 1/1
TFG Control de un motor de corriente continua mediante microcontrolador ESP32
Realizado por Sergio Fontanillo López
UVa


SERIGRAFÍA INFERIOR
Universidad de Valladolid – DISA
Sheet:
File: PLACA PROPIA.kicad_pcb
Title: Placa electrónica de maqueta para prácticas
Size: A4
Date: 2025-06-14
Rev: Sergio F.
KiCad E.D.A. 8.0.1
Id: 1/1

CONTORNO DE LA PLACA
Universidad de Valladolid – DISA
Sheet:
File: PLACA PROPIA.kicad_pcb
Title: Placa electrónica de maqueta para prácticas
Size: A4
Date: 2025-06-14
Rev: Sergio F.
KiCad E.D.A. 8.0.1
Id: 1/1


MÁSCARA DE SOLDADURA SUPERIOR
Universidad de Valladolid – DISA
Sheet:
File: PLACA PROPIA.kicad_pcb
Title: Placa electrónica de maqueta para prácticas
Size: A4
Date: 2025-06-14
Rev: Sergio F.
KiCad E.D.A. 8.0.1
Id: 1/1


MÁSCARA DE SOLDADURA INFERIOR
Universidad de Valladolid - DISA
Sheet:
File: PLACA PROPIA.kicad_pcb
Title: Placa electrónica de maqueta para prácticas
Size: A4
Date: 2025-06-14
Rev: Sergio F.
KiCad E.D.A. 8.0.1
Id: 1/1

B. LISTADOS DE MATERIALES Y COSTE

En la tabla mostrada a continuación se resumen los materiales empleados para la fabricación de la placa electrónica, así como las direcciones de enlace de compra a dichos componentes.

Tabla 17: : Componentes empleados para la construcción de la placa electrónica con los enlaces de compra respectivos.

Nº	Componente	Modelo/Valor	Fabricante/ Vendedor	Cantidad	Descripción	Sitio web
1	Resistencias	1KΩ	ELEGOO	1	OpAmp	Enlace
2		47KΩ		1	OpAmp	
3		220Ω		2	Diodos LED	
4	Resistencia shunt cerámica	1Ω 5W	Jiamingxin Technology	1	Medición de corriente	Enlace
5	Condensador electrolítico	35V 220uF	DIP – Bao Yuanqi Electronics	2	Valores de tensión estables en la maqueta	Enlace
6	Diodo LED	3V 20mA	ELEGOO	2	Indicadores de tipo PID	Enlace
7	Diodos schottky rectificadores	1N5819 MIC	Goldelewey Electronics Technology	8	Correcto funcionamiento del puente H	Enlace
8	Bornera 2 terminales	BXT - 301	Baixinte E-commerce Co.	3	Alimentación y motores	Enlace
9	Pines hembra para PCB	Set de 40 pines	Yiwu lulan Trading Co.	2	Fácil reemplazo de componentes	Enlace
10	Pines macho para PCB	Set de 40 pines		2	Accesibilidad a señales del	

					controlador y alimentación	
11	Socket DIP8 IC	8 pines DIP-8 2.54mm	Youmile	1	Fácil reemplazo del OpAmp	Enlace
12	Regulador de tensión	L7805CV 5V	TSTAR Electronics United	1	Obtener 5V estables a partir de 12V	Enlace
13	Amplificador Operacional	LM358P	ECSiNG	1	Acondicionamiento de la señal de corriente	Enlace
14	Microcontrolador ESP32	ESP32 DevKit V1	Espressif Systems	1	Lógica de la maqueta	Enlace
15	Conector MicroUSB a USB	-	-	1	Comunicación con el PC	-
16	Puente H	ZIP-15 L298N	QUANXINFA	1	Control de motor	Enlace
17	Disipador Aluminio	-	KARELLS	1	Disipación térmica regulador de tensión	Enlace
18	Codificador incremental	KY-040 (a ser posible el modelo roscado)	QWORK	1	Navegación por el menú de la maqueta	Enlace
19	Pantalla LCD	LCD 16x2 i2c (1602i2c)	Freenove	1	Visualización del menú	Enlace
20	Tornillos M3*16	Métrica 3	Romon	6		Enlace
21	Tornillos M3*20			4		

22	Tuercas M3			6	Soporte de los elementos en la maqueta	
23	Arandelas M3			6		
24	Separadores placa base	Métrica 3	Darkyoun	2	Elevan la placa de la base	Enlace
25	Fabricación / Producción de la placa electrónica	KiCAD	JLCPCB	1	Cualquier empresa es válida (PCBWay, Españolas…)	Enlace
ALTERNATIVA A LA PLACA (USO DE MÓDULOS)						
26	Módulo controlador para motor DC	L298N	SZYTF	1	Módulo con ZIP15 L298 integrado y regulador de tensión a 5V	Enlace
27	Zócalo ESP32 DevKit V1	-	DORHEA	1	Extensión de los pines de ESP32 a borneras	Enlace
28	Módulo medidor de corriente	ACS712	Logic ICS	1	Medición de corriente por efecto Hall	Enlace

El coste aproximado de una maqueta completa con los componentes mencionados anteriormente, la fabricación de la placa y el motor DC, ronda los 51.24€ aproximadamente, teniendo en cuenta que todos los elementos como resistencias, condensadores, y sujeciones son adquiridas mediante grandes lotes, lo cual reduce el coste por unidad. Por otro lado, la empresa especializada en la fabricación de la placa electrónica es de origen extranjero, por lo que la mayoría del coste viene dado por impuestos y exportaciones del país (JLCPCB).

C. CÓDIGO FUENTE PRINCIPAL

Debido a la densidad del código empleado en el control de la maqueta, se realizará una visualización de la estructura principal únicamente (archivo .ino). El resto del código se encontrará de forma accesible en los anexos del documento:

```
#include <ESP32Encoder.h>
#include "Librerias.h"

system_t sys;   //Estructura definida en Globales.h que contiene los variables del programa que se graba en la Eeprom

typedef union //Unión para conversión de float a byte
{
  float number;
  uint8_t bytes[4];
} FLOATUNION_t;

//Variables a recibir de simulink
FLOATUNION_t ValorPIDSimulink;
FLOATUNION_t ReferenciaTension;
FLOATUNION_t ElegirEntradas; //Elegir las entradas de simulink o las de la maqueta

//Variables enviadas por la ESP32
FLOATUNION_t SIM_Tiempo;
FLOATUNION_t SIM_TipoControl;
FLOATUNION_t SIM_TipoPID;
FLOATUNION_t SIM_Encoder;
//FLOATUNION_t SIM_PIDSalida;
FLOATUNION_t SIM_Corriente;
FLOATUNION_t SIM_Tension;
FLOATUNION_t SIM_TorqueCarga;

String stringEstado[]={"MARCHA","PARO"};
String stringControl[]={"VELOC","POSIC"};
String stringEntrada[]={"ESCALON","SENO","CONST","RAMPA"};
String stringPID[]={"LOCAL","SIMULINK","IDENTIFICACION"};

```

```
// Motor
#define encoderA 34
#define encoderB 35
#define LOCAL 16
#define SIMULINK 17
#define EnableVent 33 //Control PWM para el segundo motor (Configurado para la refrigeración de la maqueta en caso de ser necesario)
#define CorrienteOpAmp 12    //Valor de corriente amplificado (x45 veces) //44K, 1K --> G = ((44K/1K)+1) = 45
#define CorrienteShunt 14    //Medida sin amplificar de la corriente (desde el emisor de los transistores del puente H)

ESP32Encoder myEnc;

const float SHUNT_OHMIOS = 1.2;    // Valor de la resistencia shunt (Ω)
const float GANANCIA = 41.25;     // Ganancia del amplificador LM358
const float VREF = 3.3;          // Voltaje de referencia del ADC
const int ADC_MAX = 4095;        // Resolución del ADC (12 bits ESP32)

int test_limits = 2; // De rotary Encoder
const int PPR = 22*34; //22 pulsos por revolución, multiplicado por la reductora 1:34
double Kp, Ki, Kd;
double Setpoint, Input, Output;
double velMotor;
int tensionAnterior;
float tension;
float tactual, tanterior, t0, temp, t_freno;

int boton_pulsado=0;
//int estado=PARO;

```

```ino
bool getFloats()
{
  // Esperar a que haya al menos 13 bytes disponibles
  if (Serial.available() >= 13)
  {
    if (Serial.read() != 'B')//Comprobar que se están recibiendo datos de Simulink
    {
      return false;
    }
    else
    {
      tanterior = tactual;
    }
    // Leer 4 bytes del primer float
    for (int i = 0; i < 4; i++)
    {
      ReferenciaTension.bytes[i] = Serial.read();
    }
    // Leer 4 bytes del segundo float
    for (int i = 0; i < 4; i++)
    {
      ValorPIDSimulink.bytes[i] = Serial.read();
    }
    // Leer 4 bytes del tercer float
    for (int i = 0; i < 4; i++)
    {
      ElegirEntradas.bytes[i] = Serial.read();
    }
    return true;
  }
  return false;  //No había suficientes datos
}
```

```ino

PID myPID(&Input, &Output, &Setpoint, Kp, Ki, Kd, DIRECT);

void controlVelocidad();
void controlPosicion();
double  move(double xd, double vmax, double a, double dt);

void setup()
{
  settingsLoadFromEEprom();
  analogReadResolution(12);//Aumentamos la resolución a 2^12 --> 4096 estados
  //analogSetPinAttenuation(CorrienteMotor, ADC_0db);//Reducimos el filtrado de la señal para abarcar un rango más amplio de recepción (corriente del motor)
  analogSetPinAttenuation(CorrienteShunt, ADC_11db);
  analogSetPinAttenuation(CorrienteOpAmp, ADC_11db); //El OpAmp satura a 3.670V con Vcc ~ 5V

  Serial.begin(115200);
  pinMode(LOCAL, OUTPUT);
  pinMode(SIMULINK, OUTPUT);
  pinMode(EnableVent, OUTPUT);
  digitalWrite(EnableVent, HIGH);

  inicializaLcd();                  // Incialización del display
  incializaRotaryEncoder();         //Incialización encoder rotativo HW-040

  myEnc.attachHalfQuad(encoderA, encoderB);
  myEnc.setCount(0);

  setupMovement();                // Incializacion Pines Motor
  myPID.SetSampleTime(20);
  myPID.SetOutputLimits(-255, 255);

  if(sys.estado==PARO) myPID.SetMode(MANUAL);
  else myPID.SetMode(AUTOMATIC);
```

```
128
129   escribeLcd(" UNIVERSIDAD DE","   VALLADOLID");
130   delay(1000);
131   escribeLcd("   ESCUELA DE","INGENIERIAS(EII)");
132   delay(1000);
133   escribeLcd("Estado: "+stringEstado[sys.estado],stringControl[sys.control]+": "+stringEntrada[sys.entrada]);
134   t0 = millis();
135 }
136
137 void loop()
138 {
139   if(botonEncoderPulsado()) menuPrincipal();
140   if(sys.estado==MARCHA)
141   {
142     tactual = millis()-t0;
143     CargaExterna();
144
145     if(sys.control==VELOCIDAD)  controlVelocidad();
146     else if (sys.control==POSICION) controlPosicion(); //Sólo diseñado para LOCAL
147
148     if(sys.tipoPID == PID_SIMULINK) //Transmisión sólo para Simulink
149     {
150      Transmision();
151      if(tactual-tanterior > 1000)
152       {
153        sys.estado = PARO;
154        escribeLcd("ERROR SIMULACION","   PULSE: RUN"); //Desconexión de la maqueta si no está comunicando con simulink
155       }
156     }
157   }
158
```

```
159   else if(sys.estado==PARO)
160   {
161     tanterior = millis()-t0; //Para poder reanudar la marcha en cualquier instante
162     myPID.SetMode(MANUAL);
163     Motor(0, sys.frenoDinamico);
164   }
165
166   switch(sys.tipoPID)
167   {
168     case 0: digitalWrite(LOCAL, HIGH);
169             digitalWrite(SIMULINK, LOW);break;
170     case 1: digitalWrite(LOCAL, LOW);
171             digitalWrite(SIMULINK, HIGH);break;
172     case 2: digitalWrite(LOCAL, LOW);
173             digitalWrite(SIMULINK, LOW);break;
174   }
175 }
176
177 void CargaExterna()
178 {
179   if(sys.frenoDinamico == true) //Código para generar freno dinámico aplicando tensión lógica a ambas direcciones del motor
180    {
181     if(velMotor>0)
182     SIM_TorqueCarga.number = 0.004;
183     else
184     SIM_TorqueCarga.number =-0.004;
185
186     if(millis()-t_freno> 5000)
187     {
188       sys.frenoDinamico = false;
189       SIM_TorqueCarga.number = 0;
190     }
191    }
```

```
    else
    {
     t_freno = millis();
    }
}

void Transmision()
{
  //Recibo Simulink
  getFloats();
  // Envio variables a simulink
  SIM_Tiempo.number = TiempoReal();
  SIM_TipoControl.number = sys.control;
  SIM_TipoPID.number = sys.tipoPID;
  SIM_Encoder.number = Input;
  SIM_Corriente.number = MedirCorriente(); //Corriente en mA
  SIM_Tension.number = tension; //Valor de tensión desde la maqueta

  //Trama para enviar todos los paquetes
  Serial.write('A');
  // Print float data
  for (int i=0; i<4; i++)
  {
    Serial.write(SIM_Tiempo.bytes[i]);
  }
  for (int i=0; i<4; i++)
  {
    Serial.write(SIM_TipoControl.bytes[i]);
  }
  for (int i=0; i<4; i++)
  {
    Serial.write(SIM_TipoPID.bytes[i]);
  }
```

```
  for (int i=0; i<4; i++)
  {
    Serial.write(SIM_Encoder.bytes[i]);
  }
  for (int i=0; i<4; i++)
  {
    Serial.write(SIM_Corriente.bytes[i]);
  }
  for (int i=0; i<4; i++)
  {
    Serial.write(SIM_Tension.bytes[i]);
  }
  for (int i=0; i<4; i++)
  {
    Serial.write(SIM_TorqueCarga.bytes[i]);
  }
  // Print terminator
  Serial.print('\n');
  delay(20);
  // Use the same delay in the Serial Receive block
}

float MedirCorriente()
{
  int lecturaADC = 0;
  long suma = 0;
  const int muestras = 64;

  // Promedio para reducir ruido
  for (int i = 0; i < muestras; i++)
  {
    suma += analogRead(CorrienteOpAmp);
  }
```

```
  lecturaADC = suma / muestras;

  // Convertir a tensión en la entrada del ESP32
  float voltajeLeido = (lecturaADC * VREF) / ADC_MAX;

  // Retrocalcular la tensión original en el shunt
  float voltajeShunt = voltajeLeido / GANANCIA;

  // Calcular corriente según ley de Ohm (I = V / R) en mA
  float corriente = voltajeShunt * 1000 / SHUNT_OHMIOS;

  // === SI LA CORRIENTE SUPERA UMBRAL, CAMBIAMOS A LECTURA DIRECTA ===
  if (abs(corriente) > 66.0)
  {
    suma = 0;
    for (int i = 0; i < muestras; i++)
    {
      suma += analogRead(CorrienteShunt);
    }
    lecturaADC = suma / muestras;
    voltajeShunt = (lecturaADC * VREF) / ADC_MAX; // ahora sin amplificación
    float corrienteDirecta = (voltajeShunt * 1000) / SHUNT_OHMIOS;
    corriente = corriente + corrienteDirecta;
  }

  if(sys.tipoPID != 2)
  {
    if(tension < 0)
    {
    corriente = -corriente;
    }
  }
```

```
  else
  {
    if(sys.Tension < 0)
    {
    corriente = -corriente;
    }
  }
  return corriente;
}

float TiempoReal() //Función para realizar entrada escalón ideal (pendiente infinita)
{
    if((abs(tensionAnterior-tension)>0)&&(sys.entrada==ESCALON))
    {
      temp = tanterior;
    }
    else
    {
      temp = tactual;
    }

   if (isnan(tension))
   tension = tensionAnterior; //o último valor válido
   else
   tensionAnterior = tension;

   return temp/1000;
}
```

```
void controlVelocidad()
{
  unsigned long int t, dt=100; // intervalo para el calculo de velocidad
  float tiempo;
  int8_t inc;
  double Pos;

  static double Pos_ant = myEnc.getCount();

  //float periodo;
  static unsigned long int tinicio=millis();
  static unsigned long int t_ant=millis();  // Temporizador para calculo velocidad
  static unsigned long int t_ant2=millis();// Temporizador para calculo Periodo

  if(sys.tipoPID == PID_SIMULINK)
  {
   if(ElegirEntradas.number)
   {
    tension = ReferenciaTension.number;
   }
   else
   {
    tension = sys.Tension;
   }
  }

// Calculo velocidad
  t=millis();

  if(t-t_ant> dt)
  {
    Pos= myEnc.getCount();
    Input=(double)1000*(Pos-Pos_ant)*60/((t-t_ant)*PPR);  // Calculo la velocidad en RPM
    t_ant=t;
    Pos_ant=Pos;
```

```
  switch(sys.tipoPID)
  {
    case 0:
    Serial.print("+SetPoint(RPM):"+String(Setpoint));
    Serial.print(", ");
    if(sys.entrada==ESCALON)
    {
     Serial.print("SetPoint(RPM):"+String(-Setpoint));
     Serial.print(", ");
    }
    Serial.print("VelocidadReal(RPM):"+String(Input));
    Serial.print(", ");
    Serial.println("ValorPID(0-255):"+String(Output));
    Serial.print(", ");
    Serial.println("Corriente(mA):"+String(MedirCorriente(), 2));
    escribeLcd1(stringEstado[sys.estado]+":"+String(Input)+"RPM");break;

    case 1: //escribeLcd(stringEstado[sys.estado]+":"+String(Input)+"RPM", stringEntrada[sys.entrada]+":"+String(tension)+"V");break;
            escribeLcd(stringEntrada[sys.entrada]+":"+String(Input)+"RPM"/*+" t:"+String(SIM_Tiempo.number)+"s"*/, String(tension)+"V "+String(MedirCorriente(), 2)+"mA");break;

    case 2:
    Serial.print("+RefTension(V):"+String(sys.Tension));
    Serial.print(", ");
    /*if(sys.entrada==ESCALON)
    {
     Serial.print("RefTension(V):"+String(-sys.Tension));
     Serial.print(", ");
    }*/
    Serial.print("VelocidadReal(RPM):"+String(Input));
    Serial.print(", ");
    Serial.println("Corriente(mA):"+String(MedirCorriente(), 2));
    escribeLcd(stringEntrada[sys.entrada]+":"+String(Input)+"RPM", String(sys.Tension)+"V "+String(MedirCorriente(), 2)+"mA");break;
  }
}
```

```
388   if(sys.entrada==ESCALON)
389   {
390     inc=deltaEncoder();
391     if(sys.tipoPID == 0)
392     {
393      if(inc!=0) Setpoint=(Setpoint>0?Setpoint+inc:Setpoint-inc); // Puedo cambiar la referencia con el rotary encoder
394      if(t-t_ant2>sys.periodo*1000)
395      {         // Cambio sentido
396       Setpoint=-Setpoint;
397       t_ant2=t;
398      }
399     }
400     else
401     {
402      if(inc!=0) sys.Tension=(sys.Tension>0?sys.Tension+inc:sys.Tension-inc); // Puedo cambiar la referencia con el rotary encoder
403      if(sys.Tension>12)sys.Tension=sys.Tension-inc;
404      if(sys.Tension<-12)sys.Tension=sys.Tension+inc;
405      if(t-t_ant2>sys.periodo*1000)
406      {         // Cambio sentido
407       sys.Tension=-sys.Tension;
408       t_ant2=t;
409      }
410     }
411   }
412   else if(sys.entrada==CONST) //Valor constante, variable con el encoder
413   {
414     inc=deltaEncoder();
415     if(sys.tipoPID == 0)
416     {
417      if(inc!=0) Setpoint=(Setpoint>0?Setpoint+inc:Setpoint-inc); // Puedo cambiar la referencia con el rotary encoder
418      if(Setpoint>12)Setpoint=Setpoint-inc;
419      if(Setpoint<-12)Setpoint=Setpoint+inc;
420     }
```

```
421     else
422     {
423      if(inc!=0) sys.Tension=(sys.Tension>0?sys.Tension+inc:sys.Tension-inc); // Puedo cambiar la referencia con el rotary encoder
424      if(sys.Tension>12)sys.Tension=sys.Tension-inc;
425      if(sys.Tension<-12)sys.Tension=sys.Tension+inc;
426     }
427   }
428   else if(sys.entrada==SENO)
429   {
430    tiempo=(float)(millis()-tinicio);
431
432    if(sys.tipoPID == 0)
433    Setpoint = sys.setPoint*sin(2*PI*tiempo/(2*sys.periodo*1000)+PI);
434    else
435    {
436     sys.Tension = 12*sin(2*PI*tiempo/(2*sys.periodo*1000)+PI);
437    }
438   }
439   else if(sys.entrada==RAMPA)
440   {
441     tiempo=(float)(millis()-tinicio);
442
443     // Convertir el tiempo actual en segundos
444     float t_s = tiempo/1000.0;
445
446     // Normalizar dentro de un ciclo
447     float t_ciclo = fmod(t_s, sys.periodo);
448
449     // Rampa lineal: va de 0 a 12 V durante un período
450     float rampa = (t_ciclo/sys.periodo)*12.0;
451
452     if(sys.tipoPID == 0)
453     Setpoint = (sys.setPoint/12.0)*rampa;  // Escalado si setPoint < 12
```

```
    else
    {
     sys.Tension = rampa;
    }
    //Setpoint=move(sys.setPoint, 1000, 800, 0.1); //double xd, double vmax, double a, double dt);
  }

  if(Setpoint - Input > 50 || Setpoint - Input <-50)
  {
      Kp=sys.kPZMVel; //0.1
      Ki=sys.kIZMVel; //0.9
      Kd=sys.kDZMVel; //0
  }
  else
  {
      Kp=sys.kPVel*0.1;// 0.01
      Ki=sys.kIVel*10; //3.5
      Kd=sys.kDVel*0.1; //0.2
  }
  myPID.SetTunings(Kp,Ki,Kd);
  myPID.Compute();

  switch(sys.tipoPID)
  {
    case 0:velMotor = Output;break;
    case 1:
    if(tension >= 0)
    {
      velMotor = map(tension, 0, 12, 0, 255);
    }
    else
    {
      velMotor = - map(abs(tension), 0, 12, 0, 255);
    };break;
    case 2:
    if(sys.Tension >= 0)
    {
      velMotor = map(sys.Tension, 0, 12, 0, 255);
    }
    else
    {
      velMotor = - map(abs(sys.Tension), 0, 12, 0, 255);
    };break;
  }
  Motor(velMotor, sys.frenoDinamico);
}

void controlPosicion()
{
  float tiempo;
  unsigned long int t, dt=100; // intervalo para la visualización valores
  static unsigned long int tinicio=millis();  // Para calculo  funcion sinoidal
  static unsigned long int t_ant=millis();  // Temporizador para calculo velocidad
  static unsigned long int t_ant2=millis();// Temporizador para calculo Periodo
  //static int dir=1;

  t=millis();

    if(sys.entrada==ESCALON)
    {
    int8_t inc=deltaEncoder();
    if(inc!=0) Setpoint=(Setpoint>0?Setpoint+inc:Setpoint-inc); // Puedo cambiara la referencia con el rotary encoder

    if(t-t_ant2>sys.periodo*1000)
    {        // Cambio sentido
       Setpoint=-Setpoint;
```

```
520         t_ant2=t;
521       }
522
523     }
524     else if(sys.entrada==CONST)
525     {
526       int8_t inc=deltaEncoder();
527       if(inc!=0)
528       {
529         if(sys.tipoPID == 0)
530         {
531           //Setpoint+=inc*10;
532           if(inc!=0) Setpoint=(Setpoint>0?Setpoint+inc:Setpoint-inc); // Puedo cambiar la referencia con el rotary encoder
533           if(Setpoint>12)Setpoint=Setpoint-inc;
534           if(Setpoint<-12)Setpoint=Setpoint+inc;
535         }
536         else
537         {
538           if(inc!=0) sys.Tension=(sys.Tension>0?sys.Tension+inc:sys.Tension-inc); // Puedo cambiar la referencia con el rotary encoder
539           if(sys.Tension>12)sys.Tension=sys.Tension-inc;
540           if(sys.Tension<-12)sys.Tension=sys.Tension+inc;
541         } // cada pulso se multiplica *10 para cambios más rápidos
542       }
543     }
544     else if(sys.entrada==SENO)
545     {
546       tiempo=(float)(millis()-tinicio);
547       Setpoint = sys.setPoint*sin(2*PI*tiempo/(2*sys.periodo*1000)+PI)-200;
548
549     }
550     else if(sys.entrada==RAMPA)
551     {
552       if((abs(Setpoint-sys.setPoint))==0)
```

```
553         { // Si se alcanza el destino cambia la posicion
554             sys.setPoint=-sys.setPoint;
555         }
556         Setpoint=move(sys.setPoint, 1000, 800, 0.1); //double xd, double vmax, double a, double dt);
557       }
558       //Input = myEnc.read(); // Lee encoder del motor
559       //Input = myEnc.readEncoder();;
560       Input = myEnc.getCount();
561       if(Output<140 || Output>-140)
562       {
563         Kp=sys.kPZMPos*0.1; //3
564         Ki=sys.kIZMPos*0.1; //0.05
565         Kd=sys.kDZMPos*0.1; //0.0
566       }
567       else
568       {
569         Kp=sys.kPPos*0.1; //0.76
570         Ki=sys.kIPos*0.1; //0.05
571         Kd=sys.kDPos*0.1; //0.05
572       }
573
574       myPID.SetTunings(Kp,Ki,Kd);
575       myPID.Compute();
576
577       if(t-t_ant> dt && sys.entrada!=RAMPA)
578       { /*                                    // Imprime valores cada periodo de muestreo
579         Serial.print("+SetPoint:"+String(Setpoint));
580         Serial.print(", ");
581         if(sys.entrada==ESCALON)
582         {
583           Serial.print("SetPoint:"+String(-Setpoint));
584           Serial.print(", ");
585         }
```

```
    Serial.print("Input:"+String(Input));
    Serial.print(", ");
    Serial.println("Output:"+String(Output));
    Serial.print(", ");
    Serial.print("Corriente:"+String(MedirCorriente()));*/
    escribeLcd1(stringEstado[sys.estado]+": "+String(Input));
    t_ant=t;
   }

   if(sys.tipoPID == 0)
   {
    velMotor=Output; //Cambiar signo si el motor gira solo en un sentido
   }

   Motor(velMotor, sys.frenoDinamico);
}

double  move(double xd, double vmax, double a, double dt)
{
  static unsigned long int t_ant=millis();

  static double v,x,x_ant=myEnc.getCount();
  unsigned long int t;
  int sentido_giro;

  x=myEnc.getCount();
  t=millis();
  if(t-t_ant>dt*1000){

      t_ant=t;
      sentido_giro=(xd-x>0?1:-1);
```

```
    if(abs(xd-x) >2*vmax*vmax*dt/a){
    //if(abs(xd-x) >500){
      if(abs(v) < vmax){      // Acelero
          x=x+v*dt+0.5*a*dt*dt*sentido_giro;
          v=v+a*dt*sentido_giro;
          Serial.println("Acelero");
      }
      else{
          v=vmax*sentido_giro;
          x=x+v*dt;        // Vel constante
          Serial.println("Vel constante");
      }
    }
    else{                           // Decelero
        x=x+v*dt-0.5*a*dt*dt*sentido_giro;
         v=v-a*dt*sentido_giro;
        if((xd-x)*sentido_giro<0){
          x=xd;    // Comprobamos no pasarnos
          v=0;
        }
        Serial.println("Decelero");
    }
    Serial.print("xd:"+String(xd));
    Serial.print(", ");
    Serial.print("x:"+String(x));
    Serial.print(", ");
    Serial.print("Input:"+String((int32_t)myEnc.getCount()));
    Serial.print(", ");
    Serial.println("v:"+String(v));
  }
 return x;
}
```

Figura 67: extracto del código ".ino". Elaboración propia.

D. MODELOS 3D (STL)

Todos los modelos impresos en 3D han sido diseñados en el programa Fusion360 de Autodesk (compatible con archivos de Autodesk Inventor):

Figura 68: Soporte para pantalla LCD 1602 I2C del interfaz Humano – Máquina. Elaboración propia.

El soporte para el HMI se ha realizado con el objetivo de optimizar el material de impresión 3D sin la necesidad de perder funcionalidad. La pantalla LCD se instala mediante cuatro tornillos de métrica M3.

Existen versiones anteriores donde se incluye el codificador incremental en el propio soporte de la pantalla. Para evitar el desgaste temprano por uso continuado de las maquetas, se ha decidido diseñar un soporte independiente para el codificador.

Figura 69: Versiones anteriores del HMI con el codificador integrado, junto a la versión actual. Elaboración propia.

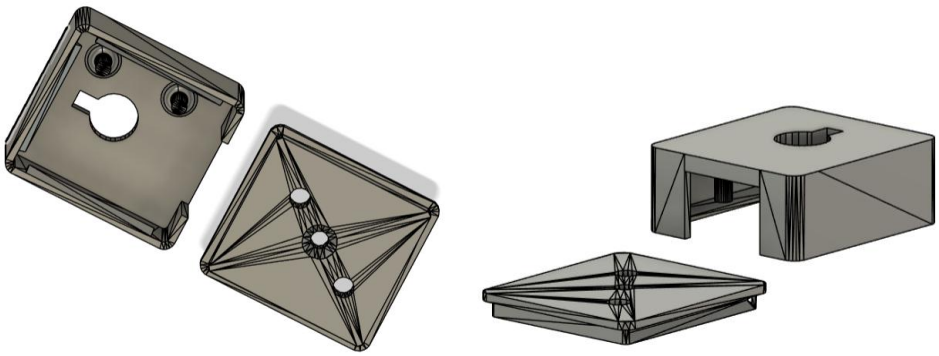


Figura 70: Soporte para el codificador incremental (encoder) KY – 040. Elaboración propia.

El diseño para el codificador incremental se basa en dos piezas por ajuste a presión. La base del conjunto se fija a la maqueta con tornillería de métrica M3, mientras que la parte superior se ajusta a presión contra la base. Tras varias pruebas de robustez, el codificador queda correctamente posicionado y las piezas no adquieren holgura. Este método es eficaz para realizar reemplazos del codificador en caso de daño al componente.

El diseño del soporte del codificador, parte de un modelo descargado de Printables (https://www.printables.com/model/869342-rotary-encoder-ky-040-ec11-housing-with-snap-fit-a), creado por su autor original, que fue posteriormente editado y adaptado para este proyecto. Se reconoce expresamente la autoría del diseño base, y las modificaciones realizadas corresponden únicamente a ajustes mecánicos específicos para su integración en la maqueta.

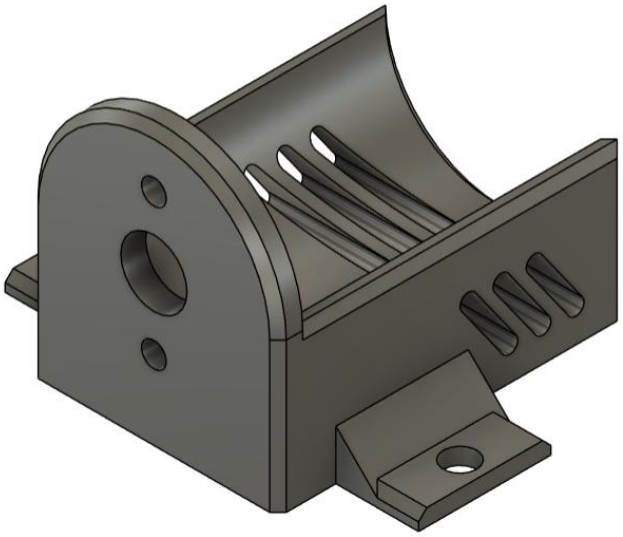


Figura 71: Soporte para motor JGA25 – 370 con reductora. Elaboración propia.

El soporte para motor se fija directamente a la maqueta con tornillería de M3. Su diseño permite la correcta alineación del eje, evitando vibraciones y reduciendo en medida de lo posible la cantidad de material empleado.

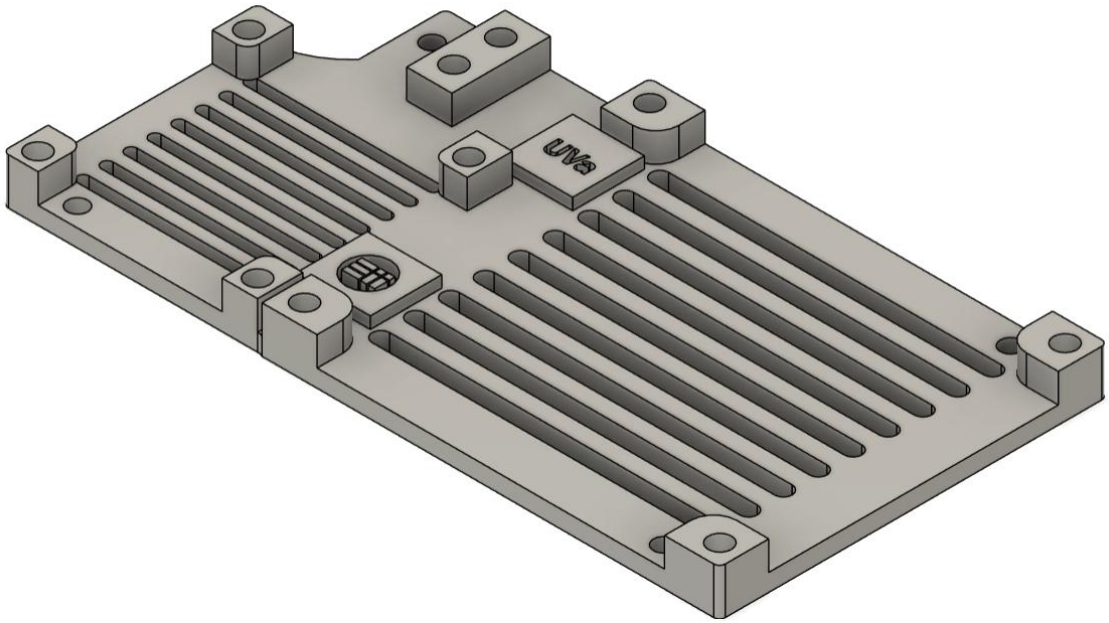


Figura 72: OPCIONAL, Soporte para módulos de control. Elaboración propia.

Como alternativa a la fabricación de la placa electrónica, es posible realizar una maqueta similar a partir de módulos, concretamente el L298N (con el ZIP-15 integrado), un adaptador de pines a borneras para ESP32 con 19 filas a cada lateral y, por último, un sensor de corriente de efecto Hall ACS712.

E. ARCHIVOS GERBER DE LA PLACA

Los archivos Gerber y de taladrado se encuentran anexados junto al trabajo en una carpeta comprimida .zip. A continuación, se realiza una previsualización de los archivos Gerber en el software KiCAD:

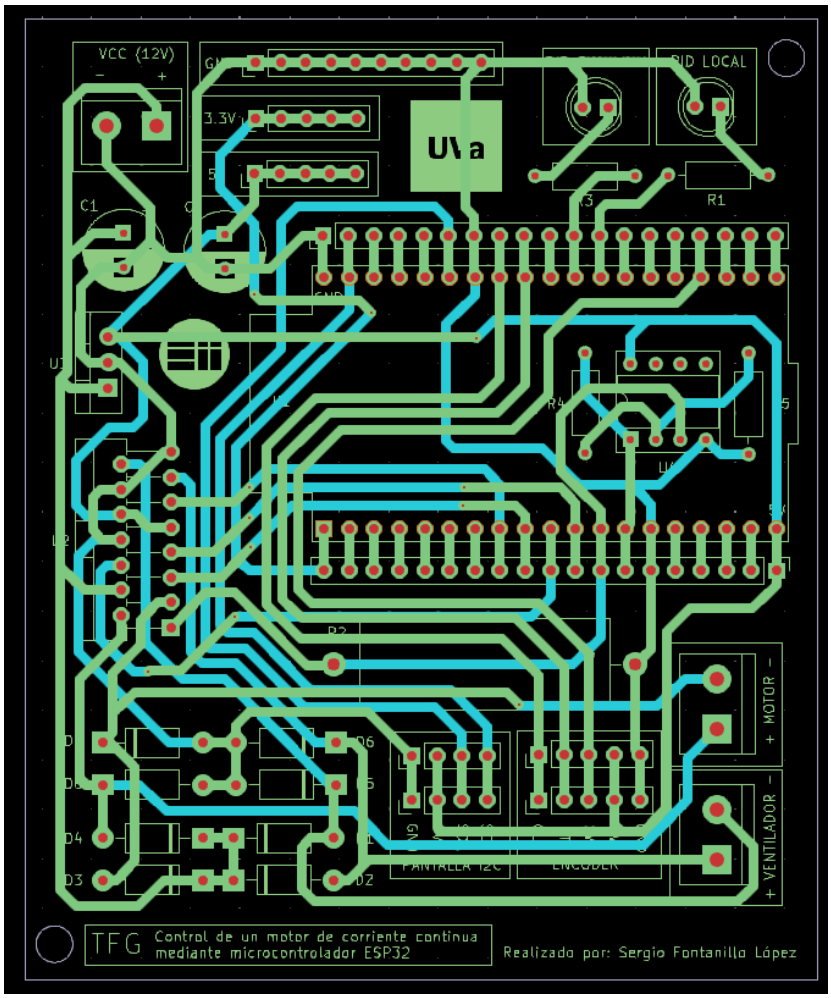


Figura 73: Previsualización archivos Gerber. Elaboración propia.

F. MANUAL DE USUARIO Y ANEXOS EXTERNOS AL DOCUMENTO

Se adjuntan las hojas de datos de cada componente electrónico empleado en este proyecto, así como el manual de usuario para el uso del microcontrolador ESP32 DevKit V1. El resto de información sobre el uso de la maqueta se encuentra a lo largo de este trabajo, así como en los anexos externos al documento:

01 - Modelos CAD Inventor (IPT)

02 - Código Fuente ESP32

03 - Hojas de Datos Componentes

04 - Fotolitos PCB

05 - Modelos 3D (STL)

06 - Documentación Placa Electrónica

07 - Modelos Simulink MATLAB

08 - Archivos Gerber Producción

09 - Anexo de Prácticas