ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

Diseño e implementación de sistemas de control embebidos; PID, por retroalimentación de estados y lógica difusa en plantas de temperatura y velocidad de motor DC, para prácticas en laboratorio de control automático y microcontroladores.

PROYECTO INTEGRADOR

Previo la obtención del Título de:

Ingeniero en Electrónica y Automatización Industrial

Presentado por:

Manuel Jesús Martínez Ramírez
Javier Daniel Nicolalde Perugachi

GUAYAQUIL - ECUADOR

Año: 2023

DEDICATORIA

Dedico en primer lugar, a mi padre, José Néstor Nicolalde, quien me ha dado su apoyo incondicional en todo aspecto y me ha enseñado lo importante que es trabajar duro por lo que uno se propone.

A mis familiares quienes han sabido aconsejarme y darme motivación para seguir adelante en mi desarrollo personal.

A mis profesores y compañeros que han sido parte de mi formación tanto del aspecto profesional como ético.

Javier Daniel Nicolalde Perugachi.

DEDICATORIA

Le dedico el resultado de este trabajo a mi familia, en especial a mis padres Marcos Martínez Cuvi y Rosa Ramírez Galora quienes han sido de sustento, inspiración y fuente de ánimo en cada etapa de formación personal y académica.

A la ESPOL, mis profesores quienes han sido de guía, compañeros y amigos de carrera que aportaron grandemente en mi formación profesional.

Manuel Jesús Martínez Ramírez.

AGRADECIMIENTOS

Estoy profundamente agradecido con mi familia y las personas que han permitido que este trabajo y mi formación profesional se haya culminado con éxito.

Agradezco a los profesores Dennys Cortez y Ronald Solís quienes brindaron su guía oportuna durante el desarrollo del proyecto.

Javier Daniel Nicolalde Perugachi.

AGRADECIMIENTOS

Agradezco en primer lugar a Dios, que en su gracia me ha permitido llegar a formarme en lo académico y completar esta etapa mediante este proyecto.

A mis hermanos, en especial a Lidia y Ángel que han sido como mis padres, mostrando siempre un apoyo incondicional en lo económico, emocional, mediante consejos y ánimo lo cual me ha permitido culminar la carrera y este trabajo de la mejor forma.

Manuel Jesús Martínez Ramírez.

DECLARACIÓN EXPRESA

"Los derechos de titularidad y explotación, nos corresponde conforme al reglamento de propiedad intelectual de la institución; Manuel Jesús Martínez Ramírez y Javier Daniel Nicolalde Perugachi damos nuestro consentimiento para que la ESPOL realice la comunicación pública de la obra por cualquier medio con el fin de promover la consulta, difusión y uso público de la producción intelectual"

Manuel Jesús Martínez Ramírez

Javier Daniel Nicolalde Perugachi

EVALUADORES

MSc. Dennys Dick Cortez Alvarez
Profesor de Materia

MSc. Ronald David Solís Mesa
Tutor de proyecto

RESUMEN

En este trabajo se diseñó e implementó un sistema electrónico basado en las tarjetas TSC-LAB, ESP32 y Arduino Uno, que permite el control y monitoreo en tiempo real de una planta electrónica de temperatura y una planta de control de velocidad de un motor DC. Mediante programación en lenguaje C++ modificado de Arduino, se diseñó los controladores PID por sintonización Ziegler-Nichols, sintonización prueba error, lógica difusa y realimentación de estados. El objetivo principal es que este desarrollo permita fortalecer el conocimiento de diseño de controladores y programación de sistemas embebidos en los laboratorios de sistemas de control y sistemas embebidos de la ESPOL durante el desarrollo profesional.

Durante la metodología, se empezó con el diseño modular del sistema electrónico, basado en las tarjetas mencionadas; para ello se diseña tarjetas PCB que permite adaptar e interconectar dichos dispositivos. Luego de ello, se realiza el diseño de los controladores usando MATLAB para luego embeberlos en las tarjetas de desarrollo mediante programación en C++ usando el IDE de Arduino. Con el objetivo de visualizar en tiempo real la variable controlada de lo desarrollado se realiza una aplicación usando la herramienta AppDesigner de Matlab; la interfaz permite visualizar el desempeño de los controladores al manipular el valor de referencia e incluso sintonizar el control PID por prueba-error manipulando las contantes Kp, Ti y Td.

Finalmente, se realizan varias pruebas del desempeño de los controladores tanto usando la tarjeta ESP32 como la de Arduino Uno, se realiza un análisis de desempeño de ambas y se muestra las conclusiones de los resultados obtenidos experimentalmente.

Palabras Clave: Sistema Embebido, Lógica difusa, Controladores, Realimentación de estados, PID.

ABSTRACT

In this work, an electronic system based on the TSC-LAB, ESP32 and Arduino Uno cards was designed and implemented, which allows the control and monitoring in real time of an electronic temperature plant and a speed control plant of a DC motor. Through programming in C++ language modified from Arduino, the PID controllers were designed by Ziegler-Nichol's tuning, error-proof tuning, fuzzy logic and state feedback. The main objective is that this development allows to strengthen the knowledge of controller design and programming of embedded systems in the laboratories of the university during professional development.

During the methodology, we started with the modular design of the electronic system, based on the cards; for this purpose, PCB cards are designed to adapt and interconnect these devices.; For this, PCB cards are designed that allow adapting and interconnecting said devices. After that, the design of the controllers is carried out using MATLAB and then they are embedded in the development boards by programming in C++ and using the Arduino IDE. In order to visualize in real time, the controlled variable of what has been developed, an application is made using the Matlab AppDesigner tool; the interface allows to visualize the performance of the controllers when manipulating the reference value and even to tune the PID control by trial-error manipulating the constants Kp, Ti and Td.

Finally, several tests of the performance of the controllers are carried out both using the ESP32 card and the Arduino Uno card, a performance analysis of both is carried out and the conclusions of the results obtained experimentally are shown.

Keywords: Embedded System, Fuzzy Logic, Controllers, State Feedback, PID.

ÍNDICE GENERAL

ABREVIATURAS

PE Prueba Error

ZN Ziegler-Nichols

RE Realimentación de estados

PCB Placa de circuito impreso

Kp Constante de control proporcional

Td Constante de Tiempo de derivativo

Ti Constante de tiempo integral

PWM Modulación de ancho de pulso

FM Fuzzy-Mamdani

PID Proporcional-Integral-Derivativo

IoT Internet Of Things

SIMBOLOGÍA

V	Voltio
I	Amperio
m	Metro
mV	Milivoltio
Q	Cantidad de Calor
k_T	Constante de pérdida de calor para el ambiente.
T	Temperatura
T_A	Temperatura ambiente.
ϵ	Emisividad
σ	Constante de Boltzmann
A	Área

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE PLANOS

APENDICE B.

APENDICE C.

CAPÍTULO 1

INTRODUCCIÓN

El estudio de los sistemas de control ha evolucionado notablemente con el paso del tiempo, de tal manera que, ahora es posible llevar a cabo técnicas de control moderno que permiten una eficiencia considerable respecto a años anteriores. Un aspecto relevante que ha contribuido a esto es la revolución tecnológica; ésta ha facilitado el diseño e implementación de distintos tipos de controladores, tal como se pretende mostrar en este trabajo. En el ámbito educativo de ingenierías se ha estudiado las técnicas de diseño de control y su análisis mediante simuladores especializados como Matlab/ Simulink, sin embargo, en muchos casos no se ha implementado proyectos aplicados a plantas con procesos reales, lo que es una desventaja competitiva ya en el campo laboral. La temática de este proyecto plantea comparar el desempeño de los controladores PID clásico, por realimentación de estados y lógica difusa mediante el uso tarjetas de desarrollo basadas en microcontroladores de bajo costo, como lo son Arduino y ESP32. Algunos autores de referencia ya han mostrado investigaciones relacionadas con microcontroladores PIC y Arduino, pero no se ha implementado un desarrollo comparativo ni un enfoque educativo para reforzar conocimientos prácticos, implementándolas en las placas electrónicas mencionadas. Con este trabajo se pretende construir una base firme sobre la cual se pueda partir para el desarrollo nuevos diseños y aplicaciones de sistemas de control con enfoque industrial.

1.1 Descripción del problema.

El laboratorio de control avanzado y microcontroladores de la facultad de Ingeniería Eléctrica y Computación en ESPOL necesita de un sistema que permita poner a prueba y medir el desempeño de distintos tipos de controladores diseñados ante una planta de un proceso que puede ser temperatura, presión, caudal, etc. El problema radica en que no existe un sistema de control moderno que haya sido embebido en un microcontrolador para el ámbito educativo y es necesario para poder mostrar un análisis del desempeño de las diferentes técnicas de control moderno durante su etapa de formación profesional.

1.2 Justificación del problema

Durante la etapa de formación profesional, los estudiantes de la materia de control avanzado adquieren conocimientos teóricos sobre el diseño de distintos tipos de controladores y estos son puestos a prueba mediante simulaciones con la herramienta Matlab/Simulink quedando vacíos respecto a su comportamiento e implementación en una planta real. Por ello, se propone implementar tres tipos de controladores como base para aplicaciones y mejoras futuras. Los controladores PID, por lógica difusa y realimentación de estados son algunos aprendidos académicamente, con los cuales los estudiantes podrán aterrizar conceptos y además aplicar conocimientos de la materia de sistemas embebidos para integrarlos en tarjetas de desarrollo como ESP32 y Arduino Uno. Adicionalmente, dicha implementación permitirá fortalecer habilidades prácticas con placas electrónicas a bajo costo para mostrar el desempeño y posibles desarrollos a soluciones de la industria.

1.3 Objetivos

1.3.1 Objetivo general.

Diseñar e implementar un sistema embebido basado en microcontroladores ESP32 y Arduino que permitan poner a prueba los distintos tipos de controladores modernos diseñados en al menos dos plantas electrónicas.

1.3.2 Objetivos específicos.

1. Diseñar un controlador PID discreto que permita la sintonización prueba-error en las tarjetas ESP32 y Arduino para el control de temperatura y velocidad.
2. Diseñar una interfaz gráfica que permita la interacción y visualización del desempeño de los controladores diseñados en tiempo real con la tarjeta TSC-LAB.
3. Implementar un prototipo electrónico que permita la interacción con los controladores diseñados en las tarjetas ESP32 y Arduino para la realización de prácticas los laboratorios de microcontroladores y control.
4. Diseñar un controlador PID discreto que permita la sintonización de Ziegler-Nichols para su implementación y pruebas de desempeño en la tarjeta TSC-LAB.
5. Diseñar un controlador por realimentación de estados y lógica difusa para su implementación y pruebas de desempeño en la tarjeta TSC-LAB.
6. Comparar el desempeño de los tres tipos de controladores implementados en las tarjetas de desarrollo ESP32 y Arduino.

1.4 Marco teórico

1.4.1 Sistema de control.

Un sistema de control es una combinación de componentes que en base a un requerimiento proporcionado en su entrada actúa o realiza una acción de manera controlada con el fin de cumplir un objetivo deseado. Un sistema no necesariamente debe ser físico, puede ser económico, biológico, biomédico e incluso socioeconómico, etc. (Ogata, Ingeniería de Control Moderna, 1998).

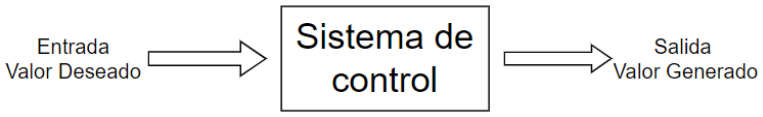


Figura 1. Ejemplo gráfico de un Sistema de Control

1.4.2 Componentes principales de un sistema de control.

- **Planta:** Esta constituida por varios componentes de un sistema, la cual ejecuta ciertas operaciones para cumplir con un requerimiento dado. En este caso llamaremos planta a una tarjeta electrónica conocida como TSC-LAB la cual cuenta con una Planta de Temperatura y Velocidad.
- **Variable controlada:** Corresponde a la variable física que se monitorea mediante un transductor de modo que esta pueda ser comparada con un valor objetivo.
- **Variable manipulada:** Es aquella mediante la cual el sistema puede actuar para provocar cambios deseados en el estado de la planta.

1.4.3 Tipos de sistemas de control.

- **Sistema de control en lazo cerrado:** Los sistemas de control que realizan el monitoreo de la variable deseada para ser comparada con la referencia son

considerados como control en lazo cerrado. El objetivo conlleva a que el sistema pueda realizar una acción de control y reducir el error en respuesta (Ogata, Ingeniería de Control Moderna, 1998).

- **Sistema de control en lazo abierto:** Son aquellos sistemas que no realizan el monitoreo de la respuesta de la planta, razón por la que no son utilizados el sistema en los que se requiere precisión o respuesta controlada.

1.4.4 Sistemas de control en tiempo discreto

Un sistema de control en tiempo discreto es aquel que cuantifica una o más variables en forma de muestras, es decir, toma valores por intervalos regulares y finitos en contraposición a un sistema continuo. Los tiempos discretos se los puede representar con t_k donde $k = 0, 1, 2, ...$ (Ogata, Sistema de control en tiempo discreto 2a edición, 1996).

Las señales continuas son aquellas que representan al comportamiento de un sistema físico como la electricidad, la temperatura, etc. A continuación, se muestra una señal continua y su respectiva señal discretizada.

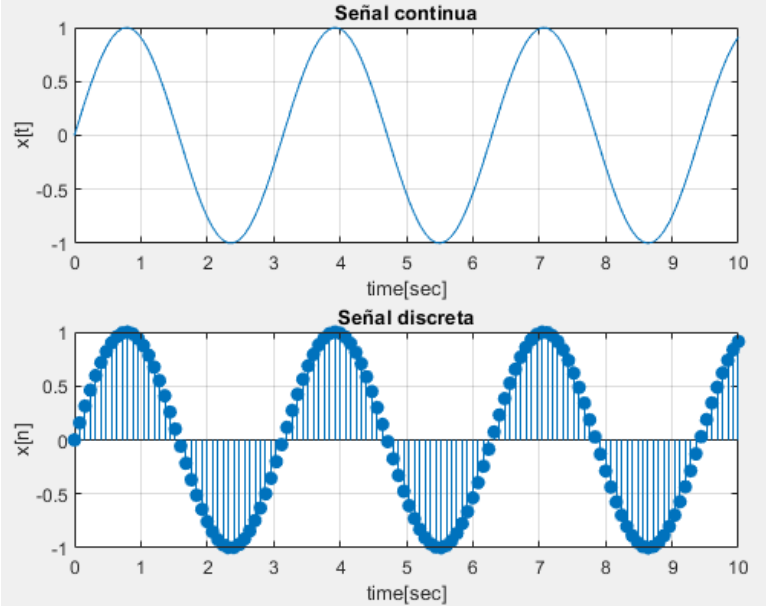


Figura 2. Señal continua y señal discretizada.

1.4.5 Tiempo de muestreo de una señal

Un proceso o comportamiento de un sistema físico se puede muestrear para evitar señales intermitentes o simplemente aproximar su naturaleza analógica mediante señales digitales; un aspecto crítico es el tiempo de muestreo de Nyquist-Shannon, de acuerdo con (Ogata, Sistema de control en tiempo discreto 2a edición, 1996) si se muestrea con una frecuencia mayor o igual a por lo menos dos veces la componente de mayor frecuencia de la señal continua, la señal no perderá información necesaria para su reconstrucción original:

$$\omega_s \geq 2\omega_1 \tag{1.1}$$

Si no se respeta este teorema la señal muestreada se traslapará y no se podría recuperar la señal original, conocido este fenómeno como aliasing.

1.4.6 Control Proporcional-Integral-Derivativo (PID) discreto

Un controlador PID consta de tres componentes, una parte proporcional, una integral y una derivativa. La acción proporcional (P) aumenta conforme aumenta el error presente del sistema; si se incrementa en exceso puede generar oscilaciones, en cambio sí disminuye el control es lento. En ambos casos siempre está presente un error de estado estacionario (Jaimes, 2009).

La acción integral (I) se asocia al error de estado estacionario; se acumula el error pasado funcionando como un registro de memoria. Aumentar en exceso esta acción causa oscilaciones que conllevan a la inestabilidad, mientras que, su disminución excesiva hace que el controlador sea menos agresivo con los errores acumulados, lo que lleva a un error de estado estacionario mayor.

La acción derivativa (D) permite hacer predicciones de errores futuros; asociado a la velocidad de respuesta ante variaciones en el error para que no haya cambios bruscos. Si se aumenta en exceso se vuelve susceptible al ruido y a la inestabilidad, mientras que para valores pequeños la respuesta es más lenta a cambios repentinos en el sistema. El control PID en tiempo continuo es:

$$u(t) = k_c e(t) + \frac{k_c}{\tau_i}\int_0^t e(t)dt + k_c\tau_d\frac{de(t)}{dt} \qquad (1.2)$$

Discretizando obtenemos la siguiente ecuación en diferencias:

$$u(k) = u(k-1) + q_0 e(k) + q_1 e(k-1) + q_2 e(k-2) \qquad (1.3)$$

Donde:

$$q_0 = k_c\left(1 + \frac{T_s}{2\tau_i} + \frac{\tau_d}{T_s}\right);\ q_1 = -k_c\left(1 - \frac{T_s}{2\tau_i} + \frac{2\tau_d}{T_s}\right);\ q_2 = k_c\frac{\tau_d}{T_s} \qquad (1.4)$$

Existen varios trabajos que usan control discreto, por mencionar uno, se ha empleado lo desarrollado anteriormente en una planta de temperatura para el curado de prendas índigo; para ello, se adquirió datos mediante un software como LabVIEW y Matlab y se diseñó un controlador PI por sintonización de Ziegler-Nichols (ZN), luego, se realizó el proceso de embeber este controlador en un microcontrolador PIC16F887 mediante programación y finalmente realizando pruebas experimentales (Castaño Giraldo, Hernández Gómez, & Gallo Blandón, 2013).

Tabla 1.0 Constantes de ajuste PID método a lazo abierto o ganancia límite de Ziegler-Nichols

Controlador	*Constante* k_p	*Constante* T_i	*Constate* T_d
Proporcional	$\frac{\tau}{K\theta}$	∞	0
Proporcional-Integral	$0.9 * \frac{\tau}{K\theta}$	$\frac{\theta}{0.3}$	0
Proporcional-Integral-Derivativo	$1.2 * \frac{\tau}{K\theta}$	2θ	0.5θ

Nota: Tabla obtenida de publicación en Revista Politécnica de Castaño & Gómez 2013

La tabla anterior se aplica solo sí la función de transferencia presenta un comportamiento sigmoidal de primer orden:

$$G(s) = \frac{Y(s)}{U(s)} = \frac{K}{\tau s+1} e^{-\theta s} \qquad (1.5)$$

Donde:

$$\tau = cte.de\ tiempo;\ \theta = retardo;\ \ K = ganancia$$

1.4.7 Control por realimentación de estados en tiempo discreto

La forma dinámica de un sistema físico se puede representar por ecuaciones diferenciales, y si el sistema es invariante en el tiempo, es decir, no cambia su comportamiento ante estados iniciales distintos, se puede representar con ecuaciones diferenciales lineales y con ello en variables de estado. (Jaimes, 2009)

Para un sistema lineal discreto e invariante en el tiempo:

$$x(k+1) = Ax(k) + Bu(k) \qquad (1.6)$$

$$y(k) = Cx(k) + Du(k) \qquad (1.7)$$

En donde:

$$A = Matríz\ de\ estado\ (nxn)$$

$$B = matríz\ o\ vector\ de\ entrada\ \ (nxr)$$

$$C = matríz\ de\ salida\ \ (mxn)$$

$$D = matríz\ de\ transmisión\ directa\ (mxr)$$

La técnica de control por realimentación de estados usa la representación de estados y genera la señal de control que parte de realimentar dichos estados del sistema.

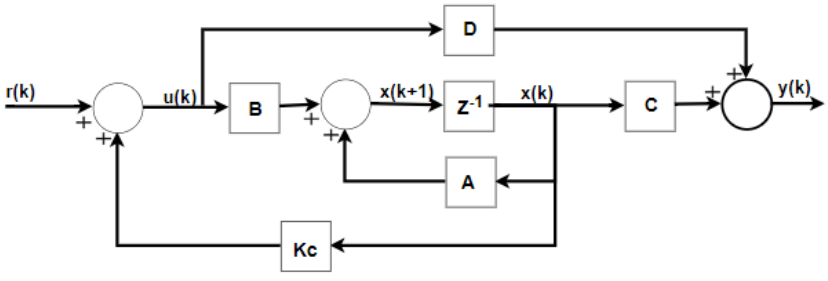


Figura 3. Diagrama en bloques de un sistema con realimentación de estados.

Las características que debe tener en cuenta en un diseño de este tipo son:

- Controlabilidad: solo si A y B son controlables en lazo abierto.
- Observabilidad: va a depender de la realimentación de estados del sistema.

1.4.8 Sistemas de control por lógica difusa

Son sistemas basados en la forma en la que el ser humano percibe el mundo, es decir es un razonamiento aproximado, enfocado en el comportamiento de una variable en un posible conjunto de pertenencia, donde se usan reglas de lógica clásica tipo "si-entonces". Así, por ejemplo, el factor calor puede ser muy caliente, medio caliente, caliente, etc. Su ventaja radica en que no es necesario conocer la función de transferencia del proceso a controlar y es un control empírico mucho más sencillo. (Theler, 2007)

Se definen los siguientes términos asociados:

- **Universo del discurso:** es el conjunto referencial que encierra todos los posibles valores que toma una variable.
- **Conjunto difuso:** es un conjunto cuyos elementos pertenecen a él parcialmente con un rango de certeza.
- **Variables lingüísticas:** son variables que permiten interpretar valores numéricos de un conjunto difuso de acuerdo con la interpretación del propio diseñador del control.
- **Función de membresía:** son funciones que asignan un valor entre 0 y 1, que indica en cuanto puede pertenecer un conjunto difuso.

El mecanismo de inferencia más usado es el propuesto por Ebrahim Mamdani y según (Samuel Diciembre Sanahuja, 2017) consta de 4 pasos:

- **Fusificación:** esta etapa convierte valores numéricos de las entradas en variables lingüísticas.
- **Valoración de reglas:** consiste en establecer reglas "si-entonces" a las variables lingüísticas, de acuerdo con el operador del sistema de control.

- **Agregación:** permite unificar las salidas de cada regla, juntando las funciones de membresía escaladas y recortadas.
- **Defusificación:** convierte los resultados obtenidos por agregación en valores numéricos en términos del conjunto de salida a controlar.

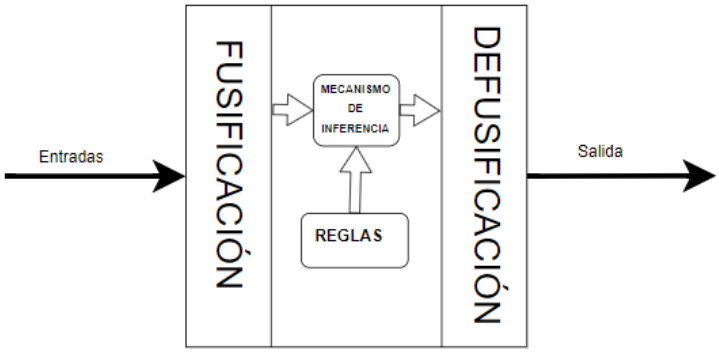


Figura 4. Etapas del control Fuzzy-Mamdani.

Para profundizar en esto véase los trabajos realizados por (Castaño Giraldo, Hernández Gómez, & Gallo Blandón, 2013) y (S. Amuthameena & S. Monisa,2017).

1.4.9 Tarjeta Arduino

De acuerdo con (Arduino, 2023) “Arduino es una plataforma de electrónica de código abierto basada en hardware y software fácil de usar”. Esta plataforma contiene una gama de aplicaciones en su sitio web, entre sus tarjetas de desarrollo se encuentran Arduino Uno, Mega, Leonardo, nano, etc. Arduino Uno se basa en microcontroladores ATmega328. Sus bajos costos y durabilidad los ha convertido en dispositivos emergentes usados en proyectos aplicados en ecosistemas IoT y actualmente se continua su estudio y desarrollo de aplicaciones para entornos industriales.

Figura 5. Tarjeta de desarrollo Arduino Uno.

La herramienta de Arduino para programación es el entorno de desarrollo integrado (IDE), el cual permite lenguaje simplificado de C/C++ y ensamblador. Su ventaja radica en que existe una amplia variedad de librerías especializadas que facilitan su programación y uso para diversas aplicaciones.

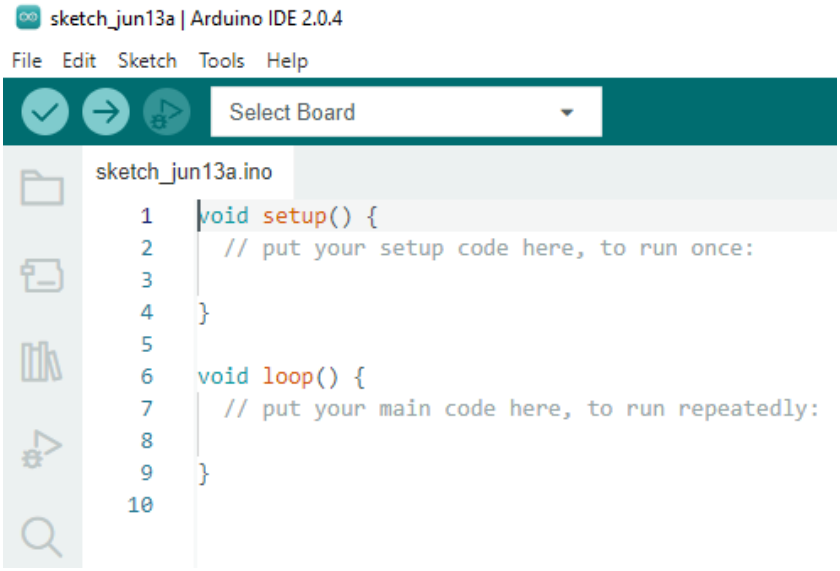


Figura 6. Entorno de Desarrollo Integrado de Arduino (IDE).

Un dispositivo Arduino Uno contiene las siguientes características de acuerdo con (Banzi & Shiloh, 2014) y su datasheet:

Tabla 2.0 Características principales de una tarjeta Arduino Uno R3.

Arduino Uno

Característica	**Especificación**
Microcontrolador	ATmega328p
Entradas/salidas digitales	14 (pines 0-13)
Entradas analógicas	6 (pines A0-A5)
Salidas PWM	6 (pines 3, 5, 6, 9, 10 y 11)
Alimentación externa	5 a 9 V máx.
Alimentación USB	2.7V a 5.5V
Botoneras	Reset
Comunicación	Serial o I2C
CPU	AVR a 16MHz
Memoria Flash	32 KB (0.5KB usada para bootloader)
Memoria SRAM	2 KB
Memoria EEPROM	1 KB

Corriente DC de pin 3.3V	50 mA
Corriente DC de cada pin I/O	20mA
Otros	Contadores/temporizadores internos

Nota: Datos tomados de Datasheet de página oficial Arduino Uno R3.

Algunas librerías especializadas que serán usadas son:

- **OneWire:** usada para permitir comunicación con sensores DS18B20.
- **DallasTemperature:** contiene funciones para trabajar con sensor DS18B20.
- **Arduino:** biblioteca principal que incluye funciones para interrupciones.
- **ArduinoMath:** presenta más funciones matemáticas para Arduino.
- **Esp32TimerInterrupt:** Librería para trabajar con interrupciones en ESP32.
- **Ticker:** permite trabajar con temporizadores de software.
- **TimerOne:** permite trabajar con interrupciones o temporizadores en varias placas de Arduino.
- **BasicLinearAlgebra:** permite realizar operaciones matriciales de algebra lineal.
- **Adafruit SSD1306:** permite utilizar una pantalla OLED.
- **Adafruit GFX Library:** permite manipular gráficas en pantallas.

1.4.10 Tarjeta ESP32

Las tarjetas de desarrollo ESP32 provienen de la compañía multinacional Espressif Systems establecida en 2008, esta ofrece soluciones de código abierto robustas y una serie de tecnologías similares para comunicación inalámbrica, bajo consumo e IoT como las placas ESP32-S, ESP8266, ESP32-C, etc. (Espressif Systems, 2023)

Su ventaja es que este dispositivo presenta características de conexión wifi, bluetooth con un costo inclusive menor al de un Arduino Uno y sobre todo se puede programar mediante el IDE de Arduino mediante librerías.

Tabla 3.0 Características y especificaciones de la ESP32-WROOM-32D.

CARACTERÍSTICA	*DETALLE*
NÚCLEO CORE	ESP32-D0WD dual-core Xtensa 32bit
SPI FLASH	32Mbits, 3.3V
CRISTAL	40MHz
ANTENA	Pin sobre la placa PCB
WIFI	Protocolos 802.11 b/g/n A-MPDU y A-MSDU
BLUETOOTH	Bluetooth V4.2 BR/EDR y LE. Radio NZIF, sensibilidad de -797dBm
MEMORIA FLASH	8 MB
MEMORIA ROM	448 KB

MEMORIA SRAM	520 KB

Nota: Obtenido de Datasheet ESP32-WROOM-32D de sitio oficial Espressif.

1.4.11 Software Matlab/Simulink.

Matlab es software de programación numérica de MathWorks, fundada en 1984 por Jack Little y Cleve Moler. Es una herramienta muy usada en investigaciones y simulaciones científicas. Simulink es una herramienta integrada en Matlab que permite simular procesos numéricos o sistemas de control complejos. Además, contiene un amplio conjunto de "Toolbox" como el "Fuzzy Logic Toolbox" que permiten simular controladores Fuzzy-Mamdani (MathWorks, 2023).

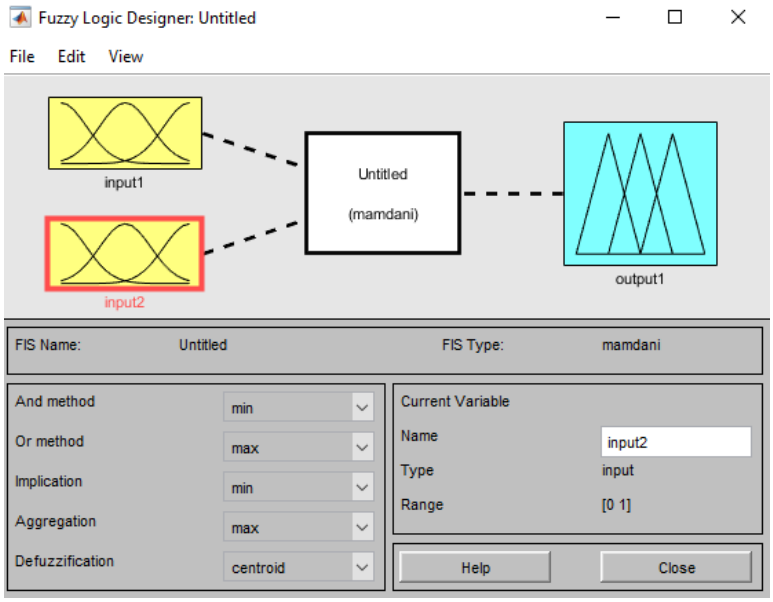


Figura 7. Interfaz de usuario gráfica de Fuzzy Logic Toolbox.

1.4.12 Tarjeta TSC-LAB.

Es un hardware de código abierto, que contiene una planta electrónica para control de temperatura y control de velocidad de un motor DC mediante una tarjeta ESP32. La cual facilita prácticas de adquisición de datos, identificación de sistemas y visualización en plataformas de IoT. (Ojeda Carrera, y otros, 2021)

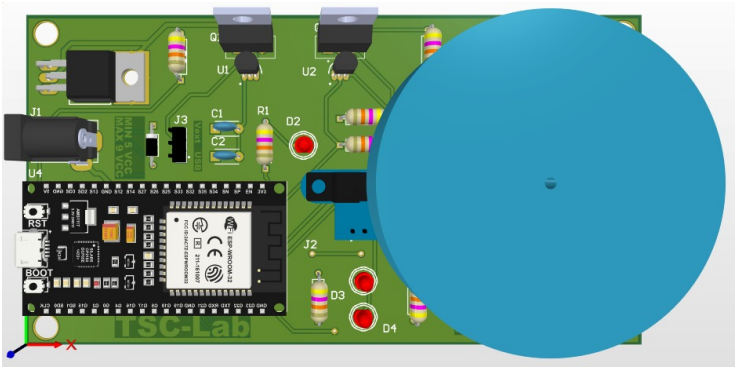


Figura 8. Laboratorio de control de temperatura y velocidad (TSC-LAB).

CAPÍTULO 2

METODOLOGÍA.

En esta sección se describe el proceso que se ha llevado a cabo para dar solución a una problemática que tiene un enfoque educativo dentro de la universidad. En la primera sección se seleccionó los equipos necesarios, en la segunda etapa se buscó desarrollar controladores PID para embeber estos en las tarjetas de desarrollo. Para esta tarea se usó programación en lenguaje simplificado de C/C++ en IDE de Arduino. La siguiente etapa consiste en diseñar controladores por lógica difusa e integrarlos en los microcontroladores. La última etapa de diseño consistió en controladores por realimentación de estados. Además, se realizaron mejoras y diseño del prototipo final del producto entregado, se agregó una pantalla OLED para visualizar los parámetros de interés de control.

2.1 Selección de la tarjeta electrónica para las plantas de control.

La selección de las tarjetas electrónicas para las plantas de control de temperatura y velocidad de motor DC se basó en trabajos o prototipos ya existentes, las cuales son de hardware y software libre, como es con la tarjeta TSC-LAB, la cual se referencia adecuadamente en el capítulo 1. La tarjeta electrónica desarrollada posee ciertas características:

- Entrada de alimentación externa de 5VDC.
- Planta de temperatura basada en transistores TIP31.
- Sensores de temperatura DS18B20.
- Planta de control de velocidad de motor DC con Driver L293D.
- Sensor óptico de una salida lógica.
- Tarjeta ESP32 Devkitc-U4.

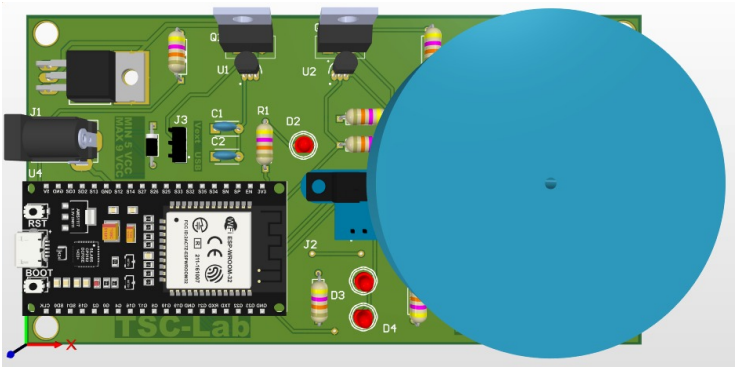


Figura 9. Tarjeta TSC-LAB para control de temperatura y velocidad de motor DC.

Un aspecto importante para considerar es que se puede usar cualquier otra planta electrónica que permita controlar temperatura o velocidad de motor DC, sin embargo, esta adquisición permite tener ambas plantas en una sola PCB de manera práctica y muy didáctica para el ámbito educativo ya que permitiría relacionar estudios anteriores con los propuestos en este trabajo.

2.2 Selección de tarjetas de desarrollo para embeber controladores.

Teniendo la premisa de la tarjeta TSC-LAB, se selecciona la tarjeta de desarrollo ESP32 con un microcontrolador Xtensa LX6. En su contraparte se selecciona una tarjeta de Arduino Uno basado en un microcontrolador ATMEGA328P. Ambas tarjetas se presentan en el mercado con costos accesibles y con un buen rendimiento.

Figura 10. Tarjetas de desarrollo Arduino Uno y ESP32.

2.3 Desarrollo de controladores.

2.3.1 Aproximación del modelo matemático de planta de temperatura.

Se puede demostrar que la planta de temperatura electrónica basada en un transistor BJT tiene un comportamiento de primer orden, de acuerdo con:

$$\frac{\Delta Q}{\Delta t} = Q_{in} - Q_{out} \tag{2.1}$$

$$Q_{in} = \alpha Q_i \rightarrow Calor\ generado\ por\ corriente\ de\ entrada$$

$\alpha \rightarrow$ Relaciona la salida del calentador con la potencia del disipador.

$$Q_{out} = k_T(T - T_A) + \epsilon\sigma A(T^4 - {T_A}^4) \tag{2.2}$$

Donde:

$k_T(T - T_A) \rightarrow$ Corresponde a la ley de enfriamiento de Newton, proceso de convección.

$\epsilon\sigma A\left(T^4 - {T_A}^4\right) \rightarrow$ Corresponde a ley de Stefan-Boltzmann, proceso de radiación.

$k_T \rightarrow$ Constante de pérdida de calor para el ambiente.

$T \rightarrow$ Temperatura del transistor TIP31

$T_A \rightarrow$ Temperatura del ambiente.

$\epsilon \rightarrow$ emisividad

$\sigma \rightarrow$ constante de Boltzman

A es el área de calentamiento del transistor.

$$\frac{\Delta Q}{\Delta t} = \alpha Q_i - k_T(T - T_A) - \epsilon\sigma A(T^4 - {T_A}^4) \quad (2.3)$$

$$k_T = UA \rightarrow U = coef.transf.calor, A = área\ de\ superficie$$

$$\frac{\Delta Q}{\Delta t} = \alpha Q_i + UA(T_A - T) + \epsilon\sigma A({T_A}^4 - T^4) \quad (2.4)$$

$$Q = mc_p(T - T_{ref}) \rightarrow Calor\ absorbido$$

$$mc_p\frac{dT}{dt} = \alpha Q_i - UA(T - T_A) - \epsilon\sigma A(T^4 - {T_A}^4) \quad (2.5)$$

La ecuación anterior corresponde al modelo matemático sin linealizar de un transistor BJT. Linealizando e igualando al punto de equilibrio se obtiene:

$$\frac{d\Delta T}{dt} = \left(-\frac{UA}{mc_p} - \frac{4\epsilon\sigma A}{mc_p}{T_s}^3\right)\Delta T + \left(\frac{\alpha}{mc_p}\right)\Delta Q \quad (2.6)$$

Finalmente, usando la transformada de Laplace tenemos:

$$\frac{\Delta T(s)}{\Delta Q(s)} = \frac{\left(\frac{\alpha}{mc_p}\right)}{s + \left(\frac{UA}{mc_p} + \frac{4\epsilon\sigma A}{mc_p}{T_s}^3\right)} \quad (2.7)$$

Donde Ts es la raíz real y positiva obtenida en la evaluación del punto de equilibrio. El resultado obtenido demuestra que el sistema de control de temperatura con un transistor BJT puede ser modelado por un sistema de primer orden. Adicionalmente, de forma experimental, se mide el tiempo de retardo (L) que representa la inercia del sistema a un cambio, esto hace que el sistema sea mucho más aproximado al real.

$$\frac{\Delta T(s)}{\Delta Q(s)} = \frac{\left(\frac{\alpha}{mc_p}\right)}{s + \left(\frac{UA}{mc_p} + \frac{4\epsilon\sigma A}{mc_p}{T_s}^3\right)} e^{-Ls} \quad (2.8)$$

2.3.2 Controlador PID (Proporcional-Integral-Derivativo) discreto.

El diseño del controlador empieza con tener conocimiento del proceso que se va a controlar, en este caso un control de temperatura de un transistor BJT y un control de velocidad de un motor DC. El primero es un proceso lento dado sus propiedades termodinámicas, mientras que el segundo es un proceso mucho más rápido. Ambas

plantas están diseñadas para que la variable de control sea a través de una señal PWM; para la planta de temperatura es el voltaje que se inyecta a la base de dicho transistor, es decir, el actuador está en la base del mismo transistor, mientras que el motor DC se controla con el voltaje inyectado en el pin habilitador del Driver L293D, esto se explica con más detalles en la sección del diseño de la ley de control.

Para describir el comportamiento un sistema de control se usa la función de transferencia, la cual se asume conocida, ya sea por un proceso previo de identificación de sistemas desarrollado en un laboratorio. Recordando que la identificación del sistema no es objetivo este estudio, sin embargo, en este trabajo para la planta de temperatura se realiza una identificación práctica de la función de transferencia mediante la herramienta "Identificación de Sistemas" de Matlab; básicamente se inyecta un porcentaje de señal PWM, traducido a voltaje en la base del transistor, para que este semiconductor se sature y empiece el proceso de disipación de calor por convección y radiación, con ello se registra la temperatura de salida con un sensor DS18B20 como se puede ver en el siguiente esquema en SIMULINK:

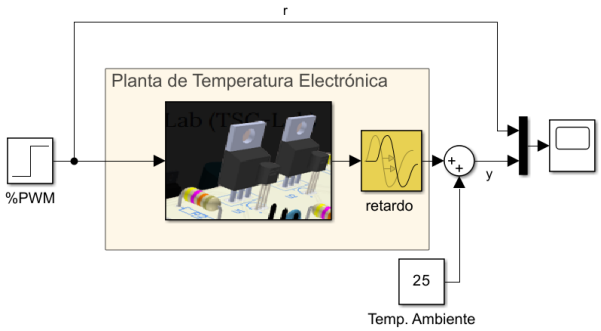


Figura 11. Respuesta del sistema en lazo abierto.

Por otro lado, para la planta de control de velocidad ya existen trabajos anteriores y se conoce la función de transferencia con cierto porcentaje de aproximación. Hay que tener presente que, en este trabajo, el diseño de controladores PID se lo realiza por un método

de sintonización Ziegler-Nichols y por prueba-error, por lo que no se necesita conocer las funciones de transferencia, sin embargo, para otros diseños de controladores como realimentación de estados si son necesarios.

El primer desarrollo del controlador PID discreto se usó en el control de temperatura aplicando el método de sintonización de Ziegler-Nichols en lazo abierto. El proceso es similar para el control por sintonización prueba y error, es decir, se utilizó funciones de programación similares para la ley de control del PID discreto, y lo que permite diferenciar uno del otro es la forma de sintonización o estimación de las constantes proporcional, integral y derivativa a las cuales nos referiremos en adelante como Kp, Ti y Td.

Los sistemas por analizarse deben tener cierto comportamiento dinámico; en la planta de temperatura, se presenta un comportamiento de un sistema de primer orden, en donde tenemos la constante de tiempo, el retardo y la ganancia de acuerdo con la ecuación 2.8. Con estos datos y usando las ecuaciones del método explicado en la tabla 0.1, se determina los parámetros buscados.

La función de transferencia experimentalmente para la planta de temperatura es:

$$G(s) = \frac{0.2259}{116.02s+1} e^{-8.04s} \tag{2.9}$$

Una vez determinada la planta se empieza a desarrollar el diseño del controlador PID, para ello se empieza con la discretización del controlador, el controlador en tiempo continuo tiene la forma:

$$u(t) = k_c e(t) + \frac{k_c}{\tau_i} \int_0^t e(t)dt + k_c \tau_d \frac{de(t)}{dt} \tag{2.10}$$

Se puede usar la aproximación trapezoidal para la componente integral y la aproximación de una derivativa como:

$$\frac{de(t)}{dt} \approx \frac{[e(k)-e(k-1)]}{T_s} \quad (2.11)$$

$$\int_0^t e(t)dt \approx \sum_{h=0}^{k} \left[\frac{e(h)-e(h-1)}{2}\right] T_s \quad (2.12)$$

Luego, en término de muestras tenemos:

$$u(k) = k_c \left(e(k) + \frac{T_s}{2\tau_i}\sum_{h=0}^{k}[e(h) - e(h-1)] + \tau_d \frac{[e(k)-e(k-1)]}{T_s}\right) \quad (2.13)$$

Desarrollando esta ecuación tendremos la siguiente ecuación en diferencias:

$$u(k) = u(k-1) + q_0 e(k) + q_1 e(k-1) + q_2 e(k-2) \quad (2.14)$$

Donde:

$$q_0 = k_c \left(1 + \frac{T_s}{2\tau_i} + \frac{\tau_d}{T_s}\right);\ q_1 = -k_c \left(1 - \frac{T_s}{2\tau_i} + \frac{2\tau_d}{T_s}\right);\ q_2 = k_c \frac{\tau_d}{T_s} \quad (2.15)$$

Estas ecuaciones se describieron en el capítulo 1, es importante tenerlas presente en el diseño de controladores PID dado que son las ecuaciones que permiten que se trabaje en tiempo discreto en un microcontrolador. Hay que tener en cuenta que el tiempo de muestreo es un parámetro importante ya que influye directamente en los valores de los coeficientes de la ecuación en diferencias para determinar la ley de control y el error del sistema.

El siguiente paso consistió en desarrollar código en el IDE de Arduino, en donde se crea una función que básicamente contiene la ecuación 2.14, cuyos parámetros q_0, q_1 y q_2 son variables que son las determinadas anteriormente en el método de Ziegler-Nichols, mientras que en la sintonización prueba-error estas constantes se deben actualizar continuamente durante la ejecución de la función del PID discreto desarrollado. Para cumplir esta tarea se crean los vectores u y e; para la ley de control (**u**) con dos elementos, control actual u(k) y control un tiempo anterior u(k-1), mientras que para el

error (**e**) con tres elementos, dado que el PID discreto depende del error actual e(k), error un tiempo atrasado e(k-1) y el error dos tiempos atrás e(k-2).

La ley de control en el sistema de temperatura y velocidad para este estudio representa el porcentaje de señal PWM (Modulación de ancho de pulso) que se "inyecta", ya sea, en la base del transistor TIP31 o en el pin habilitador del circuito integrado L293D respectivamente. Esta señal PWM se traduce en un voltaje, que depende de dicho porcentaje aplicado, de modo que si el PWM es de 0 % entonces el voltaje aplicado es 0 V y cuando se aplique el 100% de PWM, el voltaje será el máximo que proporcione el microcontrolador. Una tarjeta ESP32 trabaja con 3.3V como máximo en sus pines, mientras que Arduino Uno puede trabajar con 5V o 3.3V en sus pines destinados para este tipo de señales PWM, de esta manera el máximo PWM representa el máximo voltaje que cada microcontrolador puede proporcionar. En Arduino Uno se puede seleccionar que voltaje se usará para este fin a través del pin de referencia análoga o AREF.

El corazón del diseño del controlador PID discreto consiste en actualizar la ley de control **u** y el error **e** cada cierto tiempo conocido como tiempo de muestreo o Tm, con el fin de lograr esto, se usa una función que permita actualizar las posiciones de los vectores, definiendo la última posición de los vectores como el error actual e(k) y la ley de control actual u(k) respectivamente:

$$\boldsymbol{e} = \{e(k-2), e(k-1), e(k)\} \rightarrow e(k)\ es\ el\ error\ actual$$

$$\boldsymbol{u} = \{u(k-1), u(k)\} \rightarrow u(k)\ es\ la\ ley\ de\ control\ actual$$

Una vez sintonizado los parámetros del PID y culminado la programación se puede validar los resultados, el siguiente diagrama ilustra el sistema en lazo cerrado:

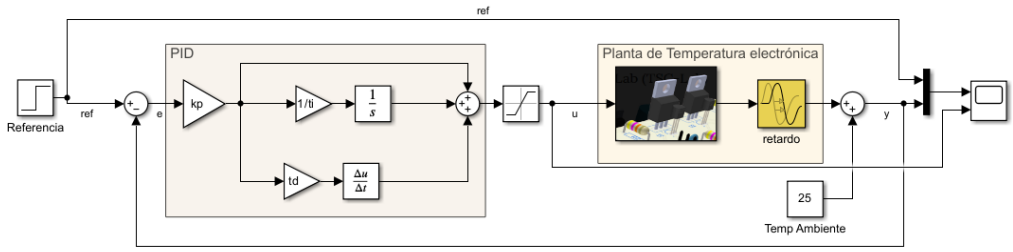


Figura 12. Sistema en lazo cerrado con planta de temperatura.

En la sección de interfaz gráfica se indica la comunicación serial entre la tarjeta TSC-LAB y la interfaz gráfica, esta permitirá validar en tiempo real los resultados del diseño.

2.3.3 Controlador por lógica difusa del tipo Fuzzy-Mamdani.

En el desarrollo de controladores por lógica difusa es muy amplio, para este tipo de control Fuzzy-Mamdani se requiere como entradas la señal del error y la derivada del error. El diagrama de bloques general del sistema es el siguiente:

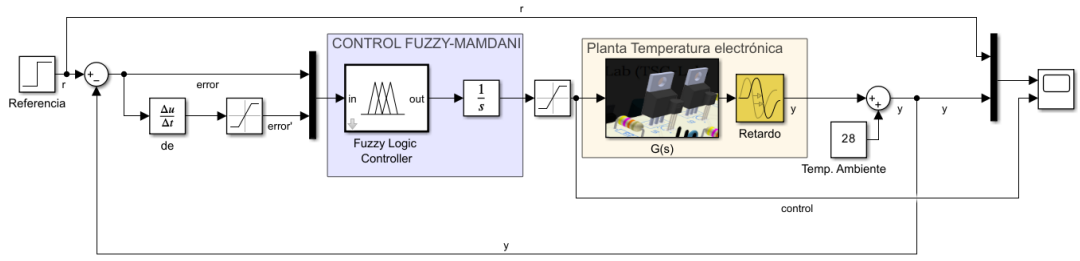


Figura 13. Diagrama de bloques de control Fuzzy-Mamdani de la planta de temperatura.

Partiendo de investigaciones anteriores como las realizadas por (Castaño Giraldo, Hernández Gómez, & Gallo Blandón, 2013), que han trabajado en un desarrollo con lógica difusa aplicado a una planta de temperatura de una industria y usando microcontroladores PIC, este trabajo busca aplicar ese conocimiento y diseñar controladores de este tipo en las plantas electrónicas propuestas, usando las tarjetas ESP32 y Arduino, con el objetivo de que los estudiantes tengan una alternativa y

enfoque a través del cual partir para poder demostrar que estos tipos de controladores pueden ser implementados en estas tarjetas de desarrollo y con ello podrán medir el desempeño de los mismos.

El control difuso, de acuerdo con (Theler, Controladores basados en lógica difusa y loops de convección natural caóticos., 2007) en resumen consiste en 3 etapas claves; Fusificación, Inferencia y Defusificación. Para el desarrollo de la primera etapa es importante definir un vector con dos elementos, un elemento es el error actual e(k), y el otro el error un tiempo pasado e(k-1), y otro vector con la ley de control actual u(k) y un tiempo pasado u(k-1).

$$\boldsymbol{e} = \{e(k-1), e(k)\} \rightarrow e(k)\ es\ el\ error\ actual$$

$$\boldsymbol{u} = \{u(k-1), u(k)\} \rightarrow u(k)\ es\ la\ ley\ de\ control\ actual$$

En el control por lógica difusa se requiere de definir el universo del discurso de las entradas y la ley de control. Estos se definieron de acuerdo con las observaciones de rangos en los que oscilan dichas variables de entradas y salida; es importante medir estos rangos experimentalmente de acuerdo con las limitaciones que se presenta en la planta de control a usarse. En la siguiente sección se explica las etapas para el desarrollo de este controlador con la herramienta "Fuzzy Logix Toolbox" de Matlab.

En la etapa de fusificación se creó 5 variables lingüísticas de acuerdo con un grado de precisión suficiente para la planta de temperatura, estas variables pueden tener cualquier nombre, teniendo en cuenta que mientras más cantidad de variables lingüísticas más procesamiento del microprocesador representará al igual que mayor espacio de memoria. De acuerdo con (Castaño Giraldo, Hernández Gómez, & Gallo Blandón,

2013) se crean variables lingüísticas para las entradas del error, la derivada del error y para la salida o ley de control de la siguiente manera:

Tabla 4.0 Variables lingüísticas de entrada del error.

Entradas del error (e)	*Significado*
Error Cero (eZ)	El error tiene el valor de la referencia
Error Sobre Cero Medio (ePM)	El error es positivo y medio
Error Sobre Cero Alto (ePG)	El error es positivo y alto
Error Bajo Cero Medio (ePM)	El error es negativo y medio
Error Bajo Cero Alto	El error es negativo y alto

Tabla 5.0 Variables lingüísticas de la entrada de la derivada del error.

Entradas de la derivada del error (de)	*Significado*
Derivada del Error Cero (deZ)	El error tiene el valor de la referencia
Derivada del Error Sobre Cero Medio(dePM)	El error es positivo y medio
Derivada del Error Sobre Cero Alto (dePG)	El error es positivo y alto
Derivada del Error Bajo Cero Medio (deNM)	El error es negativo y medio
Derivada del Error Bajo Cero Alto (deNG)	El error es negativo y alto

Tabla 6.0 Variables lingüísticas de la salida de la ley de control.

Salida de ley de control (u)	*Significado*
Salida Cero (uFZ)	La salida debe ser cero
Salida Sobre Cero Media (uFPM)	La salida es positiva y media
Salida Sobre Cero Alta (uFPG)	La salida es positiva y alta
Salida Bajo Cero Media (uFNM)	La salida es negativa y media
Salida Bajo Cero Alta (uFNG)	La salida es negativa y alta

Luego, se selecciona el tipo de función de membresía a usarse y se ajusta para que se encuentre de forma simétrica, esta se basa en la siguiente función triangular:

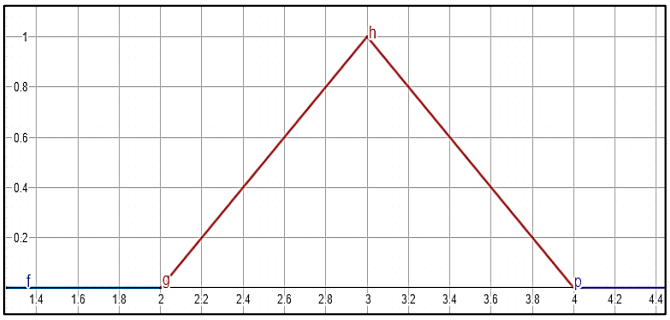


Figura 14. Función de membresía tipo triangular usada en controlador Fuzzy-Mamdani.

La función de membresía de la ilustración puede ser descrita por la siguiente ecuación:

$$f(x,g,h,p) = \max\left(\min\left(\frac{x-g}{h-g}, \frac{p-x}{p-h}\right), 0\right) \qquad (2.16)$$

A continuación, se puede visualizar las funciones de membresía creadas en Matlab:

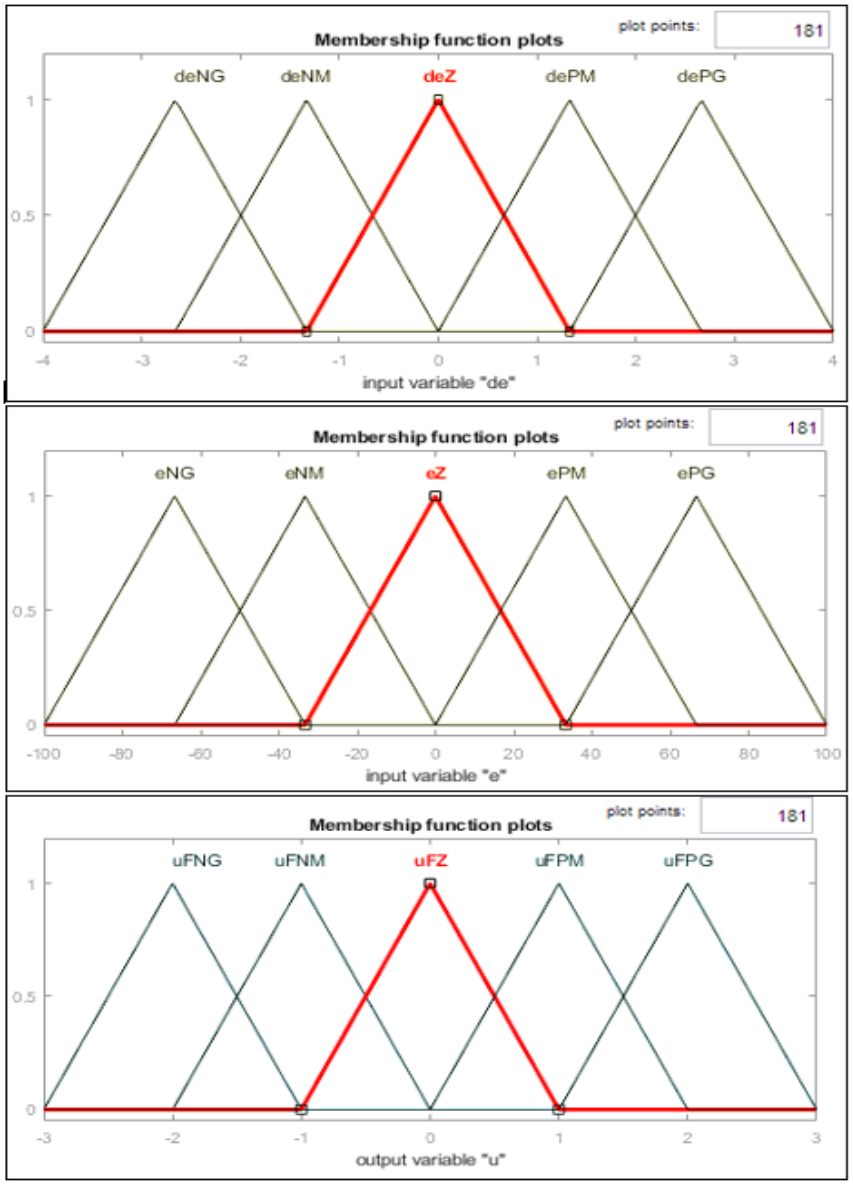


Figura 15. Funciones de membresía de las entradas (error y derivada del error) y la salida (u o ley de control) en Fuzzy Logix Toolbox de planta de temperatura.

En la segunda etapa de inferencia, se arma una tabla en la cual las filas se coloca las variables lingüísticas del error y en las columnas las de derivada del error, en este caso la tabla resulta de 5x5, con lo cual se tiene 25 reglas de inferencia.

e/de	deNG	deNM	deZ	dePM	dePG
eNG	uFNG	uFNG	uFNG	uFNM	uFZ
eNM	uFNG	uFNG	uFNM	uFZ	uFPM
eZ	uFNG	uFNM	uFZ	uFPM	uFPG
ePM	uFNM	uFZ	uFPM	uFPG	uFPG
ePG	uFZ	uFPM	uFPG	uFPG	uFPG

Figura 16. Tabla con reglas de inferencia para planta de temperatura.

De acuerdo con la tabla anterior se debe colocar las reglas de inferencia en la herramienta del Fuzzy de Matlab, estas son evaluadas con condiciones IF-THEN.

En la etapa de defusificación consta de tres fases, la primera es el método de agregación por el máximo, eso quiere decir que, una vez evaluadas cada regla de inferencia se debe tomar el máximo de ellas para cada salida; para ello en la programación se debe inicializar en cero unas variables auxiliares que permitirán realizar el método de agregación a través de la función MAX. La siguiente fase consiste en saturar las salidas en lo valores máximos que se han hallado. En etapa final se hace un cálculo del centro de gravedad en base a los polígonos que se forman en la etapa anterior, y con ello se determina el valor de la salida del control del Fuzzy.

Hay que mencionar que en la programación cíclica se debe siempre actualizar los vectores del error y la ley de control en cada tiempo de muestreo asignado, es decir, cada Tm se halla el error actual, se lo satura de acuerdo con su universo del discurso, y se determina la derivada para saturarla de igual forma. Luego, se evalúa la función principal del Fuzzy que contiene las tres etapas mencionadas anteriormente, y finalmente se halla la ley de control actual, esta debe ser saturada de 0 a 100. Una vez hallados estos dos valores, se debe hacer uso de una función de actualización de los vectores del error y ley de control; esta función ya fue usada anteriormente en el control PID.

El método descrito hasta ahora ha sido para la planta de temperatura, sin embargo, esto puede ser aplicado a cualquier otra planta de un proceso. Con esta aclaración, podemos realizar un control Fuzzy-Mamdani para la planta de control de velocidad de un motor DC.

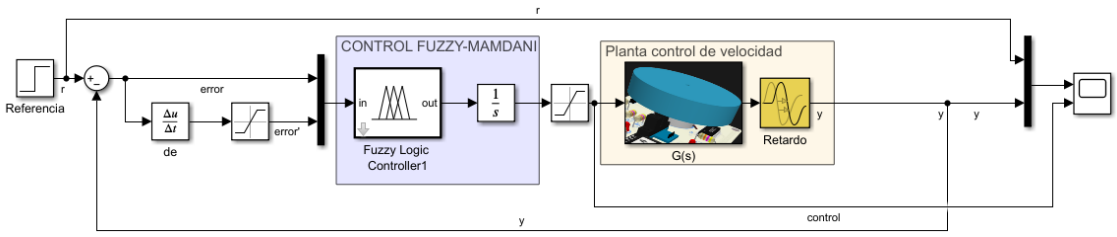


Figura 17. Diagrama de bloques de control Fuzzy-Mamdani de la planta de control de velocidad.

Los cambios respecto al controlador anterior son importantes, dado que tenemos un comportamiento dinámico distinto, el control de velocidad es un proceso mucho más rápido. Evidentemente el universo del discurso de las entradas y la salida cambia, ahora la referencia es insertada en revoluciones por minuto (RPM), y la señal de salida está en la misma unidad. Los ajustes adicionales radican en que para este control se ha añadido 7 variables lingüísticas para que el control sea mucho más potente.

A continuación, se muestra las funciones de membresía para esta planta de control de velocidad:

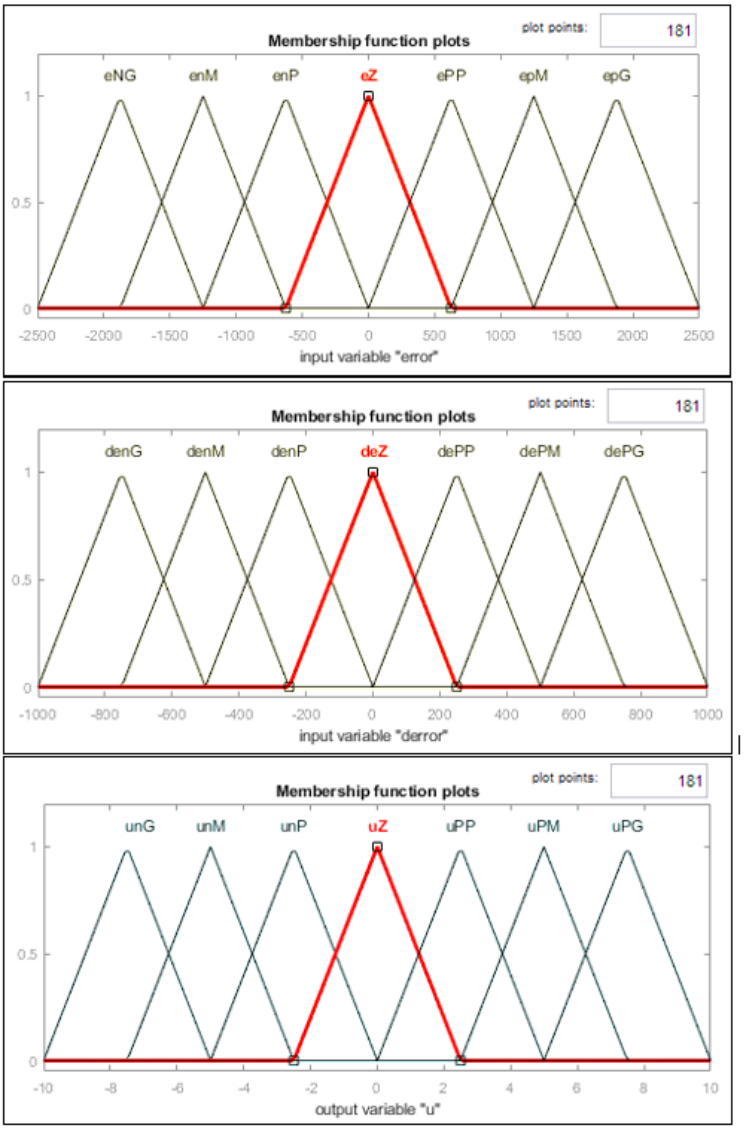


Figura 18. Funciones de membresía de las entradas (error y derivada del error) y la salida (ley de control u) para la planta de control de velocidad.

Las reglas de inferencia en cambio se establecen de la siguiente manera:

e/de	deNG	deNM	deNP	deZ	dePP	dePM	dePG
eNG	uFNG	uFNG	uFNG	uFNG	uFNM	uFNP	uFZ
eNM	uFNG	uFNG	uFNG	uFNM	uFNP	uFZ	uFPP
enP	uFNG	uFNG	uFNM	uFNP	uFZ	uFPP	uFPM
eZ	uFNG	uFNM	uFNP	uFZ	uFPP	uFPM	uFPG
ePP	uFNM	uFNP	uFZ	uFPP	uFPM	uFPG	uFPG
ePM	uFNP	uFZ	uFPP	uFPM	uFPG	uFPG	uFPG
ePG	uFZ	uFPP	uFPM	Ufpg	uFPG	uFPG	uFPG

Figura 19. Reglas de inferencia para planta de control de velocidad.

En este caso, el método descrito anteriormente para la planta de temperatura es similar, es por ello por lo que se tiene una función que realiza las etapas de control de forma similar, lo que se varía son las reglas de inferencia y universo del discurso. Hay que considerar que todas las modificaciones del diseño también se hacen de forma experimental y depende de la experiencia del diseñador respecto del proceso de control, de forma que la ley de control puede mostrar valores muy pequeños que es necesario multiplicar por un factor para que se consiga un mejor desempeño del control.

2.3.4 Controlador por realimentación de estados.

En esta sección se trabajará con variables de estado del sistema, se desarrollará un control integral en tiempo discreto. Hay que recordar que el desarrollo matemático involucrado en este diseño no es objetivo de demostración, para ello se puede referirse a la bibliografía, este trabajo usa el desarrollo teórico existente para embeber los controladores en las tarjetas de desarrollo conforme los objetivos planteados.

El control por realimentación de estados desarrollado requiere conocer su función de transferencia, para este caso nos referimos a la ecuación 2.9 de primer orden con retardo. La estructura del controlador discreto en espacio de estados a usarse tiene la siguiente forma:

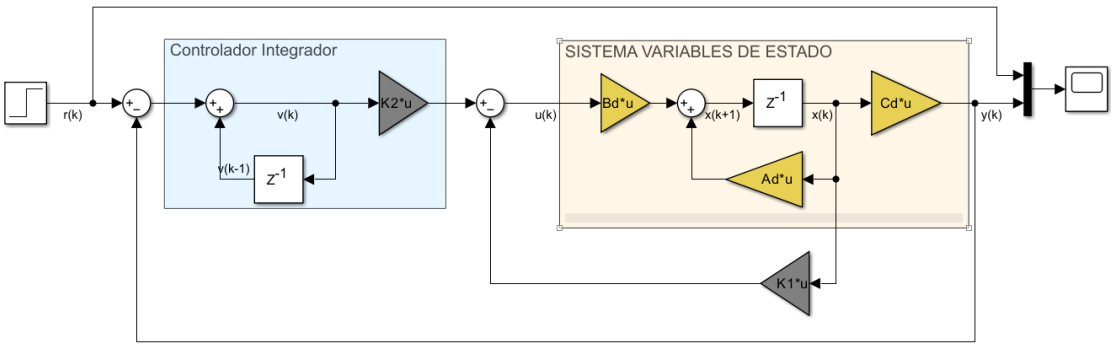


Figura 20. Diagrama de bloques de control por realimentación de estados discreto.

Donde:

$$x(k+1) = Ax(k) + Bx(k) \tag{2.17}$$

$$y(k) = Cx(k) \tag{2.18}$$

$$v(k) = v(k-1) + +r(k) - y(k) \tag{2.19}$$

$$u(k) = -K_1 x(k) + K_2 v(k) \tag{2.20}$$

El sistema en lazo cerrado a implementarse es:

$$\begin{bmatrix} x(k+1) \\ v(k+1) \end{bmatrix} = \begin{bmatrix} A - BK_1 & BK_2 \\ -CA + CBK_1 & I - CBK_2 \end{bmatrix} \begin{bmatrix} x(k) \\ v(k) \end{bmatrix} + \begin{bmatrix} 0 \\ \mathbf{I} \end{bmatrix} r(k+1) \qquad (2.21)$$

El método consiste en usar el software Matlab para desarrollar el diseño del controlador, es decir, se hallan las matrices discretas Ad, Bd, Cd, Dd, los vectores K1 y K2 y estos son inicializados en una función del programa que se ejecutará cada tiempo "Tm" en el microcontrolador. Para esto, primero se establece la función de transferencia del sistema y luego se lleva a espacio de estados con comandos de Matlab "tf2ss" y luego se aplica el comando "c2dt" que permitirá pasar el sistema a tiempo discreto almacenando las matrices discretas Ad, Bd, Cd y Dd, además con este comando es importante asociar el tiempo de muestreo "Tm" de acuerdo con el teorema de Nyquist u otro criterio.

Dado que el sistema que se desarrolla presenta un control integral se debe construir una matriz aumentada que contiene a la planta y el control de la siguiente forma:

$$A_a = \begin{bmatrix} A_d & 0_{nx1} \\ -C_d & 0 \end{bmatrix} \; donde \; n = \dim(A_d) \qquad (2.22)$$

$$B_a = \begin{bmatrix} B_d \\ 0 \end{bmatrix} \qquad (2.23)$$

$$C_a = [C_d \quad 0] \qquad (2.24)$$

$$D_a = [D_d \quad 0] \qquad (2.25)$$

El sistema que se forma es:

$$\begin{bmatrix} x(k+1) \\ \varepsilon(k+1) \end{bmatrix} = \begin{bmatrix} A_d & 0 \\ -C_d & 0 \end{bmatrix} \begin{bmatrix} x(k) \\ \varepsilon(k) \end{bmatrix} + \begin{bmatrix} B_d \\ 0 \end{bmatrix} u(k) + \begin{bmatrix} 0 \\ \mathbf{I} \end{bmatrix} r(k) \qquad (2.26)$$

Donde se ha añadido el estado ε, y se agrega el vector de entrada $E_a = \begin{bmatrix} 0 \\ \mathbf{I} \end{bmatrix}$.

Para el sistema hallado, se debe determinar si este es controlable, para ello se usa la matriz de controlabilidad siguiente:

$$C_0 = \left[B_a \;\; A_a B_a \;\; {A_a}^2 B_a + \cdots + {A_a}^{n-1} B_a\right] \qquad (2.27)$$

El sistema es controlable si la matriz de controlabilidad es de rango completo, por lo que se determina el rango con el comando "rank" en Matlab.

Una vez determinada la controlabilidad se puede hallar la ecuación característica del sistema, es aquí donde se especifican los índices de desempeño que se requieren para el controlador, por ejemplo, el valor del máximo pico, el tiempo de establecimiento, frecuencia natural, factor de amortiguamiento, etc. De acuerdo con lo anterior, se puede hacer una asignación de polos dado el orden de la ecuación característica; en el caso de la planta de temperatura con el sistema aumentado se determinan dos polos dominantes del sistema, estos son dos y se obtienen mediante las ecuaciones:

$$p1 = -2\cos\left(\omega_n T_s \sqrt{1-\delta^2}\right) * e^{-\delta\omega_n T_s} \qquad (2.28)$$

$$p2 = e^{-2\delta\omega_n T_s} \qquad (2.29)$$

Estos polos se pueden obtener por diferentes métodos, en este caso se usa el método para un control por reubicación de polos por seguimiento de referencia, sin embargo, en el sistema de control de temperatura se necesitan asignar 2 polos más por el orden de la ecuación característica hallada, estos son ubicados lo más próximos al eje del plano Z. Luego, usando el comando "place" de Matlab se ubican los polos en lazo cerrado.

Las ganancias del controlador K1 y K2 se hallan de la siguiente forma:

$$[K_1 \; K_2] = \left[K_a + [\mathbf{0} \;\; \mathbf{I}]\right] \begin{bmatrix} A_d - \mathbf{I} & B_d \\ C_d A_d & C_d B_d \end{bmatrix}^{-1} \qquad (2.30)$$

Donde:

$$K_a = [K_1 A_d - K_1 \; + K_2 C_d A_d \quad K_1 B_d + K_2 C_d B_d \; - \mathbf{I}] \qquad (2.31)$$

Recuerde consultar la bibliografía, esta sección se desarrolla de acuerdo con lo descrito en (Ogata, Sistema de control en tiempo discreto 2a edición, 1996).

Una vez realizado el diseño del controlador y determinarse las matrices Ad, Bd, Cd, Dd y las ganancias K1 y K2, se desarrolla la programación en IDE de Arduino. Se crea una función que permita ejecutarse cada tiempo de muestreo “Tm”, en esta función se actualiza el vector de estados, el vector del integrador y la ley de control que permiten que se desarrolle el control en tiempo real cada “Tm”. Es importante inicializar las ganancias K1 y K2 en la configuración SETUP, también se agregan las matrices Ad y Bd halladas en Matlab. La matriz Cd no es necesario inicializar dado que se asocia a variables internas que permitirán el control de la señal PWM o variable controlada.

2.4 Interfaz gráfica en App Designer/Matlab.

Con el objetivo de visualizar en tiempo real el comportamiento de los controladores desarrollados se ha diseñado una interfaz gráfica con la herramienta de Matlab llamada App Designer. Esta es una herramienta de programación gráfica que permite crear aplicaciones de forma sencilla.

Figura 21. Interfaz gráfica en Matlab/App Designer de controladores.

La interfaz permite interactuar con los microcontroladores, se configuró para que permita trabajar con las plantas de control temperatura y control de velocidad. Se puede seleccionar el puerto y la velocidad de transmisión de datos de comunicación serial, y finalmente la tarjeta de control ESP32 o Arduino.

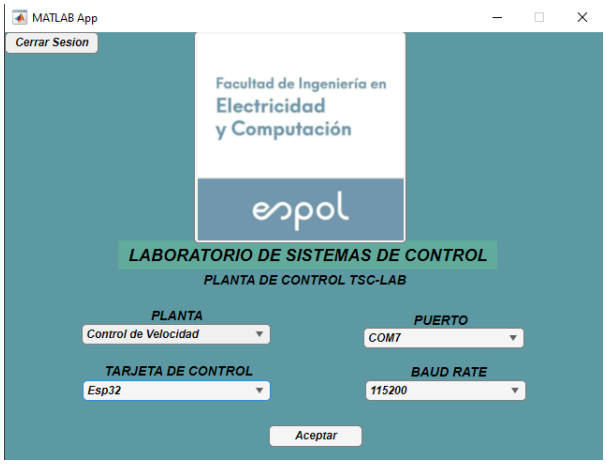


Figura 22. Configuraciones de interfaz y selección de tarjeta de control.

La interfaz permite visualizar en tiempo real la variable controlada, la salida del sistema y, además, se puede ingresar la señal de referencia. En la siguiente figura se muestra la interfaz para el control PID con sintonización prueba error para la planta de temperatura y con la tarjeta ESP32, esta permite sintonizar las contantes Kp, Ti y Td:

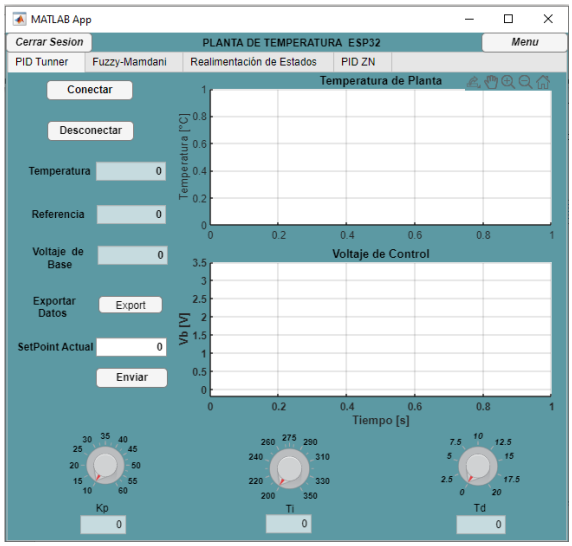


Figura 23. Interfaz con control PID por sintonización prueba error en planta de temperatura.

2.5 Diseño de prototipo físico.

El diseño del prototipo se lo realiza en el software SKETCHUP, esta herramienta permite visualizar los componentes usados en vista 3D, de esta forma se puede realizar estimaciones del prototipo final con medidas precisas.

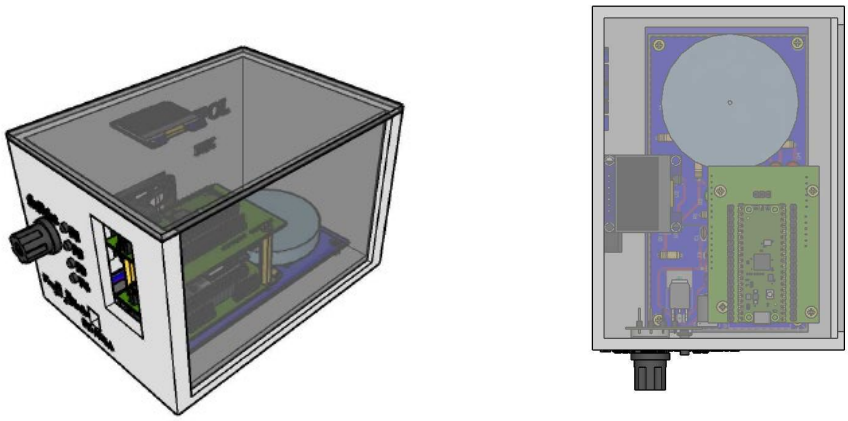


Figura 24. Diseño de prototipo físico final.

El prototipo cuenta con la tarjeta TSC-LAB, ESP32, pantalla OLED, ventilador, 4 botones configurables para usos diversos y un potenciómetro para establecer la referencia. Como adicional se compone de una tarjeta Raspberry Pi Pico para manejar el procesamiento de los botones y permita graficar en la pantalla OLED las señales requeridas. Se ha diseñado PCB de una capa para acoplar las tarjetas y poderse aprovechar las entradas, salidas y comunicación del sistema embebido.

CAPÍTULO 3

RESULTADOS Y ANÁLISIS

3.1 Desempeño de controladores en Planta de Temperatura en ESP32.

En la planta de temperatura basada en ESP32 el voltaje de alimentación en la base del transistor es de 3.3V. Lo que permitió realizar el control de la planta en un rango de 34°C hasta 37°C. A continuación, se presenta las gráficas obtenidas desde la interfaz de usuario diseñada en AppDesigner junto con sus respectivas tablas con sus índices de desempeño por controlador.

3.1.1 Controlador PID sintonización Prueba-Error (PE)

Las constantes de control consideradas en este caso fueron: Kp:19,99; Ti:250,7; Td:0.

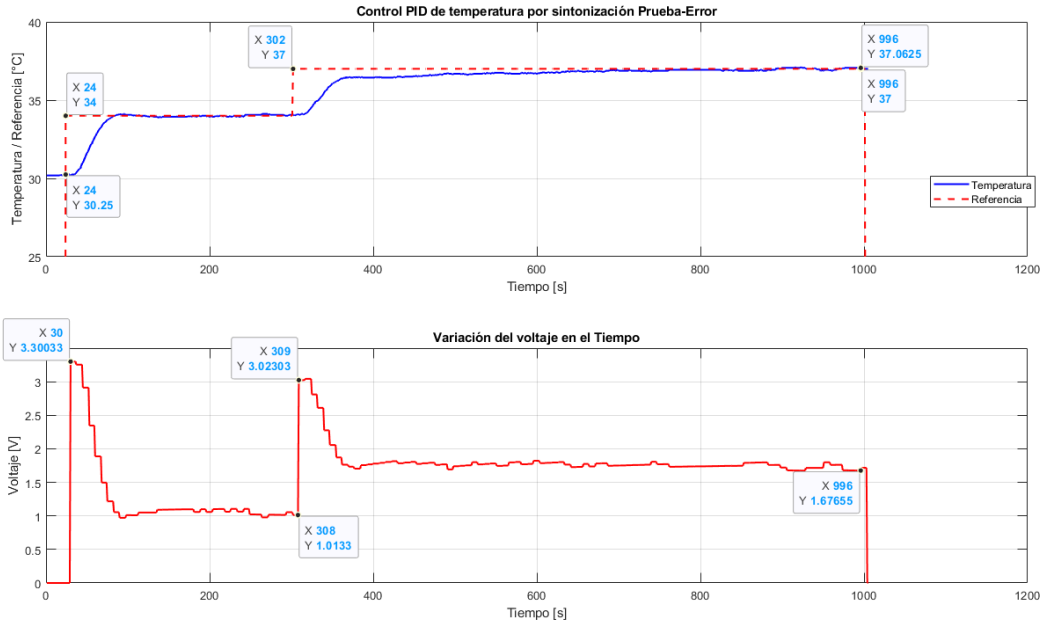


Figura 25. Resultados de control PID por sintonización Prueba-Error para control de temperatura.

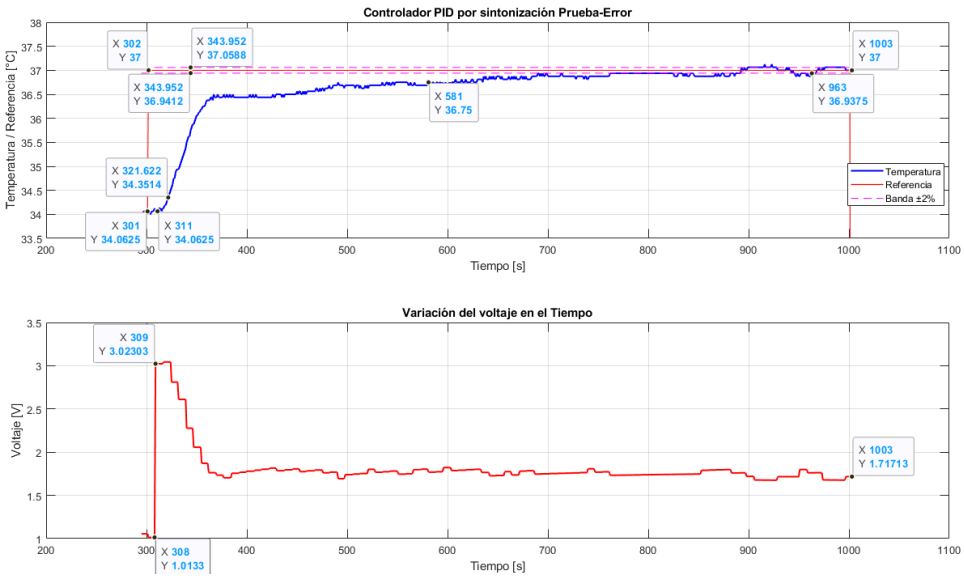


Figura 26. Segundo tramo para análisis de desempeño de controlador.

De las gráficas se tiene los siguientes datos para el segundo tramo:

Tabla 7.0 Datos obtenidos para control PID por sintonización Prueba-Error en ESP32.

Temperatura inicial	$\boldsymbol{T_o}$ $[°\boldsymbol{C}]$	34,06
Tiempo de inserción de setpoint en interfaz gráfica	t_{int} $[s]$	301
Tiempo de establecimiento al 2% de la banda	t_{s1} $[s]$	963
Setpoint	ref $[°C]$	37

Además, los índices de desempeño son los siguientes:

Tiempo muerto:

$$t_{muerto} = 311 - 301 = 10\ [s]$$

El tiempo de establecimiento:

$$t_s = 963 - 301 = 662[s]$$

El tiempo de subida (t_r) que tarda en pasar del 10% al 90% del valor final es:

$$T_{90\%} = 0{,}9(37 - 34.06) + 34.06 = \ 36{,}7\ [°C]$$

$$T_{10\%} = 0{,}1(37 - 34.06) + 34.06 = 34{,}35\ [°C]$$

$$t_r = t_{90\%} - t_{10\%} = 581 - 311 = 270[s]$$

Tabla 8.0 Índices de desempeño del control (PE) en ESP32.

Tiempo de establecimiento en segundos ($\boldsymbol{t_s}$)	662
Tiempo de subida en segundos ($\boldsymbol{t_r}$)	270

3.1.2 Controlador Fuzzy-Mamdani

Para este control se considera 5 variables lingüísticas, la cual permie tener 25 reglas de inferencia.

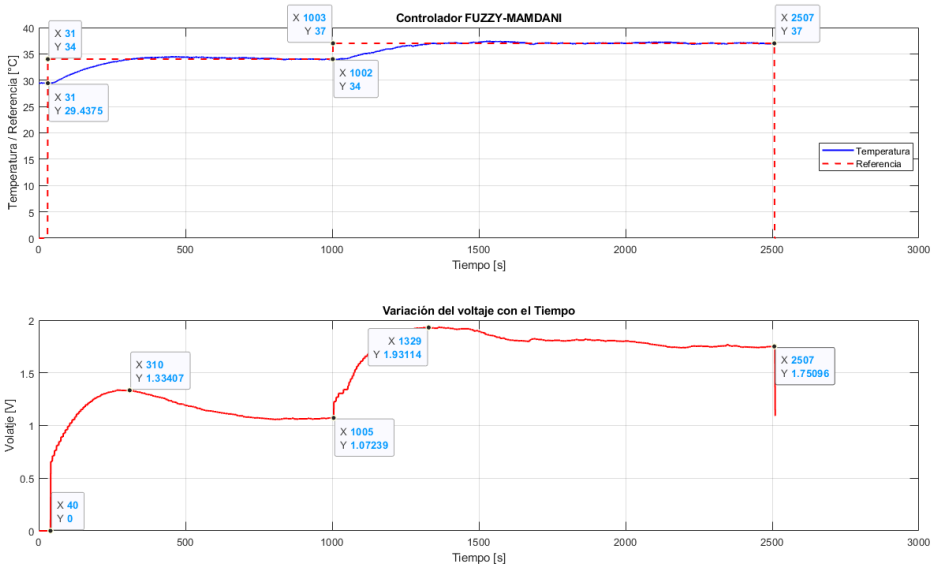


Figura 27. Resultados de control Fuzzy-Mamdani para control de temperatura.

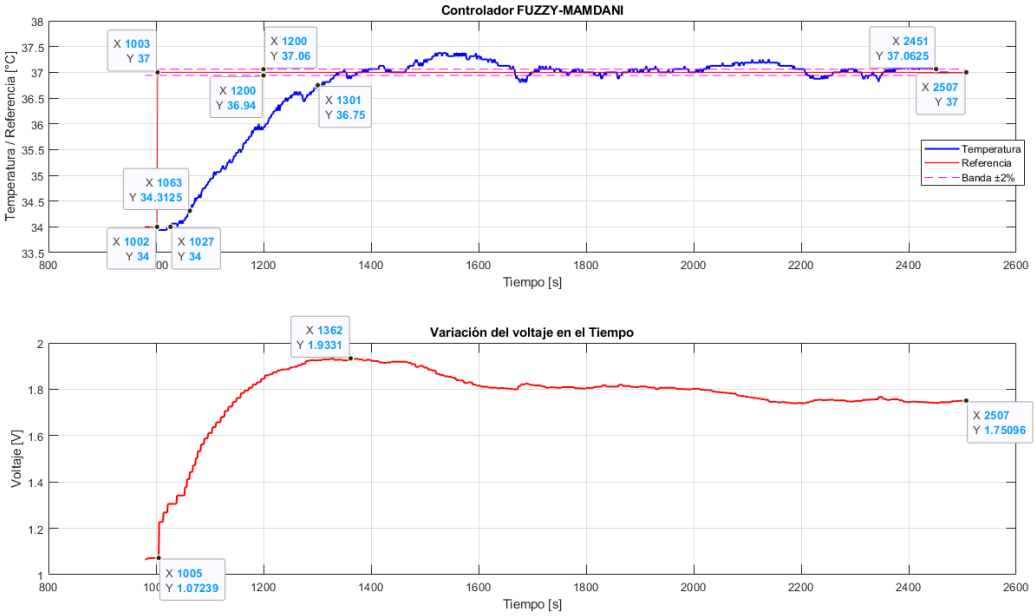


Figura 28. Segundo tramo para análisis de desempeño de controlador.

De las gráficas se tiene los siguientes datos para el segundo tramo:

Tabla 9.0 Datos obtenidos para control Fuzzy-Mamdani en ESP32.

Temperatura inicial	$\boldsymbol{T_o}$ $[°\boldsymbol{C}]$	34,00
Tiempo de inserción de setpoint en interfaz gráfica	t_{int} $[s]$	1002
Tiempo de establecimiento al 2% de la banda	t_{s1} $[s]$	2451
Setpoint	ref $[°C]$	37

Tiempo muerto:

$$t_{muerto} = 1027 - 1002 = 25\ [s]$$

El tiempo de establecimiento de acuerdo con la gráfica es:

$$t_s = 2451 - 1002 = 1449\ [s]$$

El tiempo de subida que tarda en pasar del 10% al 90% del valor final es:

$$T_{90\%} = 0{,}9(37 - 34) + 34 = \ 36{,}7\ [°C]$$

$$T_{10\%} = 0{,}1(37 - 34) + 34 = 34{,}3\ [°C]$$

$$t_r = t_{90\%} - t_{10\%} = 1301 - 1063 = 238[s]$$

Tabla 10.0 Índices de desempeño del control Fuzzy-Mamdani en ESP32.

Tiempo de establecimiento en segundos (t_s)	1449
Tiempo de subida en segundos (t_r)	238

3.1.3 Controlador PID por sintonización Ziegler–Nichols (ZN)

A partir de la respuesta del sistema en lazo abierto se obtuvo las siguientes constantes: Kp:57,49; Ti:351,57; Td:0,0.

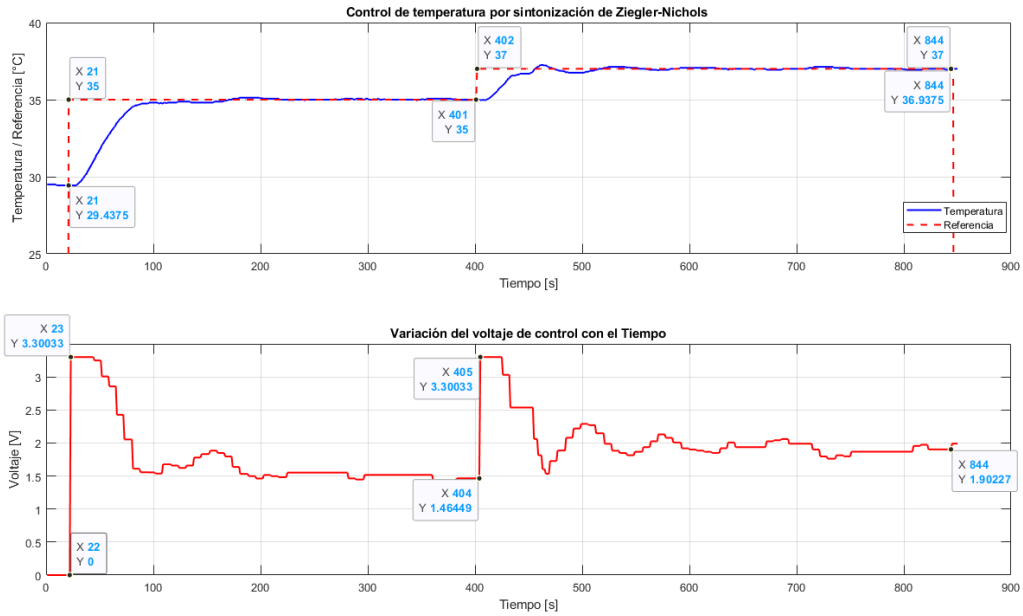


Figura 29. Resultados de control PID por sintonización Ziegler-Nichols para control de temperatura.

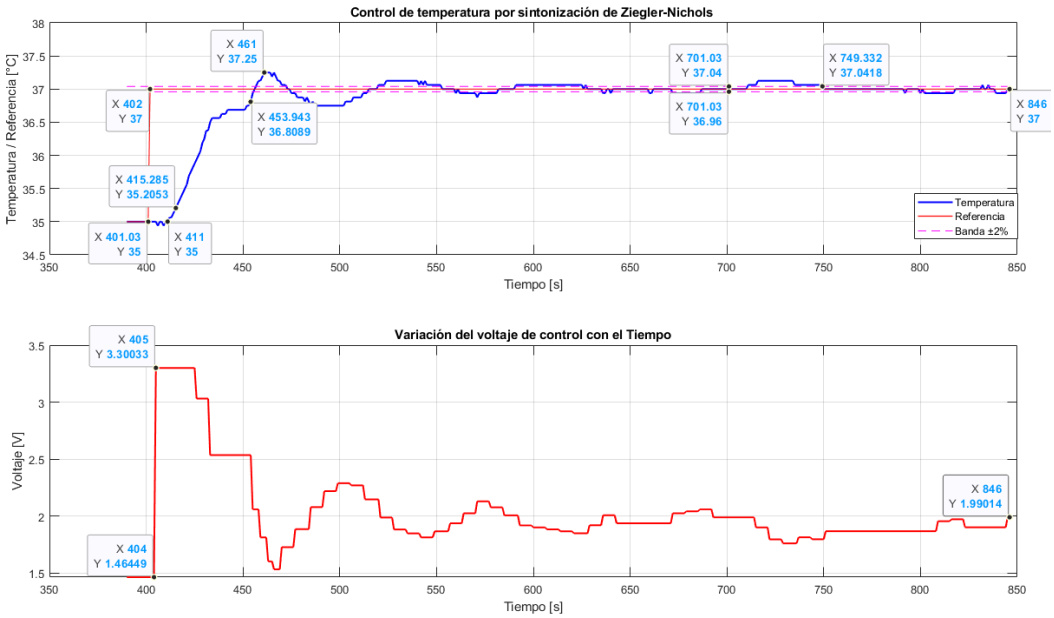


Figura 30. Segundo tramo para análisis de desempeño de controlador.

Tabla 11.0 Datos obtenidos para el control (ZN) en ESP32.

Temperatura inicial	T_o [°C]	35
Tiempo de inserción de setpoint en interfaz gráfica	t_{int} [s]	401
Tiempo de establecimiento alrededor del de la banda del 2%	t_{s1} [s]	794
Setpoint	ref [°C]	37

El tiempo muerto:

$$t_{muerto} = 411 - 401 = 10\ [s]$$

El tiempo de establecimiento de acuerdo con la gráfica es:

$$t_s = 749 - 401 = 348\ [s]$$

El tiempo de subida que tarda en pasar del 10% al 90% del valor final es:

$$T_{90\%} = 0{,}9(37 - 35) + 35 = 36{,}8\ [°C]$$

$$T_{10\%} = 0{,}1(37 - 35) + 35 = 35{,}2\ [°C]$$

$$t_r = t_{90\%} - t_{10\%} = 453 - 415 = 38[s]$$

Tabla 12.0 Índices de desempeño del controlador (ZN) en ESP32.

Tiempo de establecimiento en segundos ($\boldsymbol{t_s}$)	393
Tiempo de subida en segundos ($\boldsymbol{t_r}$)	38
Valor de máximo pico (°C)	37,25

3.1.4 Controlador por Realimentación de Estados (RE)

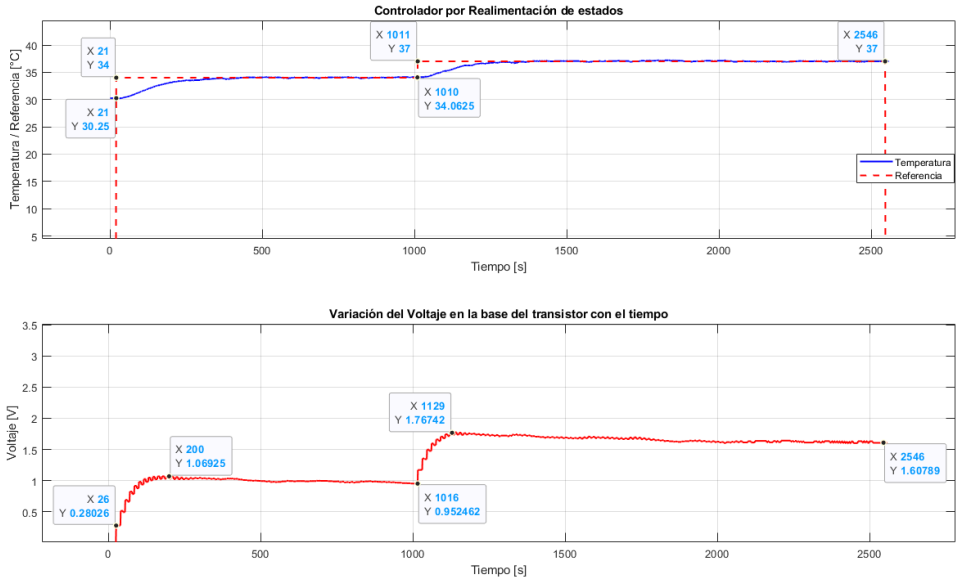


Figura 31. Resultados experimentales obtenidos del control por realimentación de estados para temperatura.

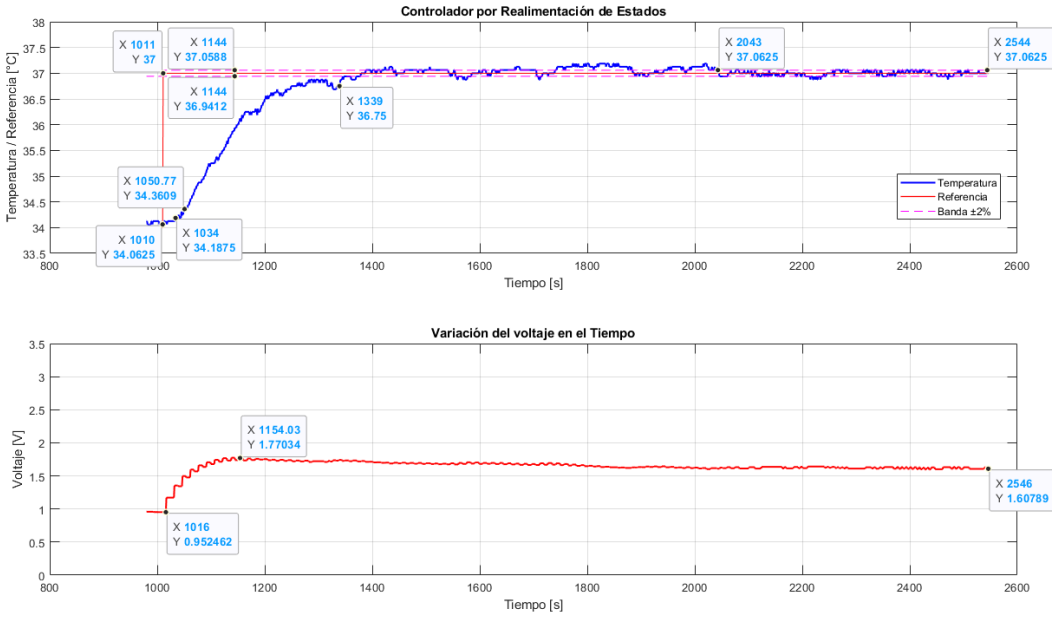


Figura 32. Segundo tramo para análisis de desempeño del controlador.

Tabla 13.0 Datos obtenidos experimentalmente del control (RE) en ESP32.

Temperatura inicial	$\boldsymbol{T_o}$ $[°\boldsymbol{C}]$	34,06
Tiempo de inserción de setpoint en interfaz gráfica	t_{int} $[s]$	1010
Tiempo de establecimiento de la banda del 2%	t_{s1} $[s]$	2043
Setpoint	ref $[°C]$	37

Tiempo muerto:

$$t_{muerto} = 1034 - 1010 = 24\ [s]$$

El tiempo de establecimiento de acuerdo con la gráfica es:

$$t_s = 2043 - 1010 = 1033\ [s]$$

El tiempo de subida que tarda en pasar del 10% al 90% del valor final es:

$$T_{90\%} = 0{,}9(37 - 34) + 34 = 36{,}7[°C]$$

$$T_{10\%} = 0{,}1(37 - 34) + 34 = 34{,}3[°C]$$

$$t_r = t_{90\%} - t_{10\%} = 1339 - 1050 = 289[s]$$

Tabla 14.0 Índices de desempeño del control (RE) en ESP32.

Tiempo de establecimiento en segundos $(\boldsymbol{t_{ss}})$	1033
Tiempo de subida en segundos $(\boldsymbol{t_r})$	289

3.2. Análisis de tiempos de respuesta con ESP32.

De acuerdo con la tabla 24 de los cuatro controladores implementados en la tarjeta ESP32 el controlador con menor tiempo muerto son los controladores por sintonización Prueba-Error y Ziegler-Nichols con un tiempo de 10 segundos, a diferencia de los demás que tardaron hasta 25 segundos en dar una respuesta a la señal de control.

En cuanto al tiempo de establecimiento se tiene que para la sintonización Prueba-Error tardó 662 segundos, mientras que el Zeigler Nichols le tomo un tiempo de 348 segundos. En cambio, al controlador Fuzzy el tiempo fue de 1449 segundos y 1033 el controlador por realimentación de estados.

El tiempo de subida para el control por sintonización Ziegler-Nichols fue el menor con 38 segundos, para el control por Prueba-Error fue de 270 segundos, 289 segundos el control por realimentación de estados, mientras que, el control Fuzzy-Mamdani 238 segundos.

Para todos los controladores se consideró un tiempo de periodo de muestreo de 8 segundos.

3.3. Análisis de señal de control con ESP32.

En cuanto al voltaje en la base del transistor de las gráficas podemos observar que tanto el controlador Fuzzy como en Realimentación de Estados, la señal de control es más suave y no tendrá problemas de ser sobre esforzada. Pero, aunque el control es más suave el tiempo de reacción del sistema incrementa con respecto del controlador Zeigler-Nichols y el Prueba-Error.

Tabla 15.0 Relación de voltaje de los controladores PE, FUZZY, ZN y RE.

Voltaje [V]	Prueba Error (PE)	Fuzzy-Mamdani (FM)	Realimentación de estados (RE)	Ziegler-Nichols (ZN)
Voltaje máximo v_{max}	3,02	1,75	1,77	3,3
Voltaje una vez estabilizado v_{tss}	1,71	1,93	1,60	1,99
Voltaje inicial v_{ini}	1,01	1,07	0,95	1,46

Con respecto a los controladores Zeigler-Nichols, y Prueba-Error los tiempos de establecimiento son menores considerablemente con respecto del control Fuzzy y Realimentación de estados, pero se puede ver que la señal de control puede tener picos que se acercan mucho al voltaje de control máximo, lo cual físicamente provocaría un sobre esfuerzo en el actuador del sistema.

3.4. Desempeño de controladores en Planta de Temperatura con Arduino Uno

En la planta de temperatura basada en Arduino Uno, la salida de voltaje hacia la base del Transistor que representa la planta es de 5Vdc. En este caso el control se realizó en rangos de temperatura ambiente hasta un máximo de 41°C. A continuación, se presenta las gráficas obtenidas desde la interfaz en AppDesigner y la implementación de los controladores en la tarjeta de control.

3.4.1. Controlador PID Sintonización prueba-error (PE)

Para este caso se ha tomado los valores de: Kp:19,99; Ti:250,1; Td:0,0.

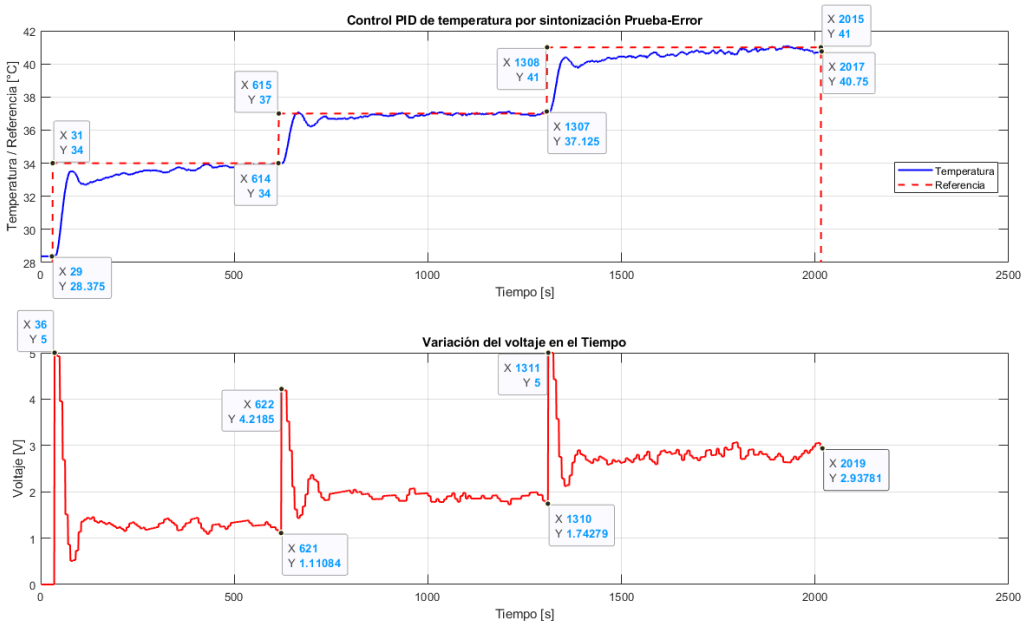


Figura 33. Resultados obtenidos para el control de temperatura PID por sintonización Prueba-Error en Arduino Uno.

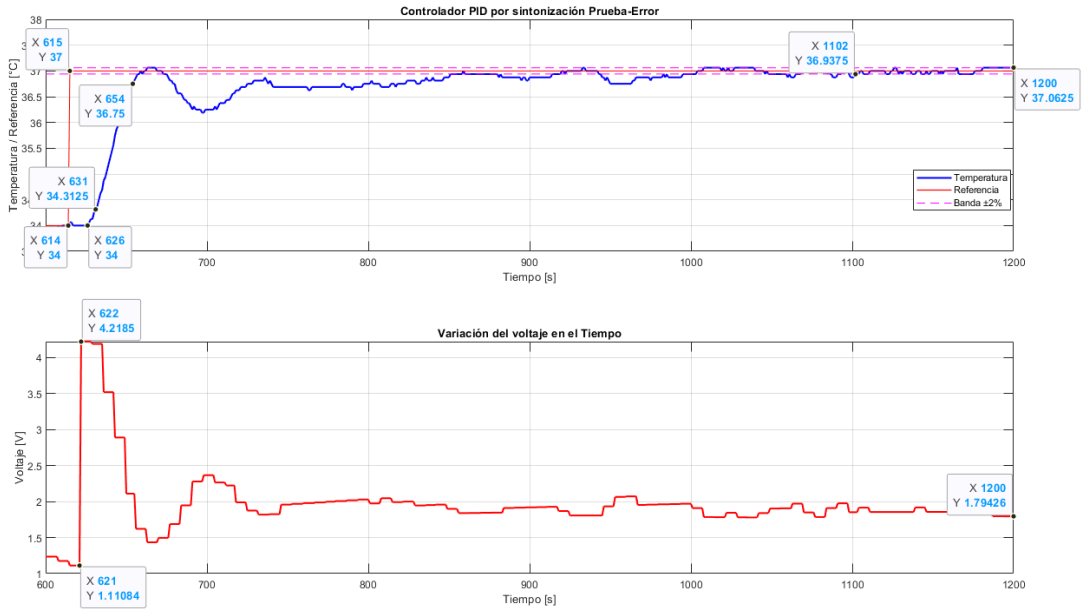


Figura 34. Segundo tramo para análisis de desempeño del control PID en Arduino.

Tabla 16.0 Datos obtenidos experimentalmente para el control PID (PE) en Arduino.

Temperatura inicial	**To** [°C]	34
Tiempo de inserción de setpoint en interfaz gráfica	t_{int} [s]	614
Tiempo de establecimiento en banda del 2%	t_{s1} [s]	1102
Setpoint	ref [°C]	37

Tiempo muerto:

$$t_{muerto} = 626 - 614 = 12[s]$$

El tiempo de establecimiento de acuerdo con la gráfica es:

$$t_s = 1102 - 614 = 488[s]$$

El tiempo de subida que tarda en pasar del 10% al 90% del valor final es:

$$T_{90\%} = 0{,}9(37{,}0625 - 34) + 34 = 36{,}75[°C]$$

$$T_{10\%} = 0{,}1(37{,}0625 - 34) + 34 = 34.3[°C]$$

$$t_r = t_{90\%} - t_{10\%} = 654 - 631 = 23[s]$$

Tabla 17.0 Índices de desempeño para el control PID (PE) en Arduino Uno.

Tiempo de establecimiento en segundos ($\boldsymbol{t_s}$)	488
Tiempo de subida en segundos ($\boldsymbol{t_r}$)	23

3.4.2. Controlador Fuzzy-Mamdani

Al igual que en la planta de temperatura basada en ESP32 en este caso se considera 5 variables lingüísticas, lo que proporciona 25 reglas de inferencia.

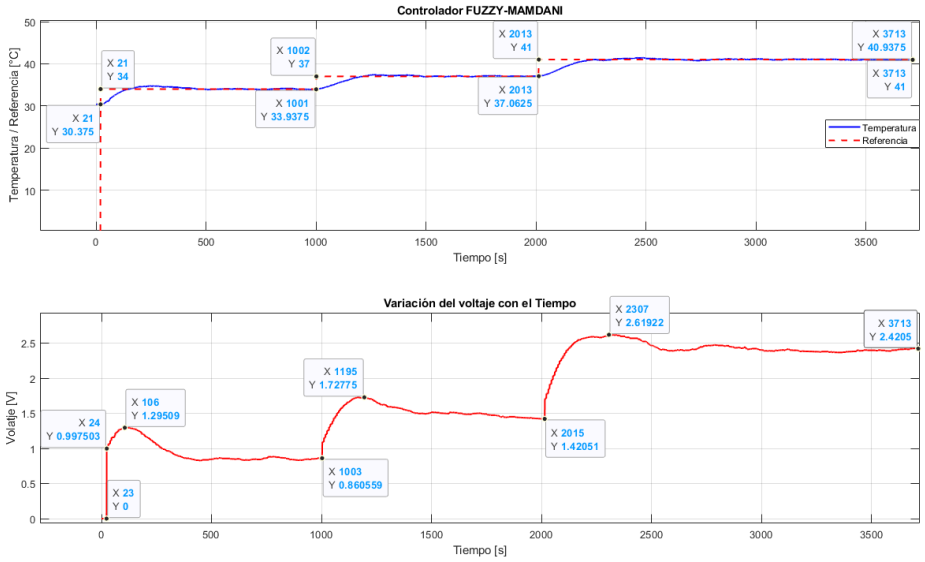


Figura 35. Resultados experimentales del control Fuzzy-Mamdani en Arduino Uno para temperatura.

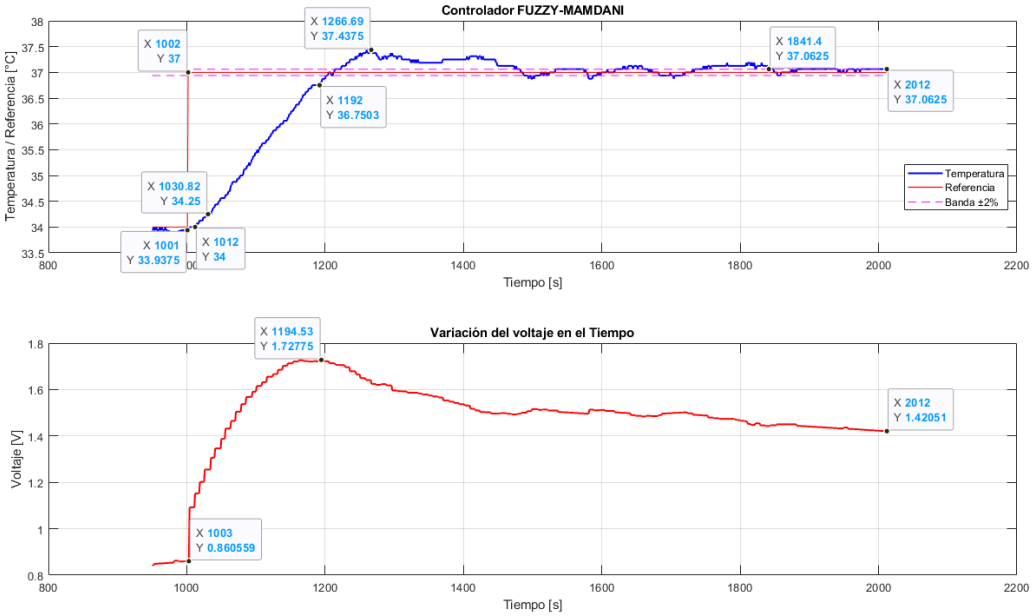


Figura 36. Segundo tramo para análisis de desempeño del control Fuzzy-Mamdani en Arduino Uno.

Tabla 18.0 Datos experimentales del control Fuzzy-Mamdani en Arduino Uno.

Temperatura inicial	$\boldsymbol{To}$ $[°C]$	33,93
Tiempo de inserción de setpoint en interfaz gráfica	t_{int} $[s]$	1001
Tiempo de establecimiento de la banda del 2%	t_{s1} $[s]$	1841
Setpoint	ref $[°C]$	37

Tiempo muerto:

$$t_{muerto} = 1012 - 1001 = 11[s]$$

El tiempo de establecimiento de acuerdo con la gráfica es:

$$t_s = 1841 - 1001 = 840[s]$$

El tiempo de subida que tarda en pasar del 10% al 90% del valor final es:

$$T_{90\%} = 0{,}9(37{,}0625 - 33{,}93) + 33{,}93 = 36{,}74[°C]$$

$$T_{10\%} = 0{,}1(37{,}0625 - 33{,}93) + 33{,}93 = 34.24[°C]$$

$$t_r = t_{90\%} - t_{10\%} = 1192 - 1030 = 162[s]$$

Tabla 19.0 Índices de desempeño para el controlador Fuzzy-Mamdani en Arduino Uno.

Tiempo de establecimiento en segundos (t_s)	840
Tiempo de subida en segundos (t_r)	162

3.4.3. Controlador PID por sintonización Ziegler–Nichols (ZN).

A partir de la respuesta de la sintonización por Ziegler-Nichols con las ecuaciones se obtuvo las siguientes constantes: Kp:38,39; Ti:36,48; Td:0,0.

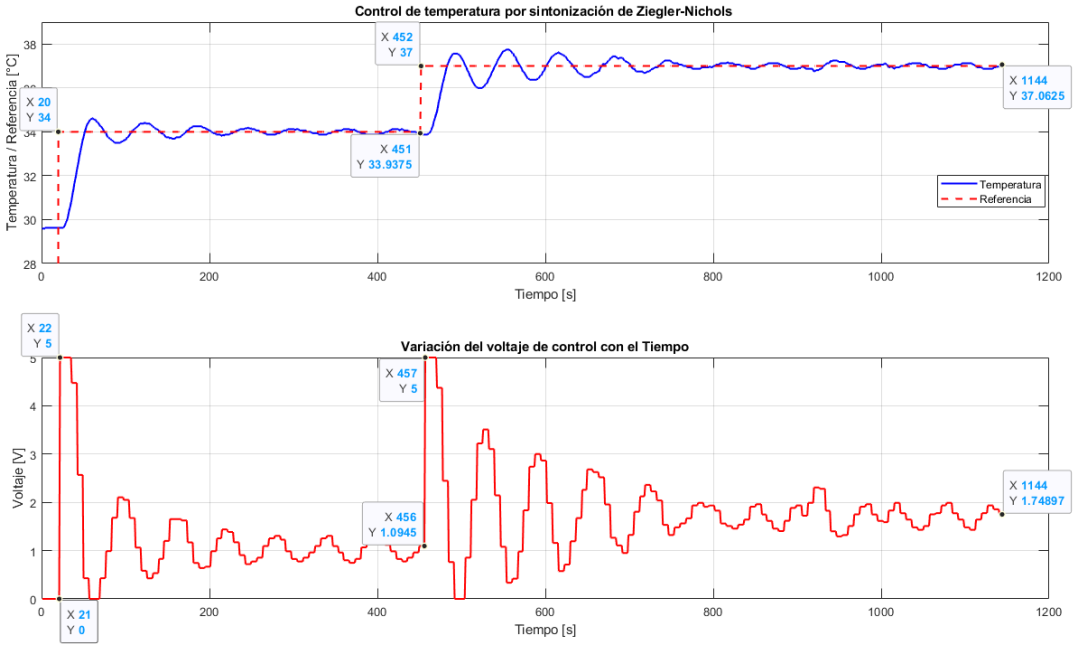


Figura 37. Resultados obtenidos del control PID por sintonización Ziegler-Nichols para temperatura en Arduino Uno.

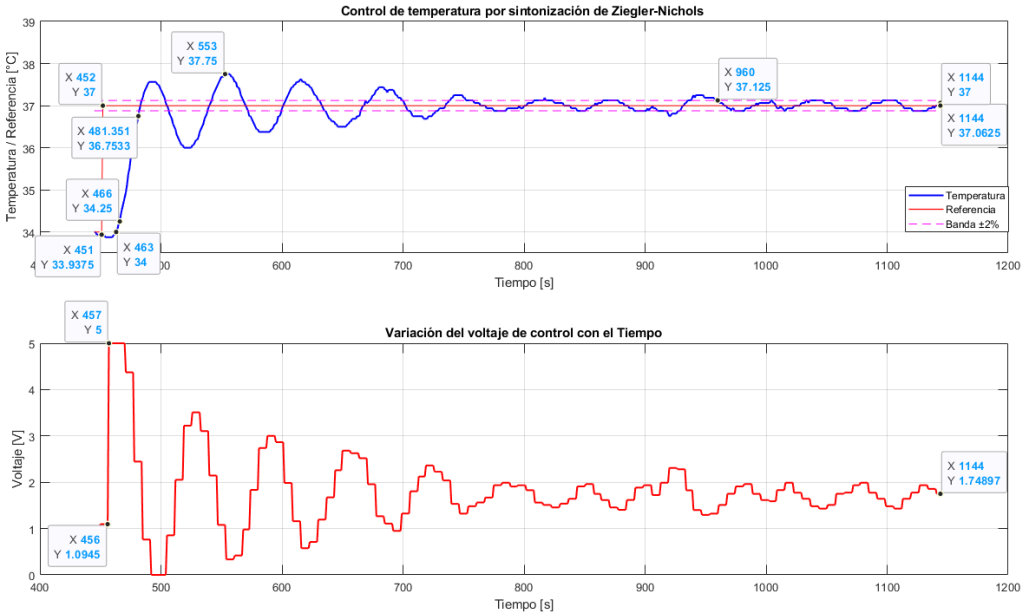


Figura 38. Segundo tramo para análisis de desempeño del controlador (ZN) en Arduino Uno.

Tabla 20.0 Datos obtenidos para control por sintonización Ziegler-Nichols en Arduino Uno.

Temperatura inicial	$\boldsymbol{To}$ $[°\boldsymbol{C}]$	33,93
Tiempo de inserción de setpoint en interfaz gráfica	t_{int} $[s]$	451
Tiempo de establecimiento a la banda del 2%	t_{s1} $[s]$	960
Setpoint	ref $[°C]$	37

Tiempo muerto:

$$t_{muerto} = 463 - 451 = 12[s]$$

El tiempo de establecimiento de acuerdo con la gráfica es:

$$t_s = 960 - 451 = 509[s]$$

El tiempo de subida que tarda en pasar del 10% al 90% del valor final es:

$$T_{90\%} = 0{,}9(37{,}0625 - 33{,}93) + 33{,}93 = 36{,}74[°C]$$

$$T_{10\%} = 0{,}1(37{,}0625 - 33{,}93) + 33{,}93 = 34.24[°C]$$

$$t_r = t_{90\%} - t_{10\%} = 481 - 466 = 15[s]$$

Tabla 21.0 Índices de desempeño del control (ZN) en Arduino Uno.

Tiempo de establecimiento en segundos ($\boldsymbol{t_s}$)	509
Tiempo de subida en segundos ($\boldsymbol{t_r}$)	15

3.4.4. Controlador por Realimentación de Estados (RE)

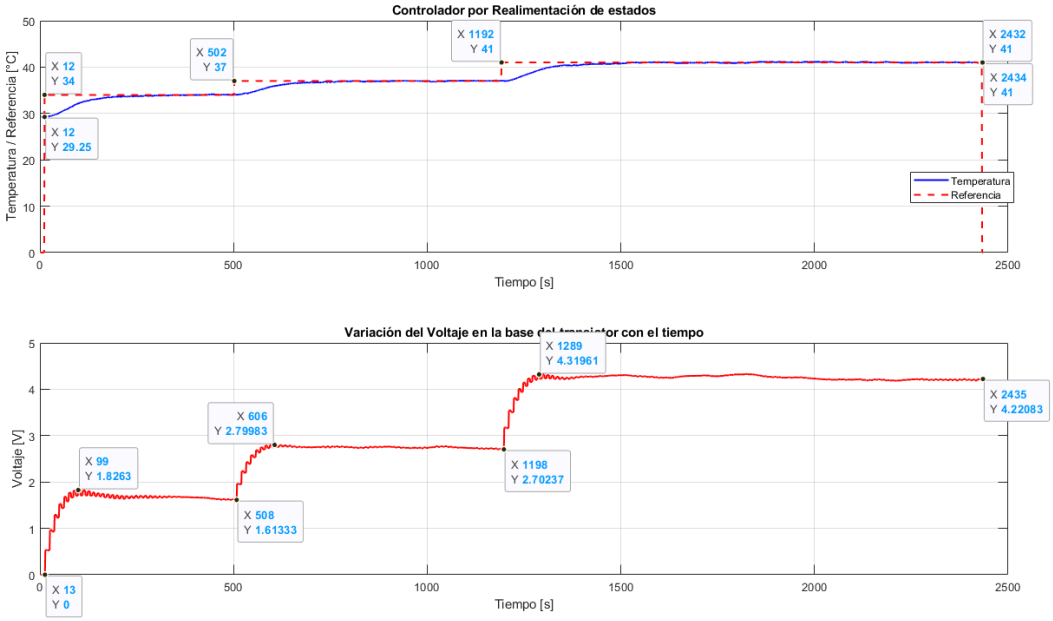


Figura 39. Resultados experimentales del control por realimentación de estados para planta de temperatura en Arduino Uno.

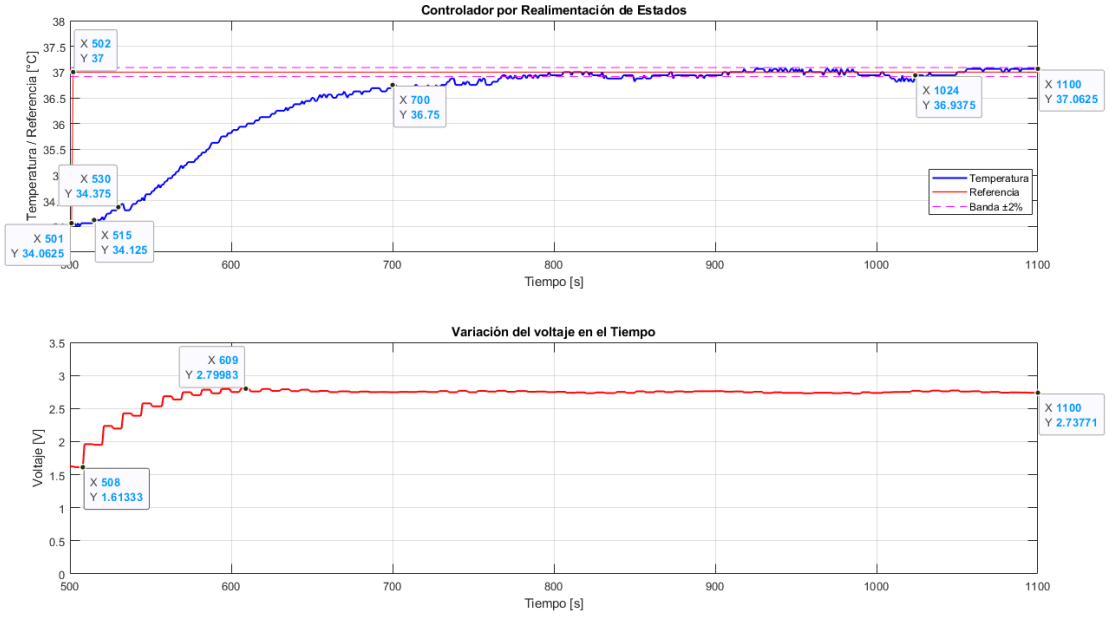


Figura 40. Segundo tramo para análisis de desempeño de control (RE) en Arduino Uno.

Tabla 22.0 Datos obtenidos del control (RE) en Arduino Uno.

Temperatura inicial	$\boldsymbol{To}$ $[°\boldsymbol{C}]$	34,0625
Tiempo de inserción de setpoint en interfaz gráfica	t_{int} $[s]$	501
Tiempo de establecimiento de llegada a banda del 2%	t_{s1} $[s]$	1024
Setpoint	ref $[°C]$	37

Tiempo muerto:

$$t_{muerto} = 515 - 501 = 14[s]$$

El tiempo de establecimiento de acuerdo con la gráfica es:

$$t_s = 1024 - 502 = 522[s]$$

El tiempo de subida que tarda en pasar del 10% al 90% del valor final es:

$$T_{90\%} = 0{,}9(37{,}0625 - 34{,}0625) + 34{,}0625 = 36{,}76[°C]$$

$$T_{10\%} = 0{,}1(37{,}0625 - 34{,}0625) + 34{,}0625 = 34.36[°C]$$

$$t_r = t_{90\%} - t_{10\%} = 700 - 530 = 170[s]$$

Tabla 23.0 Índices de desempeño obtenidos experimentalmente para el control (RE) en Arduino Uno.

Tiempo de establecimiento en segundos (t_s)	522
Tiempo de subida en segundos (t_r)	14

3.5. Análisis de tiempos de respuesta con Arduino Uno.

Para las implementaciones en la tarjeta Arduino Uno, la señal controlada tiene un comportamiento particular para los controladores Prueba – Error y Ziegler–Nichols, esto se puede ver en los tiempos de subida que presentan cada uno de ellos, 23 segundos para Prueba-Error y 15 segundos para Ziegler–Nichols que en comparación con la respuesta obtenida en el controlador Fuzzy-Mamdani con 162 segundos y 170 segundos para Realimentación de Estados, se puede observar que la respuesta de los controladores obtenidos mediante sintonización tienden a ser más rápidas en avanzar hacia las referencia dada, pero cuentan con mayor oscilación en sus respuestas. Por el contrario, los controladores Fuzzy-Mamdani y Realimentación de Estados proporcionan una respuesta más lenta, pero y tiempos de estabilización más prolongados.

Si se considera los tiempos de establecimiento de los mismos controladores, también se puede ver que el controlador Fuzzy-Mamdani es el que proporciona una respuesta considerablemente más lenta.

En cuanto a la estabilidad de la repuesta de cada controlador, se puede ver que Realimentación de Estados y Fuzzy-Mamdani nos permiten obtener señales controladas muy estables con una dinámica muy similar a la requerida mediante el Setpoint. Por el contrario, el controlador mediante sintonización Prueba-Error y Ziegler-Nichols su

respuesta, tiende a ser inestable, especialmente el controlador Ziegler-Nichols. Finalmente, cabe mencionar que la respuesta de estos dos últimos mencionados, depende básicamente de las constantes de control seleccionadas, lo que significa que se podría tener respuestas mejoradas si la sintonización es diferente.

3.6. Análisis de señal de control con Arduino Uno.

En cuanto a las señales de control, se presenta sobre picos iniciales en los controladores por sintonización Prueba-Error y Zeigler-Nichols, dichos picos llegan a los valores máximos para luego establecerse en el valor que la ley de control asigne para el setpoint requerido. Si se observa particularmente a la señal de control de Zeigler-Nichols, se presenta los picos al iniciar un cambio de Setpoint y además la señal de voltaje en la base del transistor es muy oscilante. Como se mencionó anteriormente, este comportamiento puede ser diferente si se considera otras constantes de control.

Tabla 24.0 Relación de voltaje de los controladores PE, Fuzzy, ZN y RE en Arduino Uno.

Voltaje [V]	Prueba Error (PE)	Fuzzy-Mamdani (FM)	Realimentación de estados (RE)	Ziegler-Nichols (ZN)
Voltaje máximo v_{max}	4,218	1,727	2,799	5,00
Voltaje una vez estabilizado v_{tss}	1,794	1,42	2,737	1,748
Voltaje inicial v_{ini}	1,110	0,860	1,613	1,094

En cambio, al analizar las señales de voltaje en el controlador por Realimentación de Estados y el Fuzzy son señales de control suave y que no presentan sobre nivel. Esto significa que el actuador del sistema no será sobre esforzado para al realizar el control a la planta.

3.7. Comparativa planta de Temperatura ESP32 vs Arduino Uno.

A continuación, se muestra las tablas comparativas entre el desempeño de los controladores de temperatura desarrollados.

Tabla 25.0 Índices de desempeño de los controladores PE, FUZZY, ZN y RE en ESP32.

Índices desempeño	Prueba Error (PE)	Fuzzy-Mamdani (FM)	Realimentación de estados (RE)	Ziegler-Nichols (ZN)
Tiempo estabilización t_{ss} [s]	662	1449	1033	348
Tiempo de subida t_r [s]	270	238	289	38
Tiempo muerto t_θ [s]	10	25	24	10

Tabla 26.0 Índices de desempeño experimentales para los controladores PE, ZN, RE, y FM en Arduino Uno.

Índices desempeño	Prueba Error (PE)	Fuzzy-Mamdani (FM)	Realimentación de estados (RE)	Ziegler-Nichols (ZN)
Tiempo estabilización t_{ss} [s]	488	840	522	509
Tiempo de subida t_r [s]	23	162	170	15
Tiempo muerto t_θ [s]	12	11	14	12

A pesar de que la tarjeta ESP32 cuenta con capacidad computacional más alta, mediante las tablas mostradas se puede ver que en general todos los controladores implementados tienen una mejor respuesta en la tarjeta Arduino Uno. Aunque, en análisis particular el controlador por Zeigler-Nichols implementado en Esp32 tiene un menor tiempo de

estabilización, pero mayor tiempo de subida que el mismo controlador, pero implementado mediante la tarjeta Arduino Uno.

Una de las razones por las que el control es más efectivo en una tarjeta de Arduino uno, se debe a que la base del transistor en este caso está en el rango de 0Vdc hasta 5Vdc, mientras que para una tarjeta ESP32 el voltaje esta entre valores de 0Vdc hasta 3.3Vdc.

Por lo tanto, al estar controlando el nivel de voltaje en la base mediante una señal PWM cuando la señal de control este en un 50% para ambos casos, el nivel de voltaje será 1.65Vdc para ESP32 y 2.5Vdc para el control en Arduino Uno.

3.8. Desempeño de controladores Planta de Velocidad en ESP32.

3.8.1. Controlador PID por sintonización Prueba-Error.

Tras una sintonización usando PID Tunner de Matlab se determinó las constantes $K_p = 0.01,\ Ti = 2.347\ y\ Td = 0.0$, con lo cual se obtuvo para varios valores de referencia los siguientes resultados experimentales:

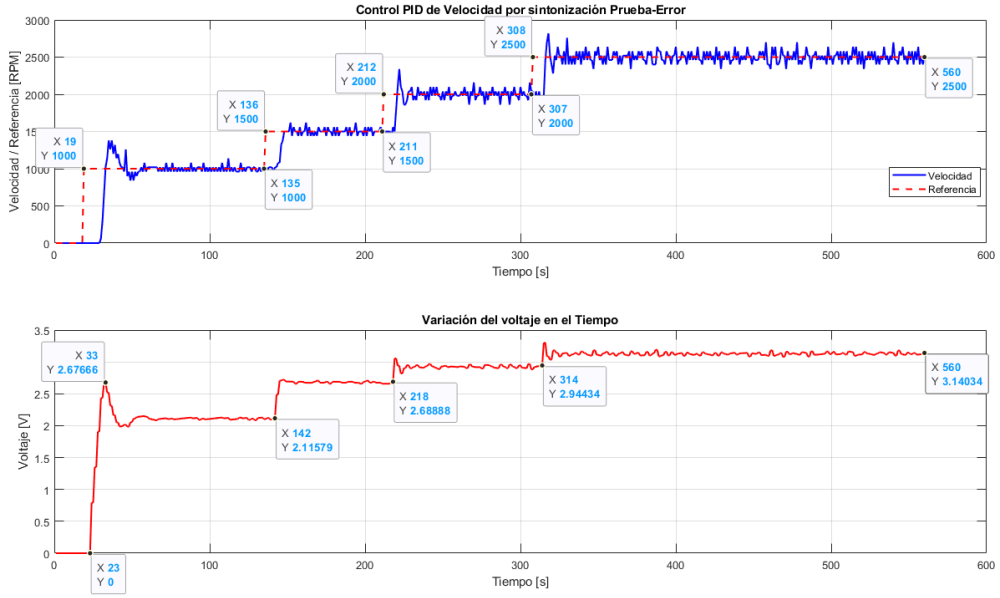


Figura 41. Respuesta obtenida de control PID para velocidad por sintonización Prueba-Error.

Al obtener la gráfica se observó que se presenta ruido eléctrico, sin embargo, de acuerdo con los objetivos de este trabajo se muestra que se puede realizar una sintonización prueba-error y se sugiere para trabajos posteriores diseñar un filtro que permita obtener mejores resultados. Dado este ruido se ha considerado una banda de análisis del 5% alrededor del valor de referencia. Considerando el segundo tramo donde el setpoint es de 1500 RPM tenemos:

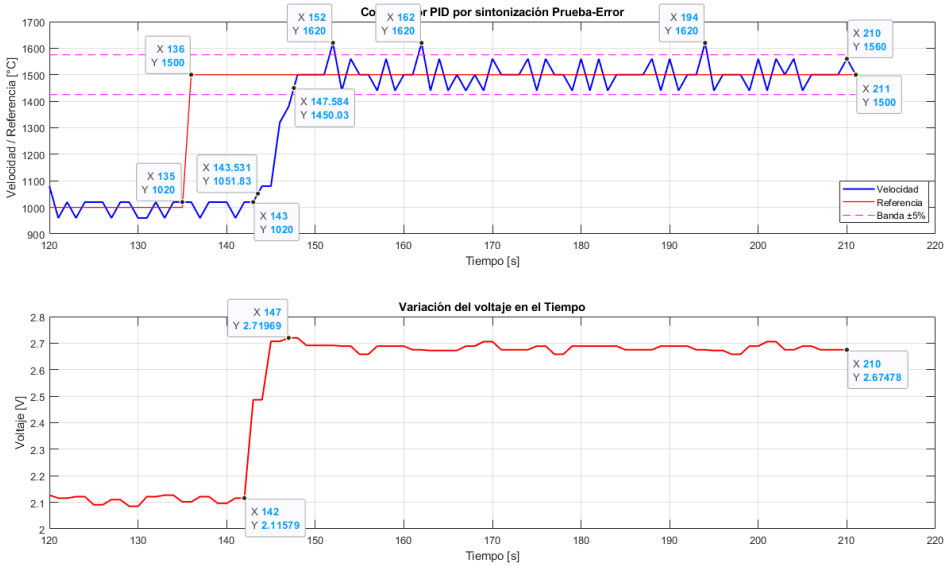


Ilustración 1. Respuesta segundo tramo para análisis de desempeño de control de velocidad por sintonización Prueba-Error.

Tabla 27.0 Índices de desempeño para el control PID (PE) con ESP32 para control de velocidad.

Tiempo de establecimiento en segundos (t_s)	17,5
Tiempo de subida en segundos (t_r)	4,05
Valor de máximo pico (M_p)	1620,0
Tiempo Pico (T_p)	17,0
Sobrenivel porcentual (%SP)	11,11
Tiempo de retardo (t_θ)	8,0

3.8.2. Controlador Fuzzy-Mamdani

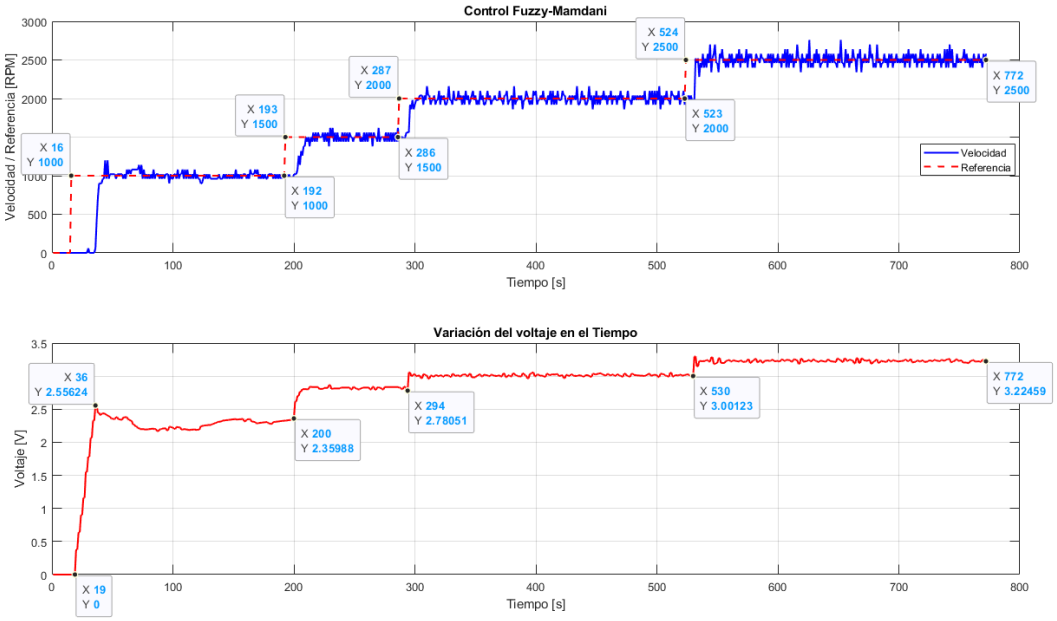


Figura 42. Respuesta experimental del control Fuzzy-Mamdani de velocidad

Si se analiza el segundo tramo con un setpoint de 1500 RPM tenemos:

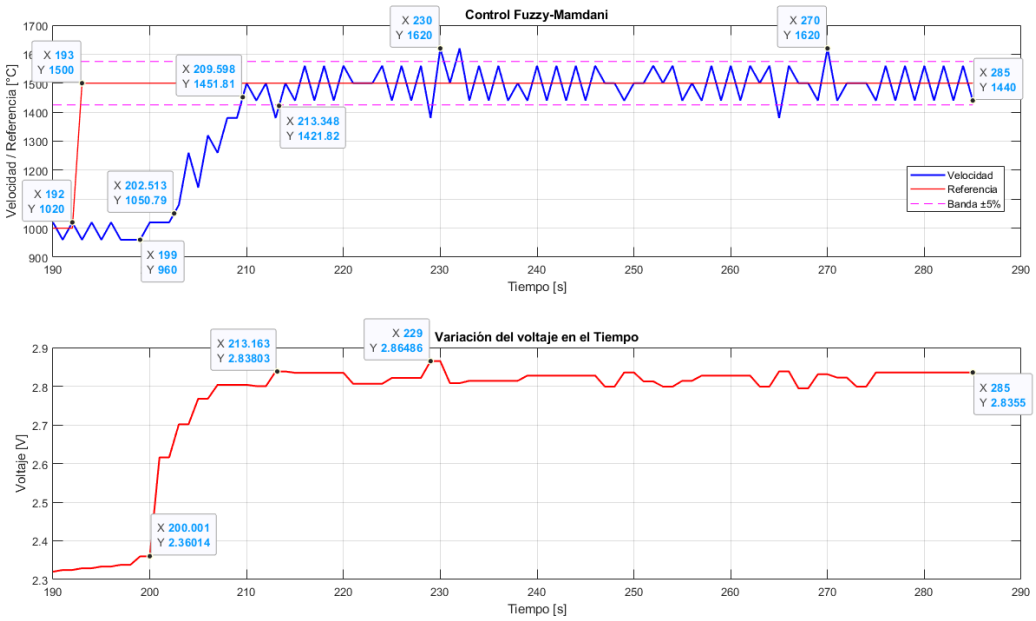


Figura 43. Resultados experimentales para análisis de desempeño del segundo tramo.

Tabla 28.0 Índices de desempeño para el control Fuzzy-Mamdani con ESP32 para control de velocidad.

Tiempo de establecimiento en segundos (t_s)	21,34
Tiempo de subida en segundos (t_r)	7,08
Valor de máximo pico (M_p)	1620,0
Tiempo Pico (T_p)	38
Sobrenivel porcentual (%SP)	11,11
Tiempo de retardo (t_θ)	7,0

3.9. Desempeño de controladores Planta de Velocidad en Arduino Uno

3.9.1. Controlador PID Sintonización prueba-error

Se ha realizado una sintonización prueba-error, determinando las constantes Kp=0.01151, Ti = 1.231 y Td = 0.2511 y se ha hallado los siguientes resultados:

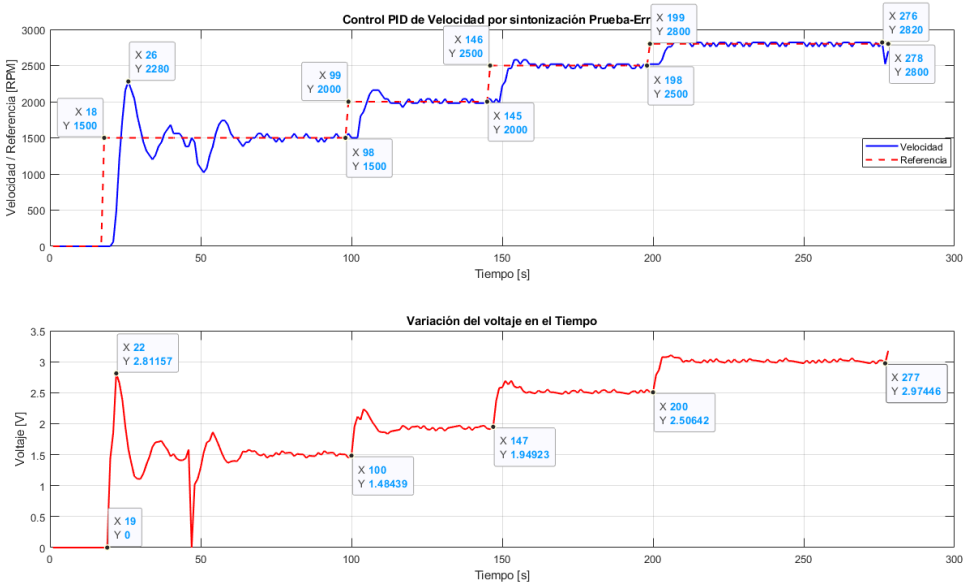


Figura 44. Resultados de control (PE) para control de velocidad implementado en Arduino Uno.

Analizando el segundo tramo con un setpoint de 2000 tenemos:

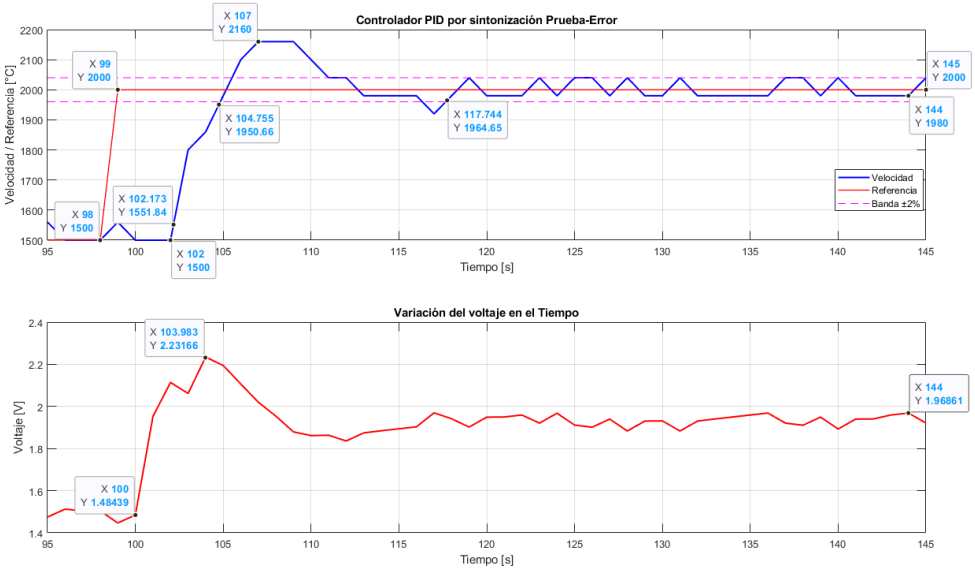


Figura 45. Resultados de segundo tramo para análisis de desempeño.

Tabla 29.0 Índices de desempeño para el control PID por sintonización PE con ESP32 para control de velocidad.

Tiempo de establecimiento en segundos (t_s)	19,74
Tiempo de subida en segundos (t_r)	2,58
Valor de máximo pico (M_p)	2160,0
Tiempo Pico (T_p)	9,0
Sobrenivel porcentual (%SP)	22,22
Tiempo de retardo (t_θ)	4,0

3.9.2. Controlador Fuzzy-Mamdani

Este controlado no depende de la función de transferencia, se aplicó 7 variables lingüísticas y se obtuvo los siguientes resultados experimentales:

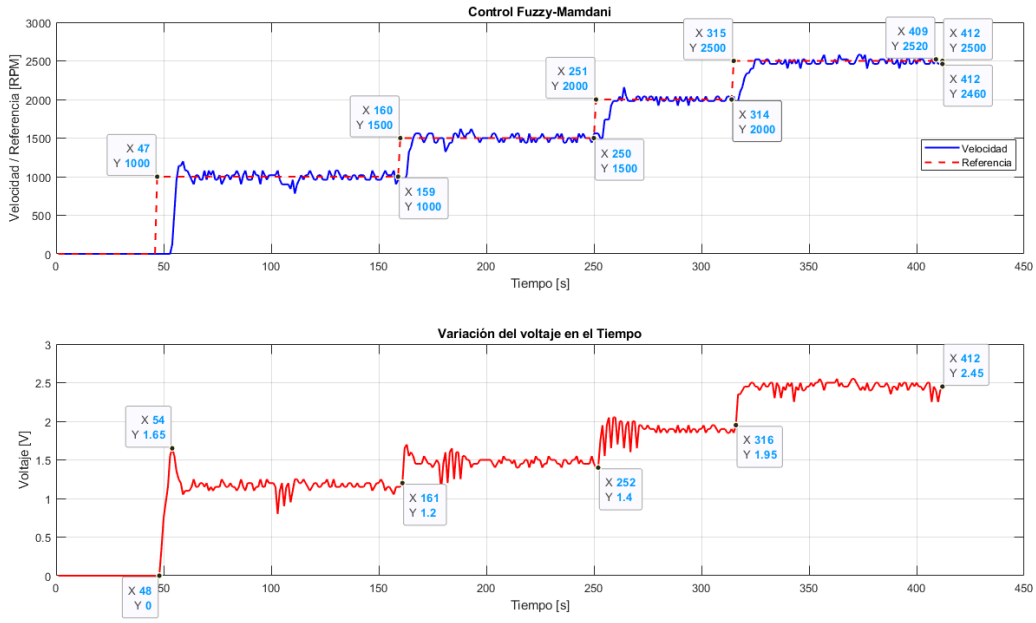


Figura 46. Resultados experimentales de controlador Fuzzy-Mamdani aplicado a control de velocidad con Arduino Uno.

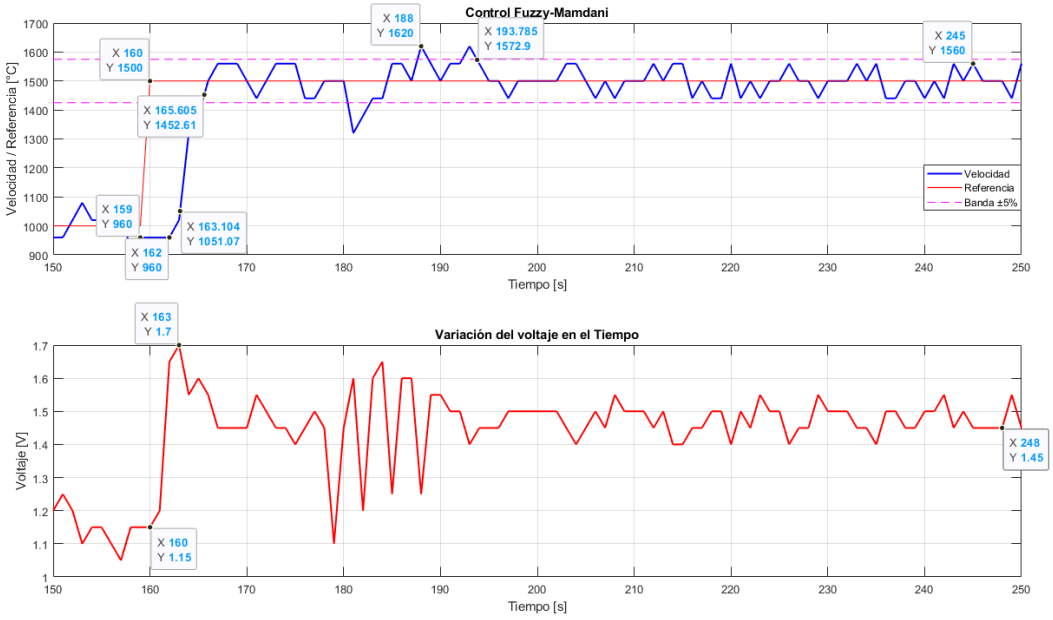


Figura 47. Resultados para análisis del segundo tramo del control.

Tabla 30.0 Índices de desempeño para el control Fuzzy-Mamdani con Arduino para control de velocidad.

Tiempo de establecimiento en segundos (t_s)	34,00
Tiempo de subida en segundos (t_r)	2,5
Valor de máximo pico (M_p)	1620,0
Tiempo Pico (T_p)	29,00
Sobrenivel porcentual (%SP)	10,0
Tiempo de retardo (t_θ)	3,0

3.10. Comparativa planta de Velocidad ESP32 vs Arduino Uno

De acuerdo con las tablas 30.0 y 31.0 de índices de desempeño halladas para los controladores PID por sintonización Prueba-Error, se obtuvo un menor tiempo de estabilización usando la ESP32 que el dado por el Arduino Uno, la diferencia entre las dos medidas es de 2,24 segundos. Sin embargo, el tiempo de subida resultó menor en el Arduino; el control PE en Arduino presenta un mayor pico, y el doble de sobrenivel porcentual. Respecto del retardo la ESP32 presenta mayor retardo, esto es por el voltaje de 3.3V.

Tabla 31.0 Índices de desempeño de los controladores PE y FUZZY en ESP32 control de velocidad.

Índices desempeño	Prueba Error (PE)	Fuzzy-Mamdani (FM)
Tiempo estabilización t_{ss} [s]	17,50	21,34
Tiempo de subida t_r [s]	4,05	7,08
Valor pico máximo (Mp)	7,08	1620,00
Tiempo pico t_p [s]	17,00	38,00
Sobrenivel Porcentual (%SP)	11,11	11,11
Tiempo muerto t_θ [s]	8,00	7

Tabla 32.0 Índices de desempeño de los controladores PE y FUZZY en Arduino Uno control de velocidad.

Índices desempeño	Prueba Error (PE)	Fuzzy-Mamdani (FM)
Tiempo estabilización t_{ss} [s]	19,74	34,00
Tiempo de subida t_r [s]	2,58	2,50
Valor pico máximo (Mp)	2160,00	1620,00
Tiempo pico t_p [s]	9,00	29,00
Sobrenivel Porcentual (%SP)	22,22	10,00
Tiempo muerto t_θ [s]	4,00	3,00

Adicionalmente, de acuerdo con los datos se destaca un mejor desempeño de la placa Arduino Uno, esto nuevamente está ligado al voltaje de operación, que permite una corriente de base mayor en el control de temperatura, pero también en el pin habilitador del Drive L293D. Además, analizando los rangos de voltaje de cada controlador y en cada tarjeta de desarrollo tenemos:

Tabla 33.0 Relación de voltaje de los controladores PE *y* FUZZY en Arduino Uno control de velocidad.

	Tipo de Control	Voltaje inicial [V]	Voltaje de estabilización [V]	Voltaje máximo [V]
ESP32	Prueba Error (PE)	2,115	2,674	2,719
	Fuzzy-Mamdani (FM)	2,36	2,835	2,864
Arduino Uno	Prueba Error (PE)	1,484	1,968	2,23
	Fuzzy-Mamdani (FM)	1,15	1,45	1,7

3.11. Análisis de costos.

A continuación, se muestra una relación de precios y costos total de los equipos adquiridos.

Tabla 34.0 Tabla de costos de equipos usados. Véase la sección de anexos.

Dispositivo	*Cantidad*	*Costo unitario ($)*	*Costo acumulado ($)*
Motor DC SANKO 5.9V	2	2,5	5,00
Capacitor cerámico 0.33uF 50V	4	0,2	0,8
Leds	6	0,15	0,90
Conector hembra o Jack	2	0,5	1,0
Conectores	2	0,5	1,0
Transistor TIP31	4	1,5	6,0
Resistor	14	0,15	2,10
Sensor DS18B20	4	3,0	12,0
Driver L293D	2	2,5	5,0
ESP32	1	15,0	15,0
Regulador LM7805	2	1,0	2,0
PCB	2	15,0	30,0
Encoder óptico	2	3,0	6,0
Disipador	4	1,5	6,0
Soporte PCB	8	0,25	2,0
Cable USB	2	5,0	10,0
Diodo 1N4001	2	0,1	0,2
Armado y Soldado de componentes	2	30,0	60,0
Arduino Uno con cable USB (Réplica)	1	16,0	16,0
PCB para montaje y adaptaciones.	6	-	57,00
Acrílico	2	-	25
Ventilador	2	1,75	3,50
Base plástica	2	25,00	50,00
Pantalla OLED 0.96	2	6,40	12,80
Raspberry Pi Pico	2	12,00	24,00

Fuente de alimentación	2	5,00	10,00
	Costo total		363,30

De acuerdo con la tabla anterior, el costo del proyecto rodea los $364,00 considerando ambos dispositivos, con Arduino y con ESP32. Ahora, teniendo en cuenta que la finalidad de este proyecto es del tipo investigativo y educativo; se aplicaría en materias de sistemas de control y sistemas embebidos de pregrado y posgrado. Teniendo en consideración que, si se quiere implementar en un laboratorio se puede conseguir reducción de precios al conseguir dichos materiales al comprar al por mayor o mediante convenios institucionales/educativos. Dependiendo del enfoque se puede decidir si invertir en usar solo el control con ESP32 con lo cual el precio reduce a la mitad, es decir $182,00 por unidad, esto es viable y además dicha tarjeta presenta ciertas características que permitirán ampliar la investigación a otras aplicaciones de IoT para sistemas embebidos o a su vez para desarrollo de otros controladores que se imparten en posgrado. Analizando costo para un laboratorio de 10 personas e implementando solo con la tarjeta ESP32 tenemos:

Tabla 35.0 Costos de implementación para un laboratorio de 10 personas.

Equipo	*Costo unitario ($)*	*Costo de 10 unidades ($)*
ESP32 WROOM 32D	15,00	150
ARDUINO UNO (REPLICA)	16,00	160
TSC-LAB	90,00	900,00
Pantalla OLED 0.96	6,40	64,00
Raspberry Pi Pico	12,00	120,00
Acrílico	25,00	250
Base plástica	25,00	250
Ventilador	1,75	17,5
Fuente de alimentación	5,0	50,00
	Costo total ($)	**1961,50**

CAPÍTULO 4

CONCLUSIONES Y RECOMENDACIONES

El desarrollo de este proyecto ha conseguido resultados que se ajustan a lo esperado en los objetivos, lo cual es muy importante y también se muestra numéricamente en la sección anterior. Se destaca que el desarrollo de este trabajo es el puente de introducción a aplicaciones más específicas, ya que, permite forjar la base para trabajos posteriores en donde los expertos de control podrán partir y diseñar controladores mucho más eficientes de acuerdo con el tipo de aplicación y también en el manejo de los datos que se requieran obtener mediante programación de sistemas embebidos. Con este sistema el estudiante tiene oportunidades de experimentar el comportamiento en tiempo real, realizar varias prácticas para aterrizar conceptos y desarrollar habilidades tanto en el ámbito de la programación de sistemas embebidos como en el diseño del control.

4.1 Conclusiones

1. Se implementó un sistema embebido basado en las tarjetas Arduino, ESP32 y TSC-LAB que permite el control de temperatura y control velocidad en tiempo real para visualizar el desempeño de diferentes controladores usando de técnicas de control moderno.
2. Se diseño una interfaz gráfica mediante App Designer de MATLAB para la visualización del comportamiento en tiempo real de los controladores diseñados. Dicha interfaz permite la comunicación serial con el microcontrolador usado (ESP32 o Arduino), exportar los datos de interés a archivos Excel, con los cuales se puede realizar análisis detallados como se hizo en el capítulo de resultados.
3. Se diseño los controladores PID por sintonización Ziegler-Nichols y Prueba-Error para su implementación en las tarjetas ESP32 y Arduino Uno para planta

de temperatura, con lo cual se obtuvo la comparativa del desempeño destacando la tarjeta de control basada en Arduino Uno con menor tiempo de estabilización.

4. Se diseño los controladores por realimentación de estados y lógica difusa para ser implementados en la tarjeta TSC-LAB mediante el cual se realizó pruebas para el control de temperatura que permitieron validarlos como controladores de respuesta lenta pero óptimos en la precisión.
5. Se verificó que los controladores de velocidad Prueba-Error y Fuzzy-Mamdani embebidos en la tarjeta ESP32 proporcionan una mejor respuesta en tiempo de estabilización, que en la tarjeta Arduino Uno; aunque si se observa el sobre nivel porcentual en ESP32 para el controlador Prueba-Error existen mayores picos en las respuestas.

4.2 Recomendaciones

1. En el control de velocidad hay ciertos picos de velocidad, estos se deben a problemas con el Encoder óptico electrónico y el motor que se induce ruido eléctrico, se recomienda diseñar un filtro que permita mitigar este efecto en la salida de velocidad para tener una respuesta más eficiente.
2. En la sección de comunicación serial de la interfaz gráfica se debe tener en consideración que el bucle de repetición necesita un tiempo para recibir los datos, si se agrega un retardo de tiempo adicional en el código de los controladores; estos tiempo se suman y el tiempo de la gráfica no corresponderá al del tiempo real, por ello, se recomienda analizar cuando agregar o no retardos

de tiempo dependiendo de las formas de programación de los controladores y procesamiento de las tarjetas de desarrollo.

3. Para fines prácticos se recomienda diseñar una tarjeta electrónica o PCB que permita integrar de forma sencilla las plantas de control de temperatura y de velocidad con las tarjetas ESP32, Arduino y Raspberry Pi Pico.
4. Se recomienda primero realizar un proceso de identificación del sistema que se requiere controlar para obtener resultados más precisos.
5. Para tener un control óptimo se recomienda diseñar un sistema electrónico con optoacopladores que permita el control de una planta de temperatura o velocidad industrial, esto es controlar mediante señal PWM.

BIBLIOGRAFÍA

Arduino. (13 de Junio de 2023). *Arduino*. Obtenido de https://docs.arduino.cc/learn/starting-guide/whats-arduino

Banzi, M., & Shiloh, M. (2014). *Getting Started with Arduino 3a. Edition.* Sebastopol: Maker Media.

Castaño Giraldo, S., Hernández Gómez, D., & Gallo Blandón, J. (20 de Diciembre de 2013). Control y monitoreo de temperatura para un horno de curado de prendas índigo utilizando lógica difusa y controles pi. *9(17)*, 69-81. Medellín, Colombia: Revista Politécnica. Obtenido de https://revistas.elpoli.edu.co/index.php/pol/article/view/342

Espressif Systems. (14 de Junio de 2023). *Espressif Systems*. Obtenido de https://www.espressif.com/en/company/about-espressif

Jaimes, L. E. (2009). *Control Digital, Teoría y Práctica 2ª. Edición.* Medellín.

MathWorks. (2023). *MathWorks*. Obtenido de https://www.mathworks.com/company.html?s_tid=hp_ff_a_company

Ogata, K. (1996). *Sistema de control en tiempo discreto 2ª edición.* México: Prentice Hall.

Ogata, K. (1998). *Ingeniería de Control Moderna* (Tercera Edición ed.). (M. Á. Sarmiento, Trad.) Mexico: PRENTICE-HALL.

Ojeda Carrera, V., Asanza, V., Chica-Orellana, K., Cagua, J., Plaza, D., Martín, C., & Peluffo-Ordóñez, D. (25 de Abril de 2021). *Temperature and Speed Control Lab (TSC-LAB)*. Obtenido de IEEE Dataport: https://dx.doi.org/10.21227/8cty-6069

Proakis, J., & Manolakis, D. (1996). *Digital Signal Processing: Principles, Algorithms, and Applications 3a Edition.* New Jersey: Prentice Hall.

Samuel Diciembre Sanahuja. (2017). *Sistemas de Control con Lógica Difusa: Métodos de Mamdani y de Takagi-Sugeno-Kang (TSK).*

Theler, G. (2007). *Controladores basados en lógica difusa y loops de convección natural caóticos.* San Carlos de Bariloche.

S. Amuthameena & S. Monisa. (2017). Design of fuzzy logic controller for a non-linear system. IEEE International Conference on Electrical, Instrumentation and Communication Engineering (ICEICE), Karur, India, 2017, pp. 1-7, doi: 10.1109/ICEICE.2017.8191844.

APÉNDICE

APÉNDICE A.

Figura A.1 Implementaciones físicas Control TSC.

Figura 48. Equipos implementados, ESP32-ARDUINO UNO-TSCLAB vista frontal.

Figura 49. Equipos implementados, ESP32-ARDUINO UNO-TSCLAB vista superior-frontal.

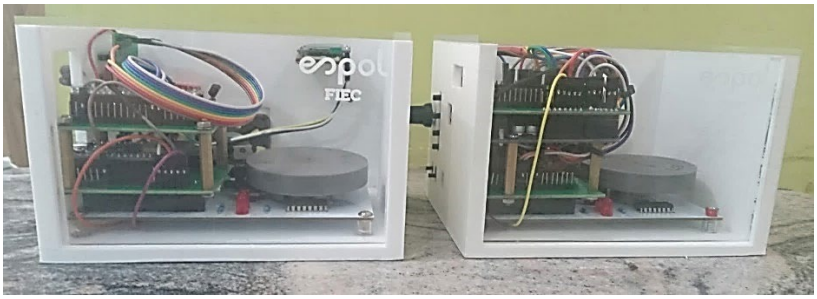
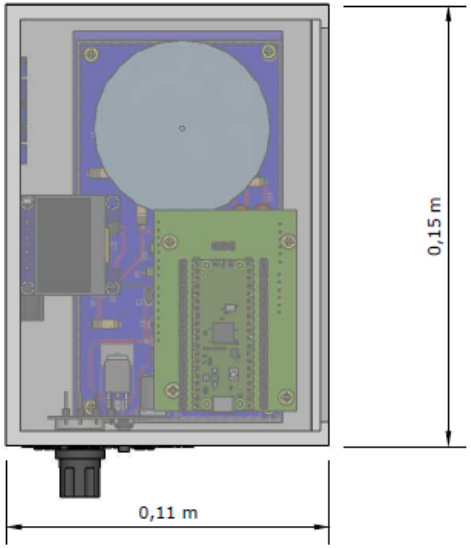


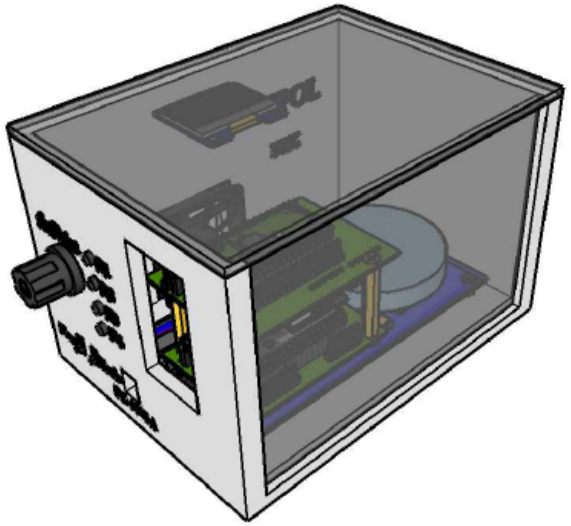
Figura 50. Equipos implementados, ESP32-ARDUINO UNO-TSCLAB vista lateral.

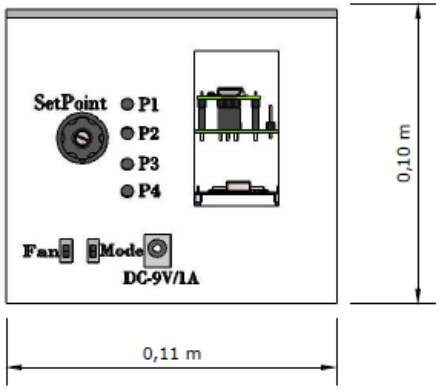
APÉNDICE B.

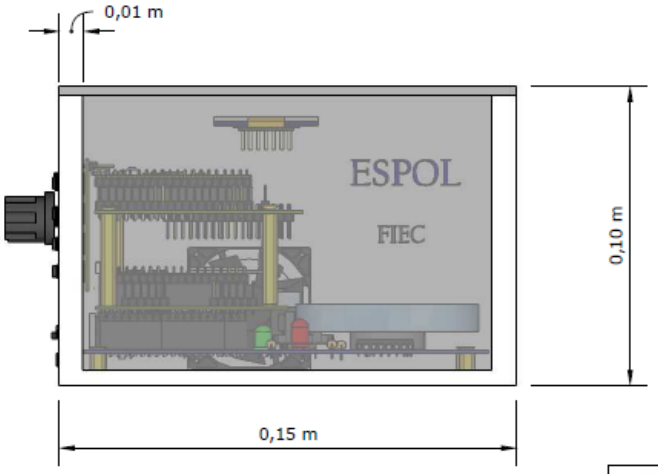
Figura B.1 Esquemático físico de equipo Control TSC con ESP32.

Figura B.2 Esquemático físico de equipo Control TSC con Arduino Uno.

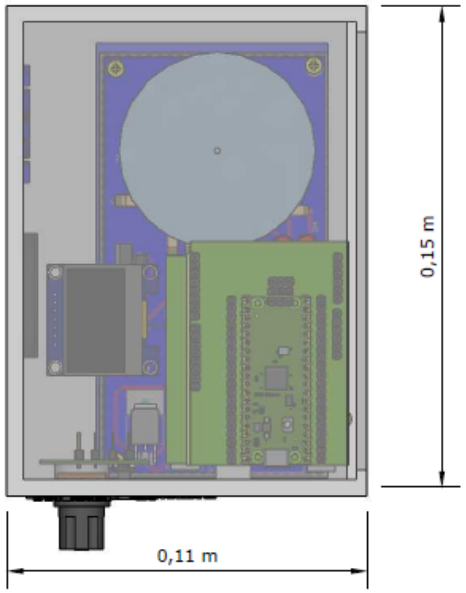


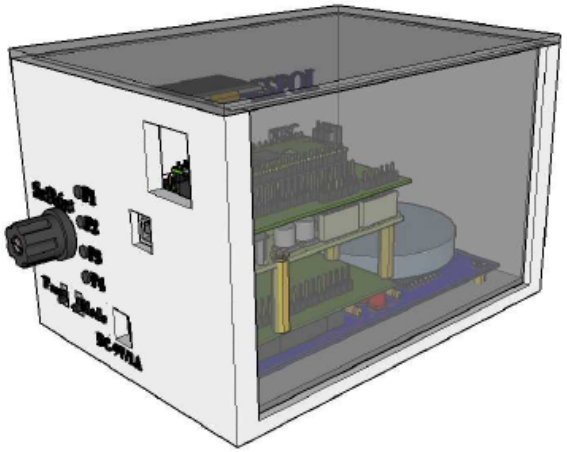


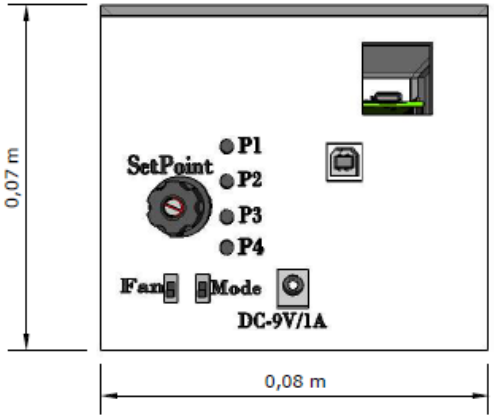


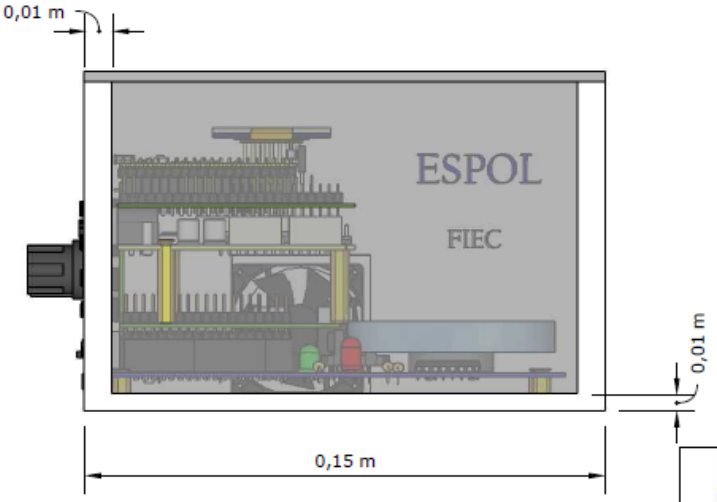


espol	ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL		
	Descripción: Esquema físico ControlTSC		
Tarjeta: ESP32	F: A4	Autor: J.Martinez-J.Nicolalde	Pag: 1 de 1
Fecha: 24/08/23	Version:1.0	Dibujo:J.Martínez	Rev:







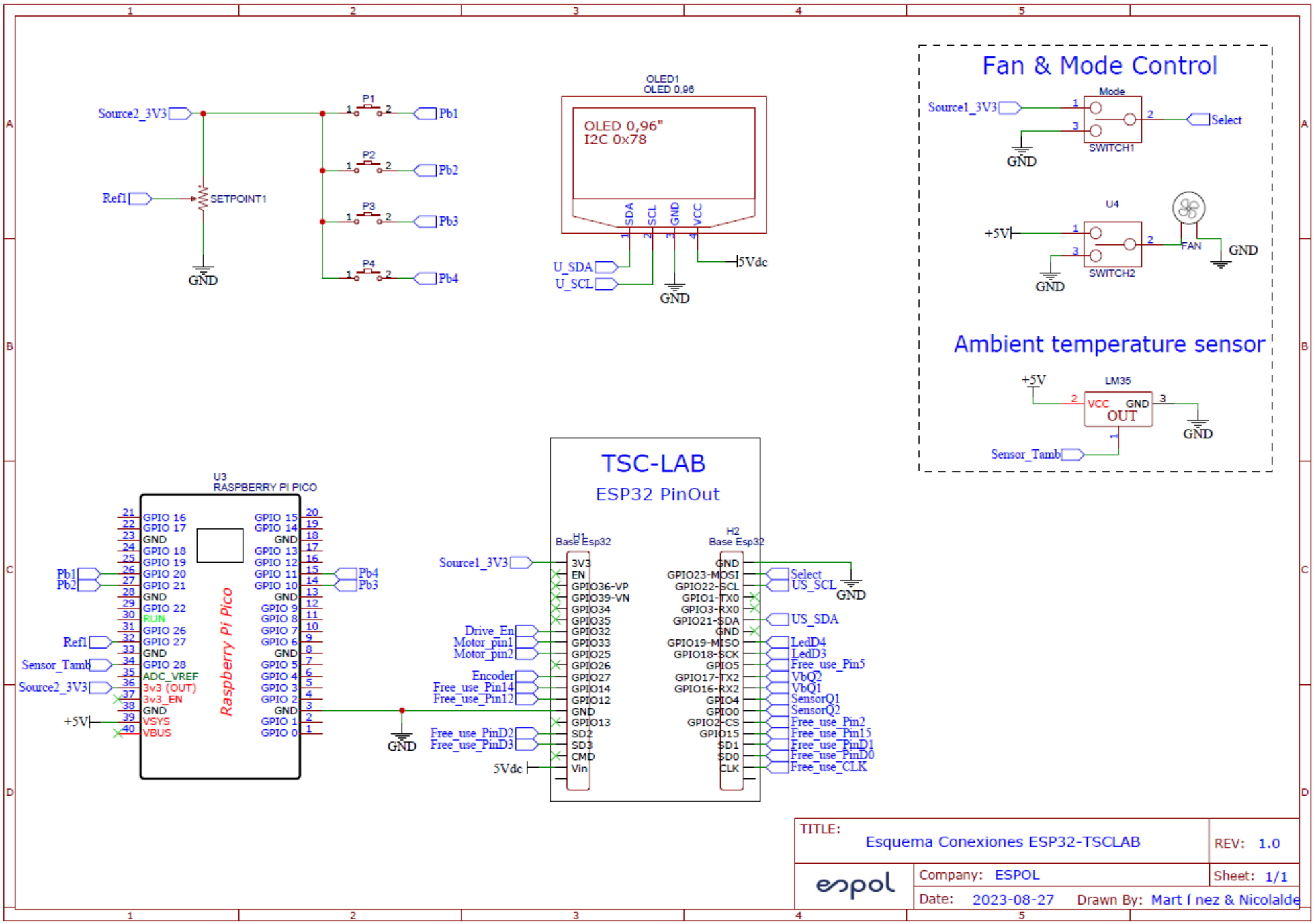


espol	ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL		
	Descripción: Esquema físico ControlTSC		
Tarjeta: Arduino Uno	F: A4	Autor: J.Martinez-J.Nicolalde	Pag: 1 de 1
Fecha: 24/08/23	Version:1.0	Dibujo:J.Martínez	Rev:

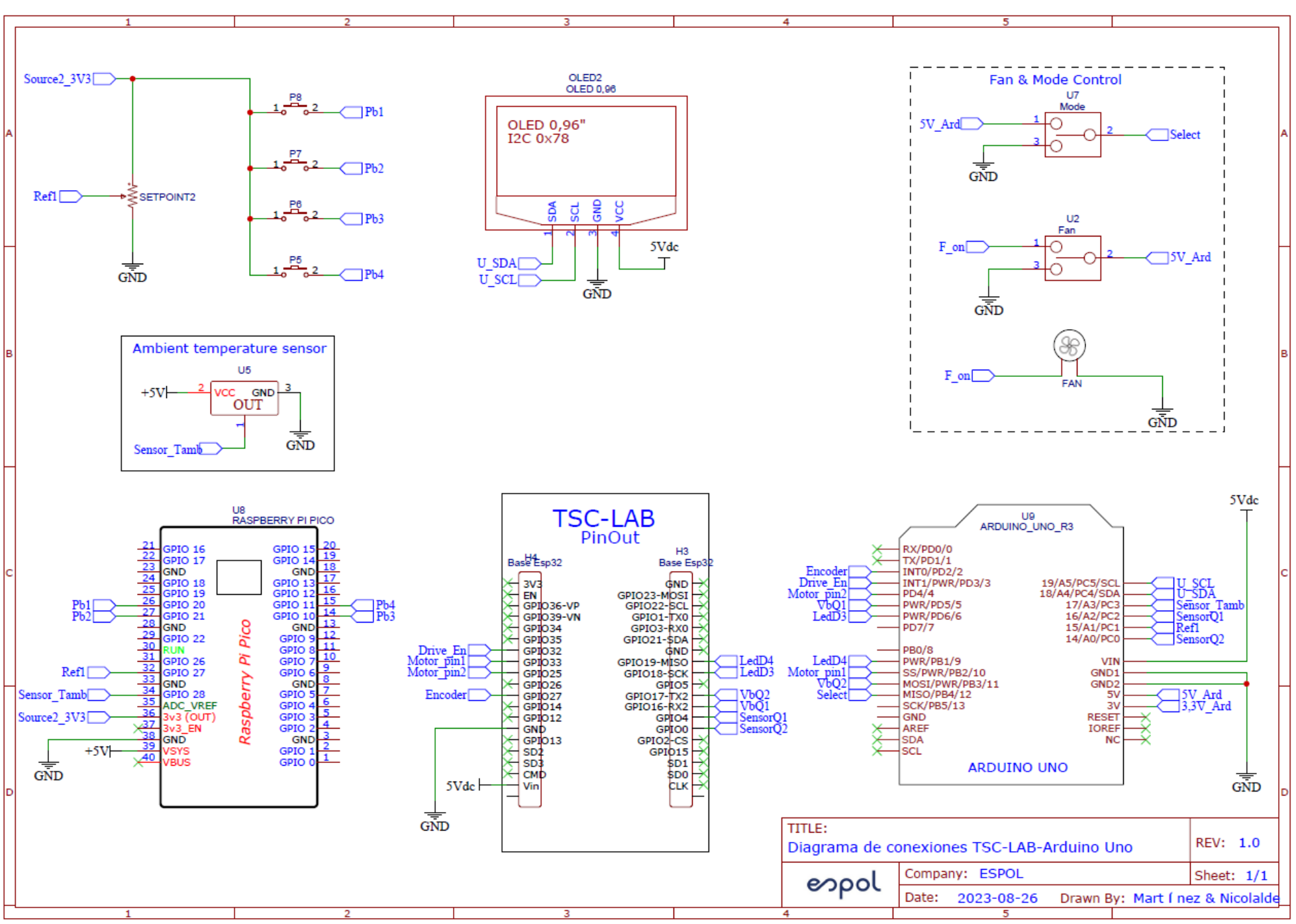
APÉNDICE C.

Figura C.1 Diagrama de conexiones de dispositivos ESP32-TSCLAB.

Figura C.2. Diagrama de conexiones de dispositivos ARDUINO UNO-TSCLAB.



TITLE: Esquema Conexiones ESP32-TSCLAB		REV: 1.0
espol	Company: ESPOL	Sheet: 1/1
	Date: 2023-08-27	Drawn By: Martínez & Nicolalde


Source2_3V3
Ref1
SETPOINT2
GND
P8
P7
P6
P5
Pb1
Pb2
Pb3
Pb4
OLED2
OLED 0,96
OLED 0,96"
I2C 0x78
SDA
SCL
GND
VCC
U_SDA
U_SCL
5Vdc
Fan & Mode Control
U7
Mode
5V_Ard
Select
U2
Fan
F_on
FAN
Ambient temperature sensor
U5
+5V
VCC
GND
OUT
Sensor_Tamb
U8
RASPBERRY PI PICO
Raspberry Pi Pico
GPIO 16
GPIO 17
GPIO 18
GPIO 19
GPIO 20
GPIO 21
GPIO 22
RUN
GPIO 26
GPIO 27
GPIO 28
ADC_VREF
3v3 (OUT)
3v3_EN
VSYS
VBUS
GPIO 15
GPIO 14
GPIO 13
GPIO 12
GPIO 11
GPIO 10
GPIO 9
GPIO 8
GPIO 7
GPIO 6
GPIO 5
GPIO 4
GPIO 3
GPIO 2
GPIO 1
GPIO 0
TSC-LAB
PinOut
H4
Base Esp32
H3
Base Esp32
3V3
EN
GPIO36-VP
GPIO39-VN
GPIO34
GPIO35
GPIO32
GPIO33
GPIO25
GPIO26
GPIO27
GPIO14
GPIO12
GPIO13
SD2
SD3
CMD
Vin
GPIO23-MOSI
GPIO22-SCL
GPIO1-TX0
GPIO3-RX0
GPIO21-SDA
GPIO19-MISO
GPIO18-SCK
GPIO5
GPIO17-TX2
GPIO16-RX2
GPIO4
GPIO0
GPIO2-CS
GPIO15
SD1
SD0
CLK
Drive_En
Motor_pin1
Motor_pin2
Encoder
LedD4
LedD3
VbQ2
VbQ1
SensorQ1
SensorQ2
U9
ARDUINO_UNO_R3
ARDUINO UNO
RX/PD0/0
TX/PD1/1
INT0/PD2/2
INT1/PWR/PD3/3
PD4/4
PWR/PD5/5
PWR/PD6/6
PD7/7
PB0/8
PWR/PB1/9
SS/PWR/PB2/10
MOSI/PWR/PB3/11
MISO/PB4/12
SCK/PB5/13
AREF
SDA
SCL
19/A5/PC5/SCL
18/A4/PC4/SDA
17/A3/PC3
16/A2/PC2
15/A1/PC1
14/A0/PC0
VIN
GND1
GND2
5V
3V
RESET
IOREF
NC
U_SCL
U_SDA
Sensor_Tamb
SensorQ1
Ref1
SensorQ2
5V_Ard
3,3V_Ard
TITLE:
Diagrama de conexiones TSC-LAB-Arduino Uno
REV: 1.0
Company: ESPOL
Sheet: 1/1
Date: 2023-08-26
Drawn By: Mart ínez & Nicolalde

APÉNDICE D.

Figura D.1 Código en IDE de Arduino para control PID de temperatura por sintonización Prueba-Error en ESP32.

PT_ESP32_PE_OLED.ino | lib_pid_zn.cpp | lib_pid_zn.h

```
/*  ================================================
    CONTROLADOR PID POR SINTONIZACIÓN PRUEBA Y ERROR
      By Jesús Martínez & Javier Nicolalde
        PLANTA DE CONTROL DE TEMPERATURA
          Implementación ESP32
          Última revisión: 23/08/2023
    ================================================*/

//                    LIBRERIAS
#include <OneWire.h>
#include <DallasTemperature.h>  // Librería de temperatura ds18b20
#include <Ticker.h>            // Librería de interrupciones
#include "lib_pid_zn.h"       // Librería usada para funciones PID ZN
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
// Definiciones componentes de la tarjeta

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET    -1
#define SCREEN_ADDRESS 0x3C
#define SENSOR_PIN 0   //sensor DS18B20
#define POT_PIN 2     // Potenciómetro referencia física
#define V_BPIN 16     // v_base transistor TIP31C
#define ALAR_PIN 18  //Led para indicar "alarma de caliente transistor"
#define MODE_PIN 23 // Selector de modo de operación

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
Ticker interrupcion1;

/* ================================================================
  VARIABLES GLOBALES PARA ERROR, PID Y LEY DE CONTROL
  ================================================================*/
float e[3]={0,0,0};                   // Vector de error (última pos. error actual)
float u[2]={0,0};                    // Vector de Ley de Control
int kU = sizeof(u)/sizeof(float)-1;  //
int kE = sizeof(e)/sizeof(float)-1; //
float volatile kp,ti,td,c0,c1,c2;           // Parámetros del PID
/*
  VARIABLES GLOBALES
*/
float v_out =0.0;       // Ley de control en voltaje (0 a 3.3v)
float H1=0.0;         // %de PWM ley de control
float senT1=0.0;     // Temperatura
float w=0;          // Referencia
int outvalue=0;    // Valor mapeado salida ley control
float temp=0.0;   // Temperatura sensada
float Pot = 0.0; // Potenciómetro para referencia física
// selector: false modo interfaz, true modo remoto
bool selectorState = false;  // Switch para seleccionar referencia física o por interfaz
/* ============================================================
  MODELO DEL SISTEMA (1er. Orden) Y TS
  ============================================================*/

float K=0.2259,tau=116.02,theta=8.04;
/*    116/20=6 y 116/10=12-> 6<Ts<12 rango de Ts de acuerdo con ZN    */
int Ts = 8;               // Periodo de Muestreo
float L = theta + Ts/2;  // Retardo aproximado en tiempo discreto ZOH
/*=========================================================================
  BUFFER PARA TRAMA DE COM. SERIAL SIMULINK-ARDUINO
  =========================================================================*/
```

```
union BtoF // Envió de temperatura T1 por serial
{
  byte b[16];
  float fval;
} up;
union BtoF2 //para recibir puerto serial
  {
  byte b2[16];
  float fval2;
  } u2;

union BtoF3 //Envió de %PWM (H1) por serial
{
  byte b3[16];
  float fval3;
} u3;
union BtoF4 // Para Envió de kp
{
  byte b4[16];
  float fval4;
} u4;

union BtoF5 // Para Envío ti
{
  byte b5[16];
  float fval5;
} u5;

union BtoF6 // Para Envío td
{
  byte b6[16];
  float fval6;
} u6;
```

```
union BtoF7 // Para Envío ref
{
  byte b7[16];
  float fval7;
} u7;
//struct FloatValues;
struct FloatValues {
  float value1;     // w
  float value2;    // kp
  float value3;   // ti
  float value4;  // td
  float value5;  // ref
  };
FloatValues instancia; // instancia o objeto tipo estruct
/*
  DECLARACIÓN DE BUS DE COMUNICACIÓN SENSOR DALLAS DS18B20
*/
OneWire oneWire(SENSOR_PIN);           // Declara pin A0 como bus de sensado analógico
DallasTemperature sensors(&oneWire);  // Crea objeto sensors para trabajar con sensor ds18b20

/*=========================================================
    INTERRUPCION 1 CADA 8 SEGUNDOS
    =========================================================*/
void interr_1_control()
{
  actualiza_vectores(u,kU);   //Actualiza vector u
  actualiza_vectores(e,kE);  //Actualiza vector e

  if(w != 0.0){ // Mientras la referencia sea 0 no ejecuta accion de control
    e[kE] = w - temp;           //Calcula el error actual: error=ref-realimentación
    // Calcula la Acción de Control PID
    u[kU] = PID_Discreto(u, e, c0, c1, c2);  //Max= 100, Min=0
    H1 = u[kU];
    outvalue = map(H1 , 0,100, 0,255);    //Mapeo de ley de control a escribir PWM
    v_out = H1/30.30;// ley de control a escribir en Voltaje
```

```
    //Aplica la acción de control en el PWM
    ledcWrite(0, outvalue); //Max= 100% PWM, Min=0% PWM

  }
  else
    {
    e[kE] = w - temp;           //Calcula el error actual: error=ref-realimentación
    // Calcula la Acción de Control PID
    u[kU] = 0; // Forzamos a 0 la ley de control
    H1 = u[kU];
    outvalue = map(H1 , 0,100, 0,255);    //Mapeo de ley de control a escribir PWM
    v_out = H1/30.30;// ley de control a escribir en Voltaje
    ledcWrite(0,outvalue); //Max= 100% PWM, Min=0% PWM
    }
}

/*=============================================================================
 |||||||||||||FUNCIÓN SETUP
 =============================================================================*/
void setup() {
  Serial.begin(115200);        // Configuramos el puerto serial baudios
  sensors.begin();            // Inicia sensado
  pinMode(POT_PIN,INPUT);    // Potenciómetro referencia física
  pinMode(ALAR_PIN,OUTPUT); // Led "Caliente" como salida
  pinMode(MODE_PIN,INPUT); // Selector de modo de operación
  //Declaración de modo de pines
  digitalWrite(ALAR_PIN,LOW);      // Inicializa apagado alarma

  // Config. de la interrupción
  interrupcion1.attach(8, interr_l_control);// Llama a la interrupción cada 8 segundos
  // Configuración PWM
  ledcSetup(0, 5000, 8);
  ledcAttachPin(V_BPIN, 0);
  //ledcWrite(0,0);          // Inicializa PWM de salida en 0%

  // Inicializar la pantalla OLED
  if(!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
    Serial.println(F("No se pudo inicializar la pantalla OLED"));
    while(true);
  }
  display.display();
  delay(2000);
  display.clearDisplay();

/*//          CONSTANTES DE SINTONIA DEL PID ZN

  kp=(1.2*tau)/(K*L); // Constante proporcional del PID
  ti=2*L;            // Constante integral del PID
  td=0.5*L;         // Constante derivativa del PID
*/
//          CONSTANTES DE SINTONIA DEL PI ZN

  kp=(0.9*tau)/(K*L); // Constante proporcional del PID
  ti=L/0.33;            // Constante integral del PID
  td=0;         // Constante derivativa del PID

  // CONSTANTES DEL CONTROLADOR PID DISCRETO

  c0=kp*(1+Ts/(2*ti)+td/Ts);
  c1=-kp*(1-Ts/(2*ti)+(2*td)/Ts);
  c2=(kp*td)/Ts;

}
```

```
/*=============================================================================
                BUCLE PRINCIPAL
=============================================================================*/
void loop() {
  sensors.requestTemperatures();        // Se solicita la lectura de la temperatura
  //delay(50);                           // Tiempo de espera para que sense correctamente
  temp = sensors.getTempCByIndex(0);  // Se obtiene la temperatura en °C;
  senT1 = filtro_pmovil(temp);       // Devuelve temperatuta filtrada digitalmente

  //Alarmas cuando supere los 40°C
  if(temp>40.0)
    digitalWrite(ALAR_PIN, HIGH); // Alarma de transistor encendida
  else
    digitalWrite(ALAR_PIN, LOW); // Alarma de transistor apagada

/*================================================
    COMUNICACIÓN SERIAL MATLAB-ESP32
  ================================================*/
if (digitalRead(MODE_PIN)==HIGH){//modo remoto
    int Val_pot = analogRead(POT_PIN);// Lectura del potenciómetro de referencia
    Pot = map(Val_pot,0,4095,20,50); // Mapear temp desde 25°C hasta 50°C máx
    w = Pot; // Asigna valor de potenciómetro físico
    display_in_OLED (temp, v_out, w);
    delay(500); // Actualizar cada 1/2 segundo
    /*
    Serial.print(w);
    Serial.print(" ");
    Serial.println(temp);
    //*/
  }else{
    //Recibir Datos por Puerto Serial
    if (Serial.available()>0)
      {
        readFromMatlab();
        w  = instancia.value1;            // Actualiza ref
        kp = instancia.value2;           // Actualiza kp
        ti = instancia.value3;          //  Actualiza ti
        td = instancia.value4;         // Actualiza td
        //float ref = instancia.value5; //  guarda ref
      }
      //Enviar Datos por Puerto Serial
      writeToMatlab(temp, v_out, kp, ti, td, u2.fval2);
      // Asignar valores obtenidos
      c0=kp*(1+Ts/(2*ti)+td/Ts);
      c1=-kp*(1-Ts/(2*ti)+(2*td)/Ts);
      c2=(kp*td)/Ts;
   }
  // delay(1000); //se envía y recibe datos cada segundo

}

void readFromMatlab()
{
  String rein = Serial.readString();
  for (int i =0; i<4; i++)
  {

    u2.b2[i] = rein[i];         // w
    u4.b4[i] = rein[i+4];      // kp
    u5.b5[i] = rein[i+8];     // ti
    u6.b6[i] = rein[i+12];   // td
  }
```

```
257
258   instancia.value1 = u2.fval2;      // guarda w
259   instancia.value2 = u4.fval4;     // guarda kp
260   instancia.value3 = u5.fval5;    // guarda ti
261   instancia.value4 = u6.fval6;   // guarda td
262 }
263
264 void writeToMatlab(float number1, float number2,float number3,float number4,float number5,float number6)
265 {
266   byte *b = (byte *) &number1;         // Apunta espacio de memoria de byte de temperatura
267   byte *b3 = (byte *) &number2;       // Apunta espacio de memoria de byte de H1
268   byte *b4 = (byte *) &number3;      // Apunta espacio de memoria de byte de kp
269   byte *b5 = (byte *) &number4;     // Apunta espacio de memoria de byte de ti
270   byte *b6 = (byte *) &number5;    // Apunta espacio de memoria de byte de td
271   //byte *b2 = (byte *) &number6;   // Apunta espacio de memoria de byte de w
272   Serial.write("A");             // caracter inicial letra A
273   Serial.write(b,4);            // Trama de temperatura (°C)
274   Serial.write(b3,4);          // Trama de ley de control (%PWM)
275   Serial.write(b4,4);         // kp
276   Serial.write(b5,4);        // ti
277   Serial.write(b6,4);       // td
278   //Serial.write(b2,4);    // ref
279   Serial.write(13);       // Envia caracter '\r'
280   Serial.write(10);      // Envia caracter final '\n'
281 }
```

```
282   void display_in_OLED (float temperatura, float voltaje, float referencia)
283 {
284   /*
285       PANTALLA OLED
286 */
287     display.clearDisplay();
288    // Mostrar la temperatura en la pantalla OLED
289     display.setTextSize(1.5);
290     display.setTextColor(SSD1306_WHITE);
291     display.setCursor(0,0);
292     display.print("Temperatura:");
293     display.setTextSize(1.5);
294     display.setCursor(75, 0);
295     display.print(temperatura, 2); // (variable, #decimales)
296     display.setTextSize(1.5);
297     display.setCursor(100, 0); //(x,y)
298     display.print(" [C]");
299     // Voltaje PWMOUT
300     display.setTextSize(1.5);
301     display.setCursor(0, 20);
302     display.print("Voltaje: ");
303     display.setTextSize(1.5);
304     display.setCursor(75, 20);
305     display.print(voltaje,2);
306     display.setTextSize(1.5);
307     display.setCursor(100, 20); //(x,y)
308     display.print(" [V]");
309
310
311     // REFERENCIA
312     display.setTextSize(1.5);
313     display.setCursor(0, 40);
314     display.print("SetPoint: ");
315     display.setTextSize(1.5);
316     display.setCursor(75, 40);
317     display.print(referencia,2);
318     display.setTextSize(1.5);
319     display.setCursor(100, 40); //(x,y)
320     display.print(" [C]");
321
322     display.display();
323   }
```

APÉNDICE E.

Figura E.1. Código en IDE de Arduino para control PID de temperatura por sintonización Ziegler-Nichols en ESP32.

PT_ESP32_ZN_OLED.ino | lib_pid_zn.cpp | lib_pid_zn.h

```
/*  ================================================
    CONTROLADOR PID POR SINTONIZACIÓN ZIEGLER-NICHOLS
        By Jesús Martínez & Javier Nicolalde
          PLANTA DE CONTROL DE TEMPERATURA
            Implementación ESP32
          Última revisión: 23/08/2023
    ================================================*/

//                    LIBRERIAS
#include <OneWire.h>
#include <DallasTemperature.h>  // Libreria de temperatura ds18b20
#include <Ticker.h>             // Libreria de interrupciones
#include "lib_pid_zn.h"         // Libreria usada para funciones PID ZN
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

// Definiciones componentes de la tarjeta
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET    -1
#define SCREEN_ADDRESS 0x3C
#define SENSOR_PIN 0  //sensor DS18B20
#define POT_PIN 2   // Potenciómetro referencia física
#define V_BPIN 16   // v_base transistor TIP31C
#define ALAR_PIN 18  //Led para indicar "alarma de caliente transistor"
#define MODE_PIN 23 // Selector de modo de operación

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
Ticker interrupcion1;

/*
  VARIABLES GLOBALES PARA ERROR, PID Y LEY DE CONTROL
*/
float e[3]={0,0,0};                   // Vector de error (última pos. error actual)
float u[2]={0,0};                    // Vector de Ley de Control
int kU = sizeof(u)/sizeof(float)-1;  //
int kE = sizeof(e)/sizeof(float)-1; //
float volatile kp,ti,td,c0,c1,c2;           // Parámetros del PID
/*
  VARIABLES GLOBALES
*/
//int conteo = 0;       // Contador
float H1=0.0;          // %de PWM ley de control
float v_out =0.0;       // Ley de control en voltaje (0 a 3.3v)
float senT1=0.0;     // Temperatura
float w=0;          // Referencia
int outvalue=0;    // Valor mapeado salida ley control
//int outvalue1=0;    // Valor mapeado salida ley control en voltaje
float temp=0.0;   // Temperatura sensada
float Pot = 0.0; // Potenciómetro para referencia física
// selector: false modo interfaz, true modo remoto
bool selectorState = false;  // Switch para seleccionar referencia física o por interfaz
```

```
/*
| | | | | | | | | MODELO DEL SISTEMA Y TS
| | Nota: Esto es lo que se modifica al tener planta diferente
*/

float K=0.2259,tau=116.02,theta=8.04;
/*    116/20=6 y 116/10=12-> 6<Ts<12 rango de Ts de acuerdo con ZN    */
int Ts = 8;               // Periodo de Muestreo
float L = theta + Ts/2;  // Retardo aproximado en tiempo discreto
/*
| | | VARIABLES PARA TRAMA DE COMUNICACIÓN
*/
const int buffer_size =8;    // Tamaño del buffer
byte buf[buffer_size];     // Crea un byte buffer de 8 bytes

union BtoF // Envió de temperatura T1 por serial
{
| byte btemp[8];
| float fvaltemp;
} tempBorF;
union BtoF2 // Envió depwm H1 por serial
{
| byte bH1[8];
| float fvalH1;
} H1BorF;
union BtoF3 // Recepción de referencia w por serial
{
| byte bw[8];
| float fvalw;
} wBorF;


//struct FloatValues;
struct FloatValues {
| float v_w;      // w
| float v_temp;  // temp
| float v_pwm;  // pwm
| };
FloatValues instancia; // instancia o objeto tipo estruct

/*
| | DECLARACIÓN DE BUS DE COMUNICACIÓN SENSOR DALLAS DS18B20
*/
OneWire oneWire(SENSOR_PIN);          // Declara pin A0 como bus de sensado analógico
DallasTemperature sensors(&oneWire);  // Crea objeto sensors para trabajar con sensor ds18b20

 /*
 | | | | INTERRUPCION 1 CADA 8 SEGUNDOS
 */
 void interr_1_control()
 {
 | actualiza_vectores(u,kU);   //Actualiza vector u
 | actualiza_vectores(e,kE);  //Actualiza vector e
 |
 | if(w != 0.0){ // Mientras la referencia sea 0 no ejecuta accion de control
 | | e[kE] = w - temp;           //Calcula el error actual: error=ref-realimentación
 | | // Calcula la Acción de Control PID
 | | u[kU] = PID_Discreto(u, e, c0, c1, c2);  //Max= 100, Min=0
 | | H1 = u[kU];
 | | outvalue = map(H1 , 0,100, 0,255);    //Mapeo de ley de control a escribir PWM
 | | v_out = H1/30.30;// ley de control a escribir en Voltaje
 | | //Aplica la acción de control en el PWM
 | | ledcWrite(0, outvalue); //Max= 100% PWM, Min=0% PWM
 | |
 | }
```

```
  else
    {
    e[kE] = w - temp;           //Calcula el error actual: error=ref-realimentación
    // Calcula la Acción de Control PID
    u[kU] = 0; // Forzamos a 0 la ley de control
    H1 = u[kU];
    outvalue = map(H1 , 0,100, 0,255);    //Mapeo de ley de control a escribir PWM
    v_out = H1/30.30;// ley de control a escribir en Voltaje
    ledcWrite(0,outvalue); //Max= 100% PWM, Min=0% PWM
    }
 }


/*=========================================================================
 ||||||||||||||FUNCIÓN SETUP
 =========================================================================*/
void setup() {
  Serial.begin(115200);        // Configuramos el puerto serial baudios
  sensors.begin();            // Inicia sensado
  pinMode(ALAR_PIN,OUTPUT);         // Led "Caliente" como salida
  //Declaración de modo de pines
  digitalWrite(ALAR_PIN,LOW);      // Inicializa apagado alarma
  pinMode(POT_PIN,INPUT);    // Potenciómetro referencia física
  pinMode(MODE_PIN,INPUT); // Selector de modo de operación

  // Config. de la interrupción
  interrupcion1.attach(Ts, interr_1_control);// Llama a la interrupción cada 8 segundos
  // Configuración PWM
  ledcSetup(0, 5000, 8);      // Canal 0, frec =5000Hz, 8 bits
  ledcAttachPin(V_BPIN, 0);  // FIja la salida al canal 0

  // Inicializar la pantalla OLED
  if(!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
    Serial.println(F("No se pudo inicializar la pantalla OLED"));
    while(true);
  }
  display.display();
  delay(2000);
  display.clearDisplay();

/*//          CONSTANTES DE SINTONIA DEL PID ZN

  kp=(1.2*tau)/(K*L); // Constante proporcional del PID
  ti=2*L;            // Constante integral del PID
  td=0.5*L;         // Constante derivativa del PID
*/
//          CONSTANTES DE SINTONIA DEL PI ZN

  kp=(0.9*tau)/(K*L); // Constante proporcional del PID
  ti=L/0.33;            // Constante integral del PID
  td=0;         // Constante derivativa del PID

  // CONSTANTES DEL CONTROLADOR PID DISCRETO

  c0=kp*(1+Ts/(2*ti)+td/Ts);
  c1=-kp*(1-Ts/(2*ti)+(2*td)/Ts);
  c2=(kp*td)/Ts;

}
```

```
/*=============================================================================
                              BUCLE PRINCIPAL
=============================================================================*/
void loop() {

  sensors.requestTemperatures();        // Se solicita la lectura de la temperatura
  delay(50);                            // Tiempo de espera para que sense correctamente
  temp = sensors.getTempCByIndex(0);  // Se obtiene la temperatura en °C;

  senT1 = filtro_pmovil(temp);         // Devuelve temperatuta filtrada digitalmente

  //Serial.println(Pot);
  if(senT1>40.0)
    digitalWrite(ALAR_PIN, HIGH); // Alarma de transistor encendida
  else
    digitalWrite(ALAR_PIN, LOW); // Alarma de transistor apagada

/*================================================
    COMUNICACIÓN SERIAL MATLAB-ESP32
  ================================================*/
  if (digitalRead(MODE_PIN)==HIGH){//modo remoto
    int Val_pot = analogRead(POT_PIN);// Lectura del potenciómetro de referencia
    Pot = map(Val_pot,0,4095,20,50); // Mapear temp desde 20°C hasta 50°C máx
    w = Pot; // Asigna valor de potenciómetro físico
    display_in_OLED (senT1, v_out, w);
    delay(500); // Actualizar cada 1/2 segundo

    /*
    Serial.print(w);
    Serial.print(" ");
    Serial.println(senT1);
    //*/
  }else{
    // Recibir Datos por Puerto Serial
    if (Serial.available()>0)
    {
      readFromMatlab(); //
      w = instancia.v_w;   //
      }
    //Enviar Datos por Puerto Serial
    writeToMatlab(senT1,v_out);//H1);
  }
  //delay(1000); //se envia y recibe datos cada segundo

}
```

```
224  void readFromMatlab()
225  {
226  //  buf2 de tamaño 8
227   String rein = Serial.readString();
228   for (int i =0; i<4; i++) //recorre los 4 bits
229   {
230   wBorF.bw[i] = rein[i];         // referencia w
231   }
232   instancia.v_w = wBorF.fvalw;
233   instancia.v_temp = senT1;
234   instancia.v_pwm = H1;
235
236   }
237
238  void writeToMatlab(float tempT1, float vout)//float pwmH1)
239  {
240   byte *btemp = (byte *) &tempT1;         //
241   byte *bH1   = (byte *) &vout;          //
242   Serial.write("A");             // caracter inicial letra A
243   Serial.write(btemp,4);            // Trama de temperatura (°C)
244   Serial.write(bH1,4);          // Trama de ley de control (%PWM)
245   Serial.write(13);      // Envia caracter '\r'
246   Serial.write(10);     // Envia caracter final '\n'
247  }
248
249  void display_in_OLED (float temperatura, float voltaje, float referencia)
250  {
251   /*
252   | | | PANTALLA OLED
253  */
254   | | display.clearDisplay();
255   // Mostrar la temperatura en la pantalla OLED
256    display.setTextSize(1.5);
257    display.setTextColor(SSD1306_WHITE);
258    display.setCursor(0,0);
259    display.print("Temperatura:");
260    display.setTextSize(1.5);
261    display.setCursor(75, 0);
262    display.print(temperatura, 2); // (variable, #decimales)
263    display.setTextSize(1.5);
264    display.setCursor(100, 0); //(x,y)
265    display.print(" [C]");
266    // Voltaje PWMOUT
267    display.setTextSize(1.5);
268    display.setCursor(0, 20);
269    display.print("Voltaje: ");
270    display.setTextSize(1.5);
271    display.setCursor(75, 20);
272    display.print(voltaje,2);
273    display.setTextSize(1.5);
274    display.setCursor(100, 20); //(x,y)
275    display.print(" [V]");
276
277
278    // REFERENCIA
279    display.setTextSize(1.5);
280    display.setCursor(0, 40);
281    display.print("SetPoint: ");
282    display.setTextSize(1.5);
283    display.setCursor(75, 40);
284    display.print(referencia,2);
285    display.setTextSize(1.5);
286    display.setCursor(100, 40); //(x,y)
287    display.print(" [C]");
288
289    display.display();
290  }
```

APÉNDICE F.

Figura F.1. Código en IDE de Arduino para control de temperatura Fuzzy-Mamdani en ESP32.

PT_ESP32_FUZZY_OLED.ino | fuzzy.cpp | fuzzy.h | tsclab_lib.cpp | tsclab_lib.h

```
/*  ================================================================
    CONTROLADOR FUZZY - MAMDANI CONTROL DE TEMPERATURA
    By Jesús Martínez & Javier Nicolalde
    Implementación ESP32
    Laboratorio TSCLAB

    Última revisión: 27/08/2023
    ================================================================*/
//                      LIBRERIAS
#include <OneWire.h>
#include <DallasTemperature.h> //Librería de temperatura ds18b20
#include <Ticker.h>           // Librería de interrupciones
#include "tsclab_lib.h"      // Librería para funciones tsclab
#include "fuzzy.h"          // Librería para funciones control Fuzzy-Mamdani
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

// Definiciones componentes de la tarjeta
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET    -1
#define SCREEN_ADDRESS 0x3C // SDA = GPIO21, SCL = GPIO22

#define SENSOR_PIN 0 // Pin del sensor DS18B20
#define LED_ALARM 18   // Pin del LED 1
#define V_BPIN 16   // Pin del TIP31
#define POT_PIN 2   // Potenciómetro referencia física
#define MODE_PIN 23 // Selector de modo de operación
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
Ticker interrupcion1;

/*================================================================================
  VARIABLES GLOBALES PARA ERROR Y LEY DE CONTROL
  ================================================================================*/
float v_out =0.0;       // Ley de control en voltaje (0 a 3.3v)
float H1=0.0;             // %PWM V_Base TIP31
float temp=0.0;          // Temperatura
float w=0;              // Referencia
float e[2]={0,0};      // Vector de error
float u[2]={0,0};     // Vector de Ley de Control
float Pot = 0.0; // Potenciómetro para referencia física
// selector: false modo interfaz, true modo remoto
bool selectorState = false;  // Switch para seleccionar referencia física o por interfaz

int kU = sizeof(u)/sizeof(float)-1;    // kU es la pos. de la ley de control actual
int kE = sizeof(e)/sizeof(float)-1;   // e[kE] es el error actual
```

```
/*=============================================================================
    DEFINICIONES GLOBALES CONTROL FUZZY-MAMDANI
      UNIVERSO DEL DISCURSO
    ============================================================================*/
//discurso_de = 1 => visualizar experimentalmente*
float discurso_e = 100, discurso_de = 4, discurso_u = 3;
float der;       // Almacenar derivada del  discreta
float lu;       // Almacenar ley de control
int Ts = 8;    // Periodo de Muestreo para calculo de derivada discreta y periodo de interrupción
const int buffer_size =8;    // Tamaño del buffer
byte buf[buffer_size];     // Crea un byte buffer de 16 bytes

// Estructuras para tratar una variable como byte o float a la vez
union BtoF // Para almacenar temperatura T1 por serial
{
  byte btemp[8];
  float fvaltemp;
} tempBorF;
union BtoF2 // Para almacenar pwm (H1) por serial

{
  byte bH1[8];
  float fvalH1;
} H1BorF;
union BtoF3 // Para almacenar referencia (w) por serial
{
  byte bw[8];
  float fvalw;
} wBorF;

//Estructura que guarda parámetros
struct FloatValues {
  float v_w;        // w (ref)
  float v_temp;    // temp
  float v_pwm;    // pwm
  };
FloatValues instancia; // instancia o objeto tipo estruct

/*=============================================================================
  DECLARACIÓN DE BUS DE COMUNICACIÓN SENSOR DALLAS DS18B20
  ============================================================================*/
OneWire oneWire(SENSOR_PIN);          // Declara pin A0 como bus de sensado analógico
DallasTemperature sensors(&oneWire);  // Crea objeto sensors para trabajar con sensor ds18b20

```

```
/*=============================================================================
    INTERRUPCION CADA 8 SEGUNDOS ACTUALIZACIÓN
    =============================================================================*/
void interr_1_control()
{
  actualiza_vectores(u,kU);   //Actualiza vector u
  actualiza_vectores(e,kE);  //Actualiza vector e

  e[kE] = w - temp;        //Calcula el error actual: error=ref-realimentación

  // Satura el error para estar dentro del universo del discurso
  if(e[kE]<-1*discurso_e+0.1)   // Limitar a -99.9
    e[kE]=-1*discurso_e+0.1;   // Satura a -99.9
  if(e[kE]>discurso_e-0.1)    // Limitar a 99.9
    e[kE]=discurso_e-0.1;    // Satura a -99.9

  // Calcula la derivada del error discreta
  der=(e[kE]-e[kE-1])/Ts; //fórmula de aprox. de una derivada discreta

  // Satura la derivada del error para estar dentro del universo del discurso
  if(der<-1*discurso_de+0.01)   // Limitar de a -3.99
    der=-1*discurso_de+0.01;   // Saturar a -3.99
  if(der>discurso_de-0.01)    // Limitar de a 3.99
    der=discurso_de-0.01;    // Saturar a 3.99

  // Ley de control fuzzy
  // Parámetros: (error_actual,derivada_error,Univ_disc_e,Univ_disc_de, Univ_disc_leycontrol)
  lu=myfuzzy(e[kE],der,discurso_e,discurso_de,discurso_u);

  //Integral Colocada en salida de u
  // lu*10 dado que lu es muy pequeño, para "fortalecer el control"
  u[kU] = lu*10 + u[kU-1]; //accion de control tiempo presente

  // Anti - Windup (Saturación de ley de control %pwm)
  if (u[kU] >= 100.0)
    u[kU] = 100.0;
  if (u[kU] <= 0.0)
    u[kU] = 0.0;

  H1 = u[kU];      // Actualiza el %PWM correspondiente
  //Aplica la acción de control en el PWM
  v_out = H1/30.30;// ley de control a escribir en Voltaje
  ledcWrite(0,map(H1 , 0,100, 0,255)); //Max= 100, Min=0
}

/*=============================================================================
    FUNCIÓN SETUP
    =============================================================================*/
void setup() {
  Serial.begin(115200);           // Configuramos el puerto serial baudios
  sensors.begin();               // Inicia sensado
  pinMode(LED_ALARM,OUTPUT);    //Led "Caliente" como salida
  pinMode(POT_PIN,INPUT);    // Potenciómetro referencia física
  pinMode(MODE_PIN,INPUT); // Selector de modo de operación

  digitalWrite(LED_ALARM,LOW); // Inicializa apagado alarma
  // Config. de la interrupción
  // Inicializar la pantalla OLED por I2C
  if(!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
    Serial.println(F("No se pudo inicializar la pantalla OLED"));
    while(true);
  }
  display.display();
  delay(2000);
  display.clearDisplay();

  interrupcion1.attach(Ts, interr_1_control);// Llama a la interrupción cada 8 segundos
  // Configuración PWM
  ledcSetup(0, 5000, 8);
  ledcAttachPin(V_BPIN, 0);

}
```

```
164
165   /*=============================================================================
166   |||||||||||||||FUNCIÓN PRINCIPAL
167   =============================================================================*/
168   void loop() {
169     sensors.requestTemperatures();       // Se solicita la lectura de la temperatura
170     //delay(50);                          // Tiempo de espera para que sense correctamente
171     temp = sensors.getTempCByIndex(0); // Se obtiene la temperatura en °C;
172     temp = filtro_pmovil(temp);       // Devuelve temperatuta filtrada digitalmente
173
174   //Indicación Luminica
175     if(temp>40.0)
176     | digitalWrite(LED_ALARM, HIGH); // Alarma de transistor encendida
177     else
178     | digitalWrite(LED_ALARM, LOW); // Alarma de transistor apagada
179   /*=============================================================================
180   | | | COMUNICACIÓN SERIAL MATLAB-ARDUINO
181   =============================================================================*/
182   if (digitalRead(MODE_PIN)==HIGH){//modo remoto
183     | int Val_pot = analogRead(POT_PIN);// Lectura del potenciómetro de referencia
184     | Pot = map(Val_pot,0,4095,20,50); // Mapear temp desde 20°C hasta 50°C máx
185     | w = Pot; // Asigna valor de potenciómetro físico
186     | display_in_OLED (temp, v_out, w);
187     | delay(500); // Actualizar cada 1/2 segundo
188     | /*
189     | Serial.print(w);
190     | Serial.print(" ");
191     | Serial.println(temp);
192     | //*/
193     }else{
194     //Recibir datos por puerto serial
195     | if (Serial.available()>0)
196     | {
197     | | readFromMatlab();     //Lectura desde Matlab Serial
198     | | w = instancia.v_w;   //Actualiza referencia
199     | | }
200     |
201     | /* Enviar Datos por Puerto Serial  */
202     | writeToMatlab(temp, v_out);
203     }
204
205     //delay(1000); //Tiempo de envio/recepción serial.
206
207   }
```

```
/*============================================================================
 |||||  FUNCIONES AUXILIARES PARA COMUNICACIÓN SERIAL
 ============================================================================*/
//  Función leer por com. serial
void readFromMatlab()
{
 //  buf2 de tamaño 8
  String rein = Serial.readString(); //Lectura desde Matlab serial
  for (int i =0; i<4; i++)
  {
   wBorF.bw[i] = rein[i];         // Almaceno referencia w recibida
  }
  instancia.v_w = wBorF.fvalw;
  instancia.v_temp = temp;
  instancia.v_pwm = H1;

 }
// Función para escribir por com. serial
void writeToMatlab(float tempT1, float pwmH1)
{
  //La trama se envia en bytes
  byte *btemp = (byte *) &tempT1;  // Guarda en btemp la temp como tipo byte
  byte *bH1   = (byte *) &pwmH1;  // Guarda en bH1 el control pwm como tipo byte
  Serial.write("A");             // Caracter inicial letra A (ASCII = 65)
  Serial.write(btemp,4);        // Trama de temperatura (°C)
  Serial.write(bH1,4);         // Trama de ley de control (%PWM)
  Serial.write(13);           // Envia caracter '\r' (ASCII = 13)
  Serial.write(10);          // Envia caracter final '\n' (ASCII = 10)
}
```

```
239  void display_in_OLED (float temperatura, float voltaje, float referencia)
240  {
241    /*
242        PANTALLA OLED
243  */
244      display.clearDisplay();
245     // Mostrar la temperatura en la pantalla OLED
246      display.setTextSize(1.5);
247      display.setTextColor(SSD1306_WHITE);
248      display.setCursor(0,0);
249      display.print("Temperatura:");
250      display.setTextSize(1.5);
251      display.setCursor(75, 0);
252      display.print(temperatura, 2); // (variable, #decimales)
253      display.setTextSize(1.5);
254      display.setCursor(100, 0); //(x,y)
255      display.print(" [C]");
256      // Voltaje PWMOUT
257      display.setTextSize(1.5);
258      display.setCursor(0, 20);
259      display.print("Voltaje: ");
260      display.setTextSize(1.5);
261      display.setCursor(75, 20);
262      display.print(voltaje,2);
263      display.setTextSize(1.5);
264      display.setCursor(100, 20); //(x,y)
265      display.print(" [V]");
266
267
268      // REFERENCIA
269      display.setTextSize(1.5);
270      display.setCursor(0, 40);
271      display.print("SetPoint: ");
272      display.setTextSize(1.5);
273      display.setCursor(75, 40);
274      display.print(referencia,2);
275      display.setTextSize(1.5);
276      display.setCursor(100, 40); //(x,y)
277      display.print(" [C]");
278
279      display.display();
280  }
```

APÉNDICE G.

Figura G.1. Código en IDE de Arduino para control de temperatura por Realimentación de Estados en ESP32.

PT_ESP32_RE_OLED.ino | tclab_lib.cpp | tclab_lib.h

```
/*  ====================================================================
    CONTROLADOR POR REALIMENTACIÓN DE ESTADOS PARA CONTROL DE TEMPERATURA
    By Jesús Martínez & Javier Nicolalde
    Implementación tarjeta ESP32
    Laboratorio TSCLAB
    Última revisión: 23/08/2023
    ====================================================================*/

//                    LIBRERIAS
#include <OneWire.h>
#include <DallasTemperature.h>        // Librería de temperatura ds18b20
#include <Ticker.h>                   // Librería de interrupciones
#include "tclab_lib.h"               // Librería usada para funciones necesarias
#include <BasicLinearAlgebra.h>     // Librería para algebra lineal
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

// Definiciones componentes de la tarjeta
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET    -1
#define SCREEN_ADDRESS 0x3C
#define POT_PIN 2          // Potenciómetro referencia física
#define SENSOR_PIN 0      // Pin del sensor DS18B20
#define LED_ALARM 18     // Pin del LED Alarma
#define V_BPIN 16       // Pin del control PWM TIP31
#define MODE_PIN 23 // Selector de modo de operación
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
Ticker interrupcion1;   // Crear objeto de interrupción

/*=====================================================================
    VARIABLES GLOBALES
    =====================================================================*/
float v_out =0.0;          // Ley de control en voltaje (0 a 3.3v)
int outvalue =0;          //mapeo de H1
float H1=0.0;            // pwmOUT
float temp=0.0;         // temperatura
float w=0;             //Referencia
float v[2]={0,0};     // Vector Integrador
float u[2]={0,0};    //Vector de Ley de Control
int kU = sizeof(u)/sizeof(float)-1;  //cantidad de elem. de u
int kV = sizeof(v)/sizeof(float)-1; //cantidad de elem. de v
float Pot = 0.0; // Potenciómetro para referencia física
// selector: false modo interfaz, true modo remoto
bool selectorState = false;  // Switch para seleccionar referencia física o por interfaz

```

```
48  /* ============================================================================
49  |||||||| MODELO DEL SISTEMA Y TS
50  ============================================================================*/
51
52  float K=0.2259,tau=116.02,theta=8.04; //Modelo planta temperatura 1er. Orden
53  int Ts = 8;                          //Periodo de Muestreo
54  /*============================================================================
55  | VARIABLES PARA TRAMA DE COMUNICACIÓN SERIAL
56  ============================================================================*/
57  const int buffer_size =8;    // Tamaño del buffer
58  byte buf[buffer_size];      // Crea un byte buffer de 8 bytes
59  // variables para tratar datos como byte o como float
60  union BtoF // Variable temperatura T1 por serial
61  {
62  | byte btemp[8];
63  | float fvaltemp;
64  } tempBorF;
65  union BtoF2 // Variable depwm H1 por serial
66  {
67  | byte bH1[8];
68  | float fvalH1;
69  } H1BorF;
70  union BtoF3 // Variable referencia w por serial
71  {
72  | byte bw[8];
73  | float fvalw;
74  } wBorF;
75  //Estructura para guardar parámetros
76  struct FloatValues {
77  | float v_w;       // w (ref)
78  | float v_temp;    // temp
79  | float v_pwm;    // pwm
80  | };
81  FloatValues instancia; // instancia o objeto tipo estruct
82
83  // Se usa "namespace BLA" para trabajar con matrices
84  using namespace BLA;
85  BLA::Matrix<3,1> Xm;       // Vector de estados
86  BLA::Matrix<3,1> dXm;     // Derivada del vector de estados
87  BLA::Matrix<3,3> Ad;     // Matriz A Discreta
88  BLA::Matrix<3,1> Bd;    // Vector B discreto
89  BLA::Matrix<1,3> K1;   // Realimentación de Estados
90  BLA::Matrix<1,1> K2;  // Ganancia Integrador
91
92  /*============================================================================
93  | DECLARACIÓN DE BUS DE COMUNICACIÓN SENSOR DALLAS DS18B20
94  ============================================================================*/
95  OneWire oneWire(SENSOR_PIN);          // Declara pin A0 como bus de sensado analógico
96  DallasTemperature sensors(&oneWire);  // Crea objeto sensors para trabajar con sensor ds18b20
97
```

```
/*=============================================================================
        INTERRUPCIÓN  CADA 8 SEGUNDOS
 =============================================================================*/
void interr_l_control(void)
{
  BLA::Matrix<1> lu;

  //Sistema
  dXm = Ad*Xm + Bd* u[kU-1];
  /*Serial.print("Sistema dXm(0,0): ");
  Serial.println(dXm(0,0));
  Serial.println(dXm(1,0));
  Serial.println(dXm(2,0)); */

  //Integrador
  v[kV] = v[kV-1] + w - temp;
  /*
  Serial.print("Integrador v[kV]: ");
  Serial.println(v[kV]); */
  // Anti - Windup

  if (v[kU] >= 600.0)
    v[kU] = 600.0;

   if (v[kU] <= 0.0)
    v[kU] = 0.0;


  // Calcula la Acción de Control
  lu = -K1*dXm + K2*v[kV];
  u[kU] = lu(0);
  /*
  Serial.print("ley de control u[kU]: ");
  Serial.println(u[kU]);*/
  // Saturación de ley de control
   if (u[kU] >= 100.0)        // Max. 100% PWM
    u[kU] = 100.0;

   if (u[kU] <= 0.0)         // Min. 0 %PWM
    u[kU] = 0.0;

  H1 = u[kU]; // Actualiza %PWM a enviar a TIP31
  outvalue = map(H1 , 0,100, 0,255);
  v_out = H1/30.30;// ley de control a escribir en Voltaje
  //Aplica la acción de control en el PWM
  ledcWrite(0,outvalue); //Max= 100, Min=0

  actualiza_vectores(u,kU);   //Actualiza vector u
  actualiza_vectores(v,kV);  //Actualiza vector v
  Xm = dXm;
  /*
  Serial.print("temp: ");
   Serial.println(temp);
   Serial.print("ref: ");
   Serial.println(w);    */
}

```

```
155  /*=============================================================================
156  |||||||||||||||FUNCIÓN SETUP
157  =============================================================================*/
158  void setup() {
159  |
160  |  Serial.begin(115200);            //Configuramos el puerto serial
161  |  sensors.begin();                // Inicia sensado
162  |  pinMode(LED_ALARM,OUTPUT);     // Led "Caliente" como salida
163  |  digitalWrite(LED_ALARM,LOW);  // Inicializa apagado alarma
164  |  pinMode(POT_PIN,INPUT);      // Potenciómetro referencia física
165  |  pinMode(MODE_PIN,INPUT); // Selector de modo de operación
166  |  // Config. de la interrupción
167  |  interrupcion1.attach(8, interr_1_control);
168  |  // Configuración PWM
169  |  ledcSetup(0, 5000, 8);      // Canal 0, Frec. 5000Hz, Resolución 8 bits.
170  |  ledcAttachPin(V_BPIN, 0);  // Asigna canal 0 a pin de salida 16
171  |
172  |  // Inicializar la pantalla OLED
173  |  if(!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
174  |  |  Serial.println(F("No se pudo inicializar la pantalla OLED"));
175  |  |  while(true);
176  |  }
177  |  display.display();
178  |  delay(2000);
179  |  display.clearDisplay();
180  |
181  |  //Inicializa vectores en 0
182  |  Xm.Fill(0);
183  |  dXm.Fill(0);
184  /*=============================================================================
185  |||||  ASIGNACIÓN DE VALORES DE VECTORES K1, K2, Y MATRICES Ad, Bd
186  |  =============================================================================*/
187  /*
188  |  Nota:
189  |||||  Estos vectores y matrices provienen del diseño en espacio de estados.
190  |||||  El diseñador establece su controlador con la función de transferencia
191  |||||  de planta y luego halla K1, K2, Ad, Bd e inicializa en esta sección.
192  */
193  // Para la planta de temperatura es un sistema de 1er Orden con:
194  // Vector K1
195  |  K1(0,0)= 0.0256;
196  |  K1(0,1)= -0.0387;
197  |  K1(0,2)= 1.0;
198  // Vector K2
199  |  K2(0,0)= 2.2645;
200  // Matriz Ad
201  |  Ad(0,0)= 0.9334;
202  |  Ad(0,1)= 0.0373;
203  |  Ad(0,2)= 7.6931;
204  |  Ad(1,0)= 0;
205  |  Ad(1,1)= 0;
206  |  Ad(1,2)= 1.0000;
207  |  Ad(2,0)=0;
208  |  Ad(2,1)=0;
209  |  Ad(2,2)=0;
210  // Vector Bd
211  |  Bd(0,0)= 0;
212  |  Bd(0,1)= 0;
213  |  Bd(0,2)=1.0000;
214  }
215
```

```
/*=========================================================================
                FUNCIÓN PRINCIPAL
=========================================================================*/
void loop() {

  sensors.requestTemperatures();       // Se solicita la lectura de la temperatura
  //delay(50);                          // Tiempo de espera para que sense correctamente
  temp = sensors.getTempCByIndex(0); // Se obtiene la temperatura en °C;
  temp = filtro_pmovil(temp);       // Devuelve temperatuta filtrada digitalmente

  //Indicación Luminica
  if(temp>40.0)
    digitalWrite(LED_ALARM, HIGH); // Transistor Caliente
  else
    digitalWrite(LED_ALARM, LOW); // Transistor Frio

/* ================================================
    COMUNICACIÓN SERIAL MATLAB-ARDUINO
================================================*/
  if (digitalRead(MODE_PIN)==HIGH){//modo remoto
    int Val_pot = analogRead(POT_PIN);// Lectura del potenciómetro de referencia
    Pot = map(Val_pot,0,4095,20,50); // Mapear temp desde 20°C hasta 50°C máx
    w = Pot; // Asigna valor de potenciómetro físico
    display_in_OLED (temp, v_out, w);
    delay(500); // Actualizar cada 1/2 segundo
    /*
    Serial.print(w);
    Serial.print(" ");
    Serial.println(temp);
    //*/
  }else{
    //Recibir datos por puerto serial
    if (Serial.available()>0)
    {
      readFromMatlab();     // Lectura serial desde Matlab
      w = instancia.v_w;   // Actualiza referencia
      }
    // Enviar Datos por Puerto Serial
    writeToMatlab(temp, v_out);
  }
  //delay(1000); // envia y recibe por serial cada 1s
}
/* ====================================================================
     FUNCIONES AUXILIARES PARA COM. SERIAL
====================================================================*/
// Leer desde Matlab por com. serial
void readFromMatlab()
{
  //  buf2 de tamaño 8
   String rein = Serial.readString(); // Recibe en string datos de serial
   for (int i =0; i<4; i++)          // Lee 4 bits
   {
     wBorF.bw[i] = rein[i];        // referencia w
   }
   instancia.v_w = wBorF.fvalw;
   instancia.v_temp = temp;
   instancia.v_pwm = H1;

  }
// Escribir por serial hacia Interfaz gráfica
void writeToMatlab(float tempT1, float pwmH1)
{
  byte *btemp = (byte *) &tempT1;    // Guarda en btemp la temp
  byte *bH1   = (byte *) &pwmH1;    // Guarda en bH1 el pwmOut
  Serial.write("A");               // caracter inicial letra A (ASCII=65)
  Serial.write(btemp,4);          // Trama de temperatura (°C)
  Serial.write(bH1,4);           // Trama de ley de control (%PWM)
  Serial.write(13);      // Envia caracter '\r'
  Serial.write(10);     // Envia caracter final '\n'
}

```

```
287  void display_in_OLED (float temperatura, float voltaje, float referencia)
288  {
289    /*
290      PANTALLA OLED
291  */
292    display.clearDisplay();
293    // Mostrar la temperatura en la pantalla OLED
294      display.setTextSize(1.5);
295      display.setTextColor(SSD1306_WHITE);
296      display.setCursor(0,0);
297      display.print("Temperatura:");
298      display.setTextSize(1.5);
299      display.setCursor(75, 0);
300      display.print(temperatura, 2); // (variable, #decimales)
301      display.setTextSize(1.5);
302      display.setCursor(100, 0); //(x,y)
303      display.print(" [C]");
304      // Voltaje PWMOUT
305      display.setTextSize(1.5);
306      display.setCursor(0, 20);
307      display.print("Voltaje: ");
308      display.setTextSize(1.5);
309      display.setCursor(75, 20);
310      display.print(voltaje,2);
311      display.setTextSize(1.5);
312      display.setCursor(100, 20); //(x,y)
313      display.print(" [V]");
314
315
316      // REFERENCIA
317      display.setTextSize(1.5);
318      display.setCursor(0, 40);
319      display.print("SetPoint: ");
320      display.setTextSize(1.5);
321      display.setCursor(75, 40);
322      display.print(referencia,2);
323      display.setTextSize(1.5);
324      display.setCursor(100, 40); //(x,y)
325      display.print(" [C]");
326
327    display.display();
328  }
```

APÉNDICE H.

Figura H.1. Código en IDE de Arduino para control PID de velocidad por Prueba-Error en ESP32.

pid_PE_ESP32PV_vout.ino | lib_pid.cpp | lib_pid.h

```
/*  ================================================
    CONTROLADOR PID POR SINTONIZACIÓN PRUEBA Y ERROR
      By Jesús Martínez & Javier Nicolalde
        PLANTA DE CONTROL DE VELOCIDAD
          Implementación ESP32
            Planta TSCLAB
        Última revisión: 23/08/2023
    ================================================*/
// LIBRERÍAS
#include <Ticker.h>           // Librería de interrupciones
#include "lib_pid.h"
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
Ticker interrupcion1;

// Definiciones componentes de la tarjeta
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET    13
#define SCREEN_ADDRESS 0x3C

#define ENCODER_PIN 27    // Encoder Digital (pin2 Arduino)
#define ENABLE1_PIN 32   // Habilitador L293D (pin3 Arduino)
#define MOTOR_PIN1 33   // Direccion 1 de motor (pin10 Arduino)
#define MOTOR_PIN2 25  // Direccion 2 de motor (pin4 Arduino)
#define LED_PIN 18    // Led indicador
#define SETPOINT_PIN 2  // Potenciómetro para referencia física
#define MODE_PIN 23 // Selector de modo de operación
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

/* ================================================================
   VARIABLES GLOBALES PARA ERROR, PID Y LEY DE CONTROL
   ================================================================*/
float e[3]={0,0,0};                   // Vector de error (última pos. error actual)
float u[2]={0,0};                    // Vector de Ley de Control
int kU = sizeof(u)/sizeof(float)-1;  //
int kE = sizeof(e)/sizeof(float)-1; //
float kp,ti,td,c0,c1,c2;           // Parámetros del PID
float Pot = 0.0;                  // Potenciómetro para referencia física
// selector: false modo interfaz, true modo remoto
bool selectorState = false;  // Switch para seleccionar referencia física o por interfaz

/*
  VARIABLES GLOBALES
*/
float v_out =0.0;       // Ley de control en voltaje (0 a 3.3v)
float pwmOut=0.0;     // %de PWM ley de control
float revpm=0.0;     // Velocidad en rpm
float ref=0;        // Referencia
int mpwmOut=0;     // Valor mapeado salida ley control
```

```
52
53    int Ts = 1;               // Periodo de Muestreo
54    /*=========================================================================
55    | | | | | | | | | | BUFFER PARA TRAMA DE COM. SERIAL SIMULINK-ARDUINO
56    =========================================================================*/
57    union BtoF // vrpm por serial
58    {
59      byte b[16];
60      float fval;
61    } up;
62    union BtoF2 //para recibir ref puerto serial
63      {
64      byte b2[16];
65      float fval2;
66      } u2;
67
68    union BtoF3 //Envió de %PWM (pwmOut) por serial
69    {
70      byte b3[16];
71      float fval3;
72    } u3;
73    union BtoF4 // Para Envió de kp
74    {
75      byte b4[16];
76      float fval4;
77    } u4;
78
79    union BtoF5 // Para Envio ti
80    {
81      byte b5[16];
82      float fval5;
83    } u5;
84
85    union BtoF6 // Para Envio td
86    {
87      byte b6[16];
88      float fval6;
89    } u6;
90
91    //struct FloatValues;
92    struct FloatValues {
93      float value1;     // w
94      float value2;    // kp
95      float value3;   // ti
96      float value4;  // td
97      };
98    FloatValues instancia; // instancia o objeto tipo estruct
99    volatile unsigned long pulseCount = 0;
100   unsigned long lastTime = 0;
101   unsigned long deltaTime = 0;
102
103   // Interrupción externa sensor óptico
104   void ICACHE_RAM_ATTR encoderInterrupt() {
105     pulseCount++; // Incremento de contador en cada pulso
106   }
```

```
/*============================================================================
||||||||||||||FUNCIÓN SETUP
============================================================================*/
void setup() {
  Serial.begin(115200);        // Configuramos el puerto serial baudios
  //Declaración de modo de pines
  pinMode(LED_PIN,OUTPUT);            // Led Alarma
  pinMode(ENCODER_PIN,INPUT);        //  Encoder como entrada
  pinMode(MOTOR_PIN1,OUTPUT);       //  Dir motdrive1 salida
  pinMode(MOTOR_PIN2,OUTPUT);      //  Dir motdrive2 salida
  pinMode(SETPOINT_PIN,INPUT);    // Pin entrada Pot setpoint fisico
  pinMode(MODE_PIN,INPUT); // Selector de modo de operación
  digitalWrite(LED_PIN,LOW); // Inicializa apagado alarma
  // Config. de la interrupción
  // Inicializar la pantalla OLED
  if(!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
    Serial.println(F("No se pudo inicializar la pantalla OLED"));
    while(true);
  }
  display.display();
  delay(2000);
  display.clearDisplay();

  lastTime = millis();        // Recupera el tiempo de inicio actual en ms
  digitalWrite(LED_PIN,LOW); // Inicializa apagado alarma
  // Config. de la interrupción con Ticker
  interrupcion1.attach(1, pid_eval);// Llama a la interrupción cada 1 segundos
  // Configuración PWM
  ledcSetup(0, 5000, 8);
  ledcAttachPin(ENABLE1_PIN, 0); //Declara Pin 32 como salida PWM
  //ledcWrite(0,0);         // Inicializa PWM de salida en 0%
  // Config. interrupción externa a partir de encoder óptico
  //Configura la interrupción del Timer 1 flanco de subida
  attachInterrupt(digitalPinToInterrupt(ENCODER_PIN), encoderInterrupt, RISING) ;
/*============================================================================
|||||||||||| INICIALIZACIÓN DE CONSTANTES DEL PID
Nota:
  | En esta sección puede inicializar las constantes Kp, Ti , Td y luego modificarlas
  | en la la interfaz gráfica.
============================================================================*/
//          CONSTANTES DE SINTONIA DEL PI

  kp=0.01;      // Constante proporcional del PID
  ti=1.3470;   // Constante integral del PID
  td=0;       // Constante derivativa del PID

  // CONSTANTES DEL CONTROLADOR PID DISCRETO
```

```
154
155     c0=kp*(1+Ts/(2*ti)+td/Ts);
156     c1=-kp*(1-Ts/(2*ti)+(2*td)/Ts);
157     c2=(kp*td)/Ts;
158
159   }
160
161   /*=========================================================================
162     |||||||||||||||BUCLE PRINCIPAL
163     =========================================================================*/
164   void loop() {
165
166     //Alarmas cuando supere las 1200 rpm
167     if(revpm>1200)
168     | digitalWrite(LED_PIN, HIGH); // Alarma de transistor encendida
169     else
170     | digitalWrite(LED_PIN, LOW); // Alarma de transistor apagada
171
172   /*=========================================================
173     |||||||| Actualización de contador
174     =========================================================*/
175     unsigned long currentTime = millis();  // Recupera el tiempo actual
176     deltaTime = currentTime - lastTime;   // Intervalo transcurrido desde que inicia programa
177     //Serial.print(Pot);
178     //  Se ejecuta una acción cada 1s:
179     if (deltaTime >= 1000) {             // Calcula la velocidad cada segundo
180       revpm = (pulseCount * 60.0);      // Convierte a Rev/s a Rev/m
181       revpm = filtro_pmovil(revpm);
182       pulseCount = 0;          // Reinicia contador para siguiente conteo de rpm
183       lastTime = currentTime; // Actualiza el tiempo
184
185       /*=================================================
186       | | COMUNICACIÓN SERIAL INTERFAZ-ESP32
187       =================================================*/
188       if (digitalRead(MODE_PIN)==HIGH){//modo remoto
189       | | int Val_pot = analogRead(SETPOINT_PIN);  // Lectura del potenciómetro de referencia
190       | | Pot = map(Val_pot,0,4095,0,2800);        // Mapear velocidad desde 0 RPM hasta 2800 RPM máx
191       | | ref = Pot; // Asigna valor de potenciómetro físico
192       | | display_in_OLED (revpm, v_out, ref);
193       | | delay(500); // Actualizar cada 1/2 segundo
194       | | /* comentar para usar modo interfaz
195       | | Serial.print(ref);
196       | | Serial.print(" ");
197       | | Serial.print(revpm);
198       | | Serial.print(" ");
199       | | Serial.println(pwmOut);
200       | | //*/
201       | |  }else{
```

```
202         // Recibir Datos por Puerto Serial
203         if (Serial.available()>0)
204         {
205           readFromMatlab();
206           ref  = instancia.value1;          // Actualiza ref
207           kp = instancia.value2;           // Actualiza kp
208           ti = instancia.value3;          //  Actualiza ti
209           td = instancia.value4;         // Actualiza td
210         }
211         // Enviar Datos por Puerto Serial
212         writeToMatlab(revpm, v_out, kp, ti, td);
213
214         // Actualizar valores obtenidos
215         c0=kp*(1+Ts/(2*ti)+td/Ts);
216         c1=-kp*(1-Ts/(2*ti)+(2*td)/Ts);
217         c2=(kp*td)/Ts;
218       }
219     }
220   }
221   /* ====================================================================
222     |||||| INTERRUPCION CADA Ts SEGUNDOS
223     ====================================================================*/
224   void pid_eval()
225   {
226     actualiza_vectores(u,kU);   //Actualiza vector u
227     actualiza_vectores(e,kE);  //Actualiza vector e
228
229     if(ref != 0){ // Mientras la referencia sea 0 no ejecuta accion de control
230     e[kE] = ref - revpm;           //Calcula el error actual: error=ref-realimentación
231     // Calcula la Acción de Control PID
232     u[kU] = PID_Discreto(u, e, c0, c1, c2);  //Max= 100, Min=0
233     pwmOut = u[kU]; // 0 a 100
234     v_out = pwmOut/30.30;// ley de control a escribir en Voltaje
235     mpwmOut = map(pwmOut , 0,100, 0,255);    //Mapeo de ley de control a escribir PWM
236     //Aplica la acción de control en el PWM
237     //Max= 100% PWM, Min=0% PWM
238     ledcWrite(0,mpwmOut); // escribe la salida del control en canal 0
239     digitalWrite(MOTOR_PIN1, HIGH);
240     digitalWrite(MOTOR_PIN2, LOW);
241
242   }
243   else
244     {
245     e[kE] = ref - revpm;           //Calcula el error actual: error=ref-realimentación
246     // Calcula la Acción de Control PID
247     u[kU] = 0; // Forzamos a 0 la ley de control
248     pwmOut = u[kU]; // 0 a 100
249     v_out = pwmOut/30.30;// ley de control a escribir en Voltaje
250     mpwmOut = map(pwmOut , 0,100, 0,255);    //Mapeo de ley de control a escribir PWM
251     ledcWrite(0,mpwmOut); // escribe la salida del control
252     digitalWrite(MOTOR_PIN1, HIGH);   // Desactiva giro de motor
253     digitalWrite(MOTOR_PIN2, LOW);
254     }
255   }
```

```
/*=====================================================================
 | | | | | FUNCIONES AUXILIARES PARA COM. SERIAL
 =====================================================================*/
// Leer desde Matlab por com. serial
void readFromMatlab()
{
  String rein = Serial.readString(); // Lee la trama desde la interfaz
  for (int i =0; i<4; i++)
  {

    u2.b2[i] = rein[i];         // referencia
    u4.b4[i] = rein[i+4];      // kp
    u5.b5[i] = rein[i+8];     // ti
    u6.b6[i] = rein[i+12];   // td
  }

  instancia.value1 = u2.fval2;      // guarda ref
  instancia.value2 = u4.fval4;     // guarda kp
  instancia.value3 = u5.fval5;    // guarda ti
  instancia.value4 = u6.fval6;   // guarda td
}
// Escribir por serial hacia Matlab
//                   revpm         pwmOut          kp              ti            td
void writeToMatlab(float number1, float number2,float number3,float number4,float number5)
{
  byte *b = (byte *) &number1;         // Apunta espacio de memoria de byte de revpm
  byte *b3 = (byte *) &number2;       // Apunta espacio de memoria de byte de pwmOut
  byte *b4 = (byte *) &number3;      // Apunta espacio de memoria de byte de kp
  byte *b5 = (byte *) &number4;     // Apunta espacio de memoria de byte de ti
  byte *b6 = (byte *) &number5;    // Apunta espacio de memoria de byte de td  */
  Serial.write("A");             // caracter inicial letra A
  Serial.write(b,4);            // Trama de velocidad (verpm)
  Serial.write(b3,4);          // Trama de ley de control (pwmOut)
  Serial.write(b4,4);         // kp  b2
  Serial.write(b5,4);        // ti
  Serial.write(b6,4);       // td      b6 */
  //Serial.write("B");    // caracter final letra B
  Serial.write(13);      // Envia caracter '\r'
  Serial.write(10);     // Envia caracter final '\n'
}

```

```
297  void display_in_OLED (float velocidad, float voltaje, float referencia)
298  {
299    /*
300      PANTALLA OLED
301  */
302    display.clearDisplay();
303    // Mostrar la temperatura en la pantalla OLED
304      display.setTextSize(1.5);
305      display.setTextColor(SSD1306_WHITE);
306      display.setCursor(0,0);
307      display.print("Velocidad:");
308      display.setTextSize(1.5);
309      display.setCursor(75, 0);
310      display.print(velocidad, 2); // (variable, #decimales)
311      display.setTextSize(1.5);
312      display.setCursor(100, 0); //(x,y)
313      display.print(" [RPM]");
314      // Voltaje PWMOUT
315      display.setTextSize(1.5);
316      display.setCursor(0, 20);
317      display.print("Voltaje: ");
318      display.setTextSize(1.5);
319      display.setCursor(75, 20);
320      display.print(voltaje,2);
321      display.setTextSize(1.5);
322      display.setCursor(100, 20); //(x,y)
323      display.print(" [V]");
324
325
326      // REFERENCIA
327      display.setTextSize(1.5);
328      display.setCursor(0, 40);
329      display.print("SetPoint: ");
330      display.setTextSize(1.5);
331      display.setCursor(75, 40);
332      display.print(referencia,2);
333      display.setTextSize(1.5);
334      display.setCursor(100, 40); //(x,y)
335      display.print(" [RPM]");
336
337      display.display();
338  }
```

APÉNDICE I.

Figura I.1. Código en IDE de Arduino para control PID de velocidad Fuzzy-Mamdani en ESP32.

fuzzy_esp32PV_49mod.ino | fuzzy.cpp | fuzzy.h | tsclab_lib.cpp | tsclab_lib.h

```
/*  ============================================================
    CONTROLADOR FUZZY - MAMDANI PARA CONTROL DE VELOCIDAD
          By Jesús Martínez & Javier Nicolalde
              Implementación ESP32
                 Planta TSCLAB
          última revisión: 29/08/2023
    ============================================================*/
//                    LIBRERIAS
#include <Ticker.h>           // Librería de interrupciones
#include "tsclab_lib.h"      // Librería para funciones tsclab
#include <Wire.h>
#include "fuzzy.h"          // Librería para funciones control Fuzzy-Mamdani
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
Ticker interrupcion1;
// Definiciones componentes de la tarjeta
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET    13
#define SCREEN_ADDRESS 0x3C
// Definiciones componentes de la tarjeta
#define ENABLE_MOT 32     //pin3 arduino  Habilitador de DRIVER L293D PWM
#define motor_pin1 33    //pin10 arduino     // pin33 esp32
#define motor_pin2 25   // pin4 arduino pin25 esp32
#define encoder 27     // pin2 arduino pin27 esp32
#define ALARM1 18     // pin 6 arduino pin18 esp32 LED Alarma PWM
#define SETPOINT_PIN 2  // Potenciómetro para referencia física
#define MODE_PIN 23 // Selector de modo de operación
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

/*============================================================
  VARIABLES GLOBALES PARA ERROR Y LEY DE CONTROL
  ============================================================*/
float v_out =0.0;       // Ley de control en voltaje (0 a 3.3v)
float pwmOut =0.0;        // %PWM V_Base TIP31
float vrpm=0;            // velocidad en rpm
float w=0;              // Referencia (400 rpm a 2500 rpm)
float Pot = 0.0;                  // Potenciómetro para referencia física
// selector: false modo interfaz, true modo remoto
bool selectorState = false;  // Switch para seleccionar referencia física o por interfaz

float e[2]={0,0};      // Vector de error
float u[2]={0,0};     // Vector de Ley de Control
int kU = sizeof(u)/sizeof(float)-1;    // kU es la pos. de la ley de control actual
int kE = sizeof(e)/sizeof(float)-1;   // e[kE] es el error actual

```

```
/*============================================================
  DEFINICIONES GLOBALES CONTROL FUZZY-MAMDANI
  UNIVERSO DEL DISCURSO
  ============================================================*/
float discurso_e = 2500, discurso_de = 1000, discurso_u = 10;
float der;       // Almacenar derivada del  discreta
float lu;       // Almacenar ley de control
int Ts = 1;    // Periodo de Muestreo para calculo de derivada discreta

const int buffer_size =16;    // Tamaño del buffer
byte buf[buffer_size];     // Crea un byte buffer de 8 bytes

// Estructura para trabajar con datos tipo byte y float a la vez
union BtoF // Para variable rpm sensados
{
  byte brpm[16];
  float fvalrpm;
} rpmBorF;
union BtoF2 // Para variable pwmOut
{
  byte bpwmOut[16];
  float fvalpwm;
} pwmBorF;
union BtoF3 // Para variable de referencia (w)
{
  byte bw[16];
  float fvalw;
} wBorF;
// Estructura para guardar parámetros
struct FloatValues {
  float v_w;       // w (ref)
  float v_rpm;    // vrpm
  float v_pwm;   // pwm
  };
FloatValues instancia; // instancia o objeto tipo estruct

```

```
/*=============================================================
| | | INTERRUPCION CADA 1 SEGUNDO ACTUALIZACIÓN
=============================================================*/
void interr_1_control()
{
  actualiza_vectores(u,kU);   //Actualiza vector u
  actualiza_vectores(e,kE);  //Actualiza vector e

  e[kE] = w - vrpm;      //Calcula el error actual: error=ref-realimentación

  // Satura el error para estar dentro del universo del discurso
  if(e[kE]<-1*discurso_e+0.1)    // limite inf discurso_e
    e[kE]=-1*discurso_e+0.1;    // Satura discurso_e
  if(e[kE]>discurso_e-0.1)     // limite sup discurso_e
    e[kE]=discurso_e-0.1;     // Satura discurso_e

  // Calcula la derivada del error discreta
  der=(e[kE]-e[kE-1])/Ts; //fórmula de aprox. de una derivada discreta

  // Satura la derivada del error para estar dentro del universo del discurso
  if(der<-1*discurso_de+0.01)
    der=-1*discurso_de+0.01;
  if(der>discurso_de-0.01)
    der=discurso_de-0.01;

  // Ley de control fuzzy
  // Parámetros: (error_actual,derivada_error,Univ_disc_e,Univ_disc_de, Univ_disc_leycontrol)
  lu=myfuzzy(e[kE],der,discurso_e,discurso_de,discurso_u);

  //Integral Colocada en salida de u
  u[kU] = lu + u[kU-1];   // Accion de control tiempo presente
  // Anti - Windup (Saturación de ley de control)
  if (u[kU] >= 100.0)
    u[kU] = 100.0;
  if (u[kU] <= 4.0)
    u[kU] = 0.0;
  pwmOut = u[kU];      // Actualiza el %PWM correspondiente
  v_out = pwmOut/30.30; // 0 a 3.3V
  //Aplica la acción de control PWM en pinEnable Driver L293D
  ledcWrite(0,map(pwmOut , 0,100, 0,255)); //Max= 100, Min=0
  // Solo gira en un semtido:
  digitalWrite(motor_pin1, HIGH);   // Activa PinDriver 1
  digitalWrite(motor_pin2, LOW);   //  Activa PinDriver 2
}
volatile unsigned long pulseCount = 0;
unsigned long lastTime = 0;
unsigned long deltaTime = 0;
unsigned long lastTime2 = 0;
```

```
unsigned long deltaTime2 = 0;
// Interrupción externa sensor óptico
void ICACHE_RAM_ATTR encoderInterrupt() {
  pulseCount++; // Incremento de contador en cada pulso
}

/*=====================================================================
            FUNCIÓN SETUP
=====================================================================*/
void setup() {
  Serial.begin(115200);            // Configuramos el puerto serial baudios
  pinMode(ALARM1,OUTPUT);          // Alarma como salida
  pinMode(motor_pin1, OUTPUT);    // motorpin1 drive L293D como salida
  pinMode(motor_pin2, OUTPUT);   // motorpin2 drive L293D como salida
  pinMode(ENABLE_MOT, OUTPUT); // Habilitador de drive L293D como salida
  pinMode(encoder, INPUT);     // Encoder óptico como entrada
  pinMode(MODE_PIN,INPUT); // Selector de modo de operación
  if(!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
    Serial.println(F("No se pudo inicializar la pantalla OLED"));
    while(true);
  }
  display.display();
  delay(2000);
  display.clearDisplay();

  lastTime = millis();       // Recupera el tiempo actual en ms
  // Config. de la interrupción
  interrupcion1.attach(Ts, interr_l_control);// Llama a la interrupción cada Ts segundos
  // Configuración PWM
  ledcSetup(0, 5000, 8);         //  0, frec=5000Hz, resolución 8 bits
  ledcAttachPin(ENABLE_MOT, 0); //pin enablemot -  canal 0
  // Config. interrupción externa a partir de encoder óptico
  //Configura la interrupción del Timer 1 flanco de subida
  attachInterrupt(digitalPinToInterrupt(encoder), encoderInterrupt, RISING) ;

}

/*=====================================================================
            FUNCIÓN PRINCIPAL
=====================================================================*/
void loop() {

  //Indicación Luminica o alarma
  if(vrpm>1200)
    digitalWrite(ALARM1, HIGH); // Indicador cuando llega a una velocidad de 1200 rpm
  else
    digitalWrite(ALARM1, LOW); // Indicador apagado
  /*-------------------------------------------------------
      Actualización de contador
  -------------------------------------------------------*/
  unsigned long currentTime = millis();  // Recupera el tiempo actual
  deltaTime = currentTime - lastTime;   // Intervalo transcurrido desde que inicia programa
  //  Se ejecuta una acción cada 1s:
  if (deltaTime >= 1000) {             // Calcula la velocidad cada segundo
    vrpm = (pulseCount * 60.0);       // Convierte a RPS a RPM

    pulseCount = 0;          // Reinicia contador para siguiente conteo de rpm
    lastTime = currentTime; // Actualiza el tiempo
  /*-------------------------------------------------------
      COMUNICACIÓN SERIAL MATLAB-ARDUINO
  -------------------------------------------------------*/
    if (digitalRead(MODE_PIN)==HIGH){//modo remoto
        int Val_pot = analogRead(SETPOINT_PIN);  // Lectura del potenciómetro de referencia
        Pot = map(Val_pot,0,4095,0,3500);        // Mapear velocidad desde 0 RPM hasta 2800
        w = Pot; // Asigna valor de potenciómetro físico
        display_in_OLED (vrpm, v_out, w);
        delay(500); // Actualizar cada 1/2 segundo
```

```
198         /* comentar para usar modo interfaz
199         Serial.print(w);      // referencia física pot
200         Serial.print(" ");
201         Serial.print(vrpm);    // velocidad encoder
202         Serial.print(" ");
203         Serial.println(v_out);    //ley de control en voltaje
204         //*/
205       }else{
206         //Recibir datos por puerto serial
207         if (Serial.available()>0)
208           {
209           readFromMatlab();     // Leer desde Matlab por serial
210           w = instancia.v_w;   // Actualiza referencia (w)
211           }
212         //  Enviar Datos por Puerto Serial
213           writeToMatlab(vrpm, v_out); // Escribir por serial vrpm y pwmOut
214       }
215     }
216   }
217   /* ================================================================
218     FUNCIONES AUXILIARES PARA COM. SERIAL
219     ================================================================*/
220   // Leer desde Matlab por com. serial
221   void readFromMatlab()
222   {
223     String rein = Serial.readString(); // Recibe trama de Matlab
224     for (int i =0; i<4; i++)          // Recorre 4 bits
225     {
226     wBorF.bw[i] = rein[i];         // Guarda referencia w
227     }
228     instancia.v_w = wBorF.fvalw;
229     instancia.v_rpm = vrpm;
230     instancia.v_pwm = pwmOut;
231     }
232   // Escribir por serial hacia Interfaz gráfica
233   void writeToMatlab(float rpms, float pwmout)
234   {
235     byte *brpm = (byte *) &rpms;       // Guarda en brpm las rpm
236     byte *bpwmOut = (byte *) &pwmout; // Guarda en bpwmOut el pwmOut
237     Serial.write("A");               // Caracter inicial letra A (ASCII=65)
238     Serial.write(brpm,4);           // Trama de velocidad (vrpm)
239     Serial.write(bpwmOut,4);       // Trama de ley de control (%PWM)
240     Serial.write(13);      // Envia caracter '\r'
241     Serial.write(10);     // Envia caracter final '\n'
242   }
243
```

```
void display_in_OLED (float velocidad, float voltaje, float referencia)
{
  /*
    PANTALLA OLED
*/
    display.clearDisplay();
    // Mostrar la temperatura en la pantalla OLED
     display.setTextSize(1.5);
     display.setTextColor(SSD1306_WHITE);
     display.setCursor(0,0);
     display.print("Velocidad:");
     display.setTextSize(1.5);
     display.setCursor(55, 0);
     display.print(velocidad, 2); // (variable, #decimales)
     display.setTextSize(1.5);
     display.setCursor(100, 0); //(x,y)
     display.print(" [RPM]");
     // Voltaje PWMOUT
     display.setTextSize(1.5);
     display.setCursor(0, 20);
     display.print("Voltaje: ");
     display.setTextSize(1.5);
     display.setCursor(55, 20);
     display.print(voltaje,2);
     display.setTextSize(1.5);
     display.setCursor(100, 20); //(x,y)
     display.print(" [V]");


     // REFERENCIA
     display.setTextSize(1.5);
     display.setCursor(0, 40);
     display.print("SetPoint: ");
     display.setTextSize(1.5);
     display.setCursor(55, 40);
     display.print(referencia,2);
     display.setTextSize(1.5);
     display.setCursor(100, 40); //(x,y)
     display.print(" [RPM]");

     display.display();
}
```

APÉNDICE J. Guía de usuario del equipo control TSC.

GUIA DE USUARIO

MODOS DE OPERACIÓN.

La presente planta cuenta con dos modos de control y monitoreo.

a) **Control Modo Interfaz Mediante App Control TSC / AppDesigner / Matlab.**

En este caso se tendrá la posibilidad de realizar el control y monitoreo mediante la aplicación App Control TSC o Matlab, la cual por medio de conexión serial hacia la tarjeta de control mostrará en tiempo real dos graficas como lo son el Setpoint deseado con la señal controlada, y la señal de control. Además, la aplicación permitirá exportar los datos obtenidos para posteriormente obtener los indicies de desempeño del controlador.

***Nota 1:**

Exportar datos solo está disponible usando AppDesigner/Matlab.

b) **Control en modo Remoto.**

El modo remoto permite realizar el control mediante la perilla de referencia y monitoreo a través de la pantalla OLED 0.96 integrada en el sistema.

***Nota 2:**

Dependiendo de la tarjeta de desarrollo usada se podrá visualizar los datos de interés, si el procesamiento del controlador es excesivo podría generarse conflicto con las librerías de la pantalla OLED. Para ejemplo véase la sección de controladores en Arduino.

REQUISITOS PARA USO DE LOS DISPOSITIVOS.

Para uso del equipo en **modo interfaz** se necesitan los siguientes Software:

- ❖ App Control TSC /AppDesigner / Matlab 2022b o versiones posteriores.
- ❖ Arduino IDE u otro software que permita cargar los controladores.

***Nota 3:**

Usando el IDE de Arduino se necesitan instalar las siguientes librerías:

- ✓ **Ticker** by Stefan Staub. En el enlace siguiente puede acceder a más información: https://github.com/sstaub/Ticker

- ✓ **TimerOne** by Stoyko Dimtrov, entre otros. En el enlace siguiente puede acceder a más información: https://playground.arduino.cc/Code/Timer1/
- ✓ **OneWire** by Jim Studt, entre otros. En el enlace siguiente puede acceder a más información: https://www.pjrc.com/teensy/td_libs_OneWire.html
- ✓ **DallasTemperature** by Miles Burton, entre otros. Véase para más información el enlace: https://github.com/milesburton/Arduino-Temperature-Control-Library
- ✓ **BasicLinearAlgebra** by Tom Stewart. Véase el siguiente enlace para más información: https://github.com/tomstewart89/BasicLinearAlgebra
- ✓ **Adafruit SSD1306** by Adafruit. Véase el siguiente enlace para más información: https://github.com/adafruit/Adafruit_SSD1306
- ✓ **Adafruit GFX Library** by Adafruit. Véase el siguiente enlace para más información: https://github.com/adafruit/Adafruit-GFX-Library

***Nota 4:**

Si una librería depende de otras, se recomienda instalar todas sus dependencias al momento de instalar.

***Nota 5:**

Si se usa la **App Control TSC** no se necesita tener instalado Matlab 2022b, solo se requiere de un espacio de memoria de 2.32GB para instalar la App en un computador.

DESCRIPCIÓN DEL HARDWARE DE LOS EQUIPOS.

- **TSC-LAB con ESP32.**

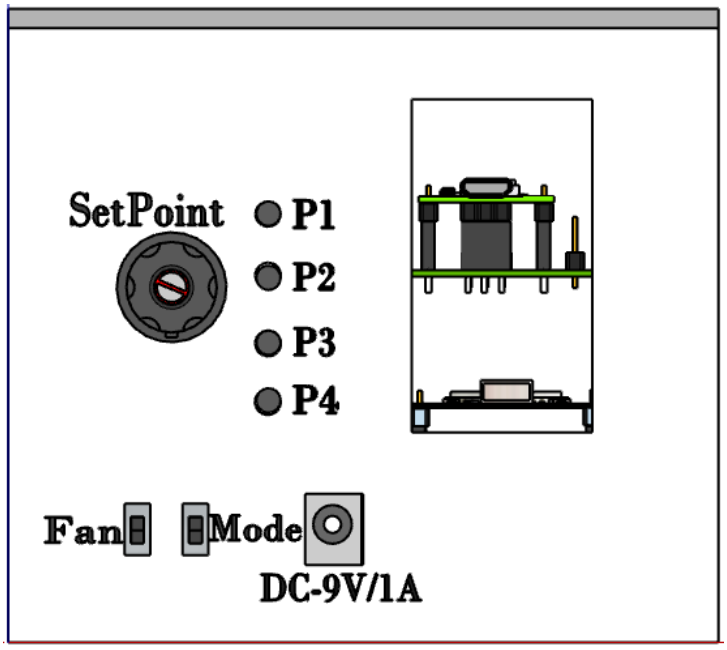


Ilustración 2. Vista frontal del equipo TSC-LAB con Esp32.

- ✓ **Puerto de Alimentación:** 9Vdc / 1 a 2 A MAX.
- ✓ **Controlador:** ESP-WROOM-32
- ✓ **Pulsadores de propósito general:** P1, P2, P3, P4
- ✓ **Modo de operación Remoto/Interfaz:** Mode Switch
- ✓ **Ventilación de planta:** Fan Switch
- ✓ **Control Externo**: Perilla Setpoint

  La planta cuenta además con una tarjeta de control **Raspberry Pi Pico** adicional mediante el cual se podrá hacer uso de los pulsadores de propósito general **P1, P2, P3 y P4**.

- **TSC-LAB con Arduino Uno.**

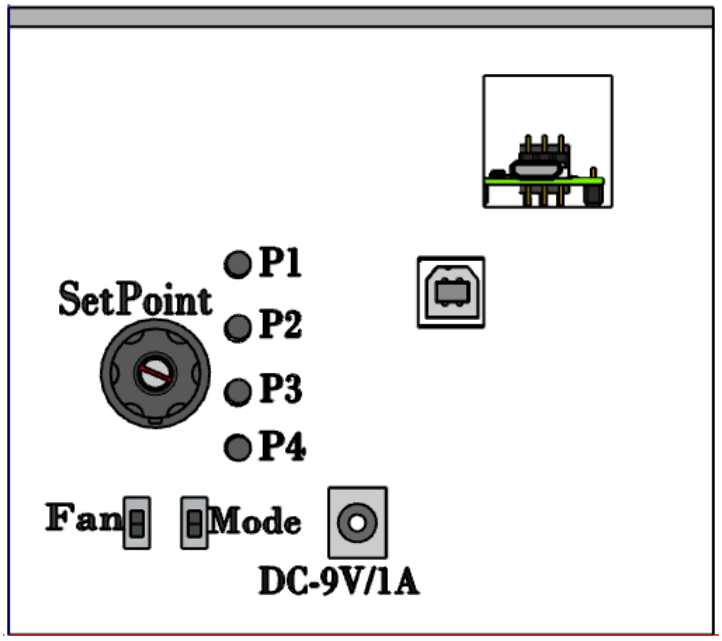


Ilustración 3. Vista frontal del equipo TSC-LAB con Arduino Uno.

- ✓ **Puerto de Alimentación:** 9Vdc / 1 a 2 A MAX.
- ✓ **Controlador:** Arduino Uno R3
- ✓ **Pulsadores de propósito general:** P1, P2, P3, P4
- ✓ **Modo de operación Remoto/Interfaz:** Modo Switch
- ✓ **Ventilación de planta:** Fan Switch
- ✓ **Control Externo**: Perilla Setpoint

La planta cuenta además con una tarjeta de control **Raspberry Pi Pico** adicional mediante el cual se podrá hacer uso de los pulsadores de propósito general **P1, P2, P3 y P4**.

***Nota 6:**

Se deja al usuario libre opción de uso de los pulsadores, debe asegurarse de activar las resistencias PULL_DOWN de la Raspberry Pi Pico en la programación.

- **Alimentación eléctrica / comunicación.**

  Para el funcionamiento la planta requiere dos conexiones, primero la alimentación que corresponde a una fuente de alimentación de 9Vdc/1 A y el conector de comunicación serial micro USB en el caso ESP32 o un conector USB tipo B si se trata de la planta basada en Arduino Uno.

  - ✓ **Regulador o fuente de alimentación 9Vdc/1 A.**

  ✓ 

  - ✓ **Conector serial Arduino Uno:**

  ✓ 

  - ✓ **Conector serial ESP32:**

  ✓ 

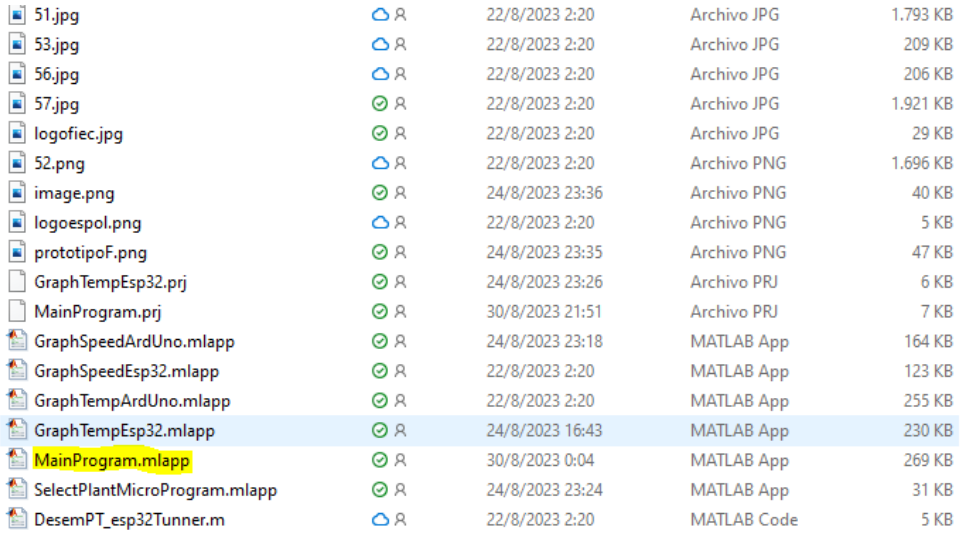
Una vez realizadas las conexiones de alimentación y comunicación serial, se procede a seleccionar y cargar el código de acuerdo con controlador deseado.

***Nota:**

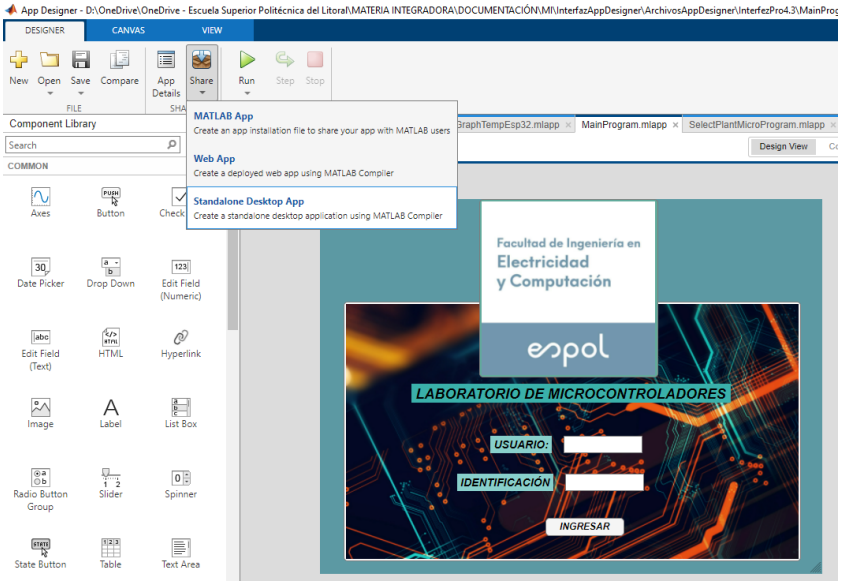
Se puede alimentar el equipo solo con la USB de comunicación. Sin embargo, se recomienda usar también la fuente de alimentación de 9Vdc 1 A.

INSTALACIÓN DE APP CONTROLTSC EN PC.

- ✓ Primero, abra Matlab 2022b. Luego, ejecute en el "Command Windows" **"AppDesigner".** Abra el archivo **MainProgram.mlapp** de la carpeta de archivos:

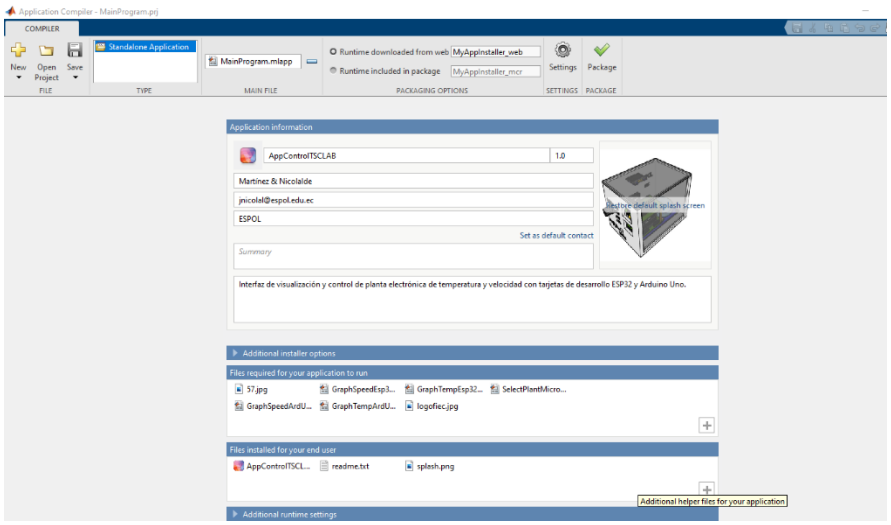


- ✓ Una vez abierto, de clic en "**share**" y luego, "**Standalone Desktop App**":

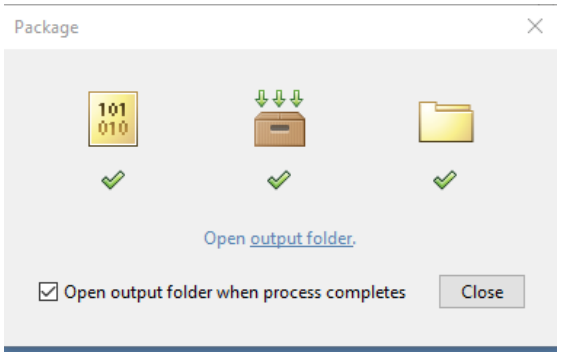


- ✓ Se abre una ventana de configuraciones, en esta se puede modificar el nombre de la aplicación, colocar una imagen de icono inicial, agregar descripción de la aplicación,

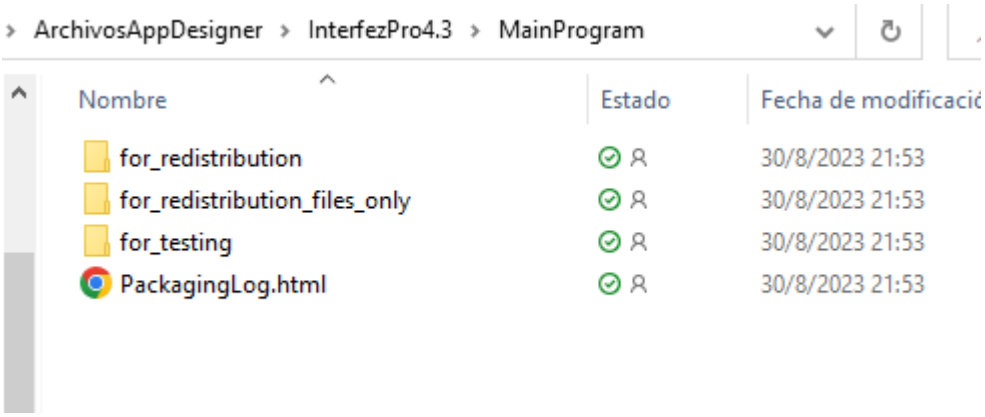
colocar autor, organización, etc. Una vez configurado debe asegurarse de tener seleccionado la opción, "**Runtime dowloaded from web**":



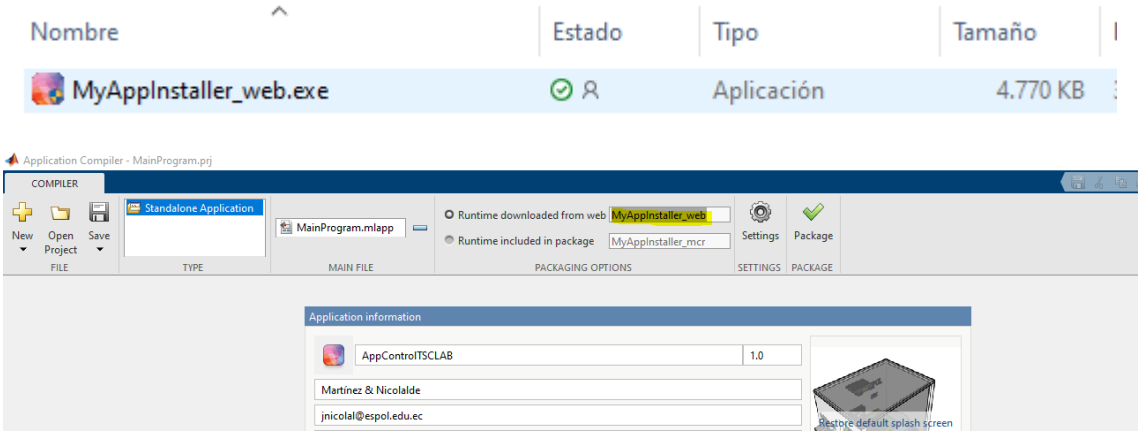
✓ Luego, una vez terminado de configurar, dar clic en "**Package**", se iniciará la generación de los archivos necesarios, debe dar clic en permitir ejecutar del administrador:



✓ Una vez finalizado, se genera una carpeta llamada "**MainProgram**", dentro se hallan 3 subcarpetas:



✓ La carpeta que permite la instalación local en un computador es "**for_redistribution",** abra la carpeta mencionada y ejecute el archivo "**MyAppInstaller_web.exe**" que es el mismo nombre que se colocó en las configuraciones:

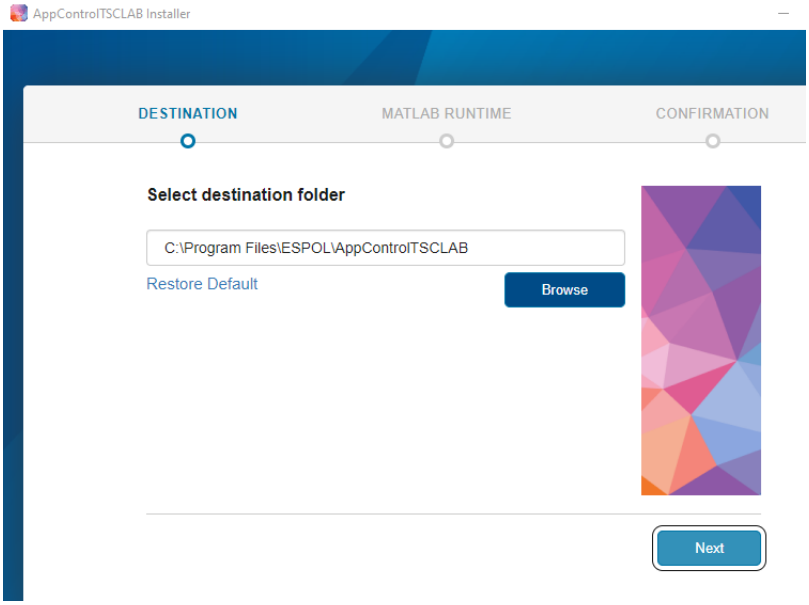


***Nota:**

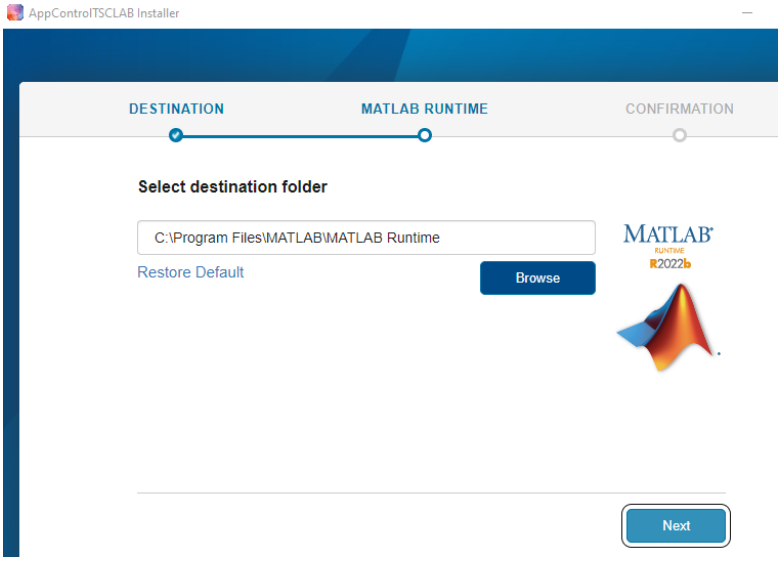
Si aparece un cuadro de dialogo, debe dar clic en permitir la instalación del administrador del PC.

Luego, aparecerá el asistente de instalación, dar clic en "**Next**":

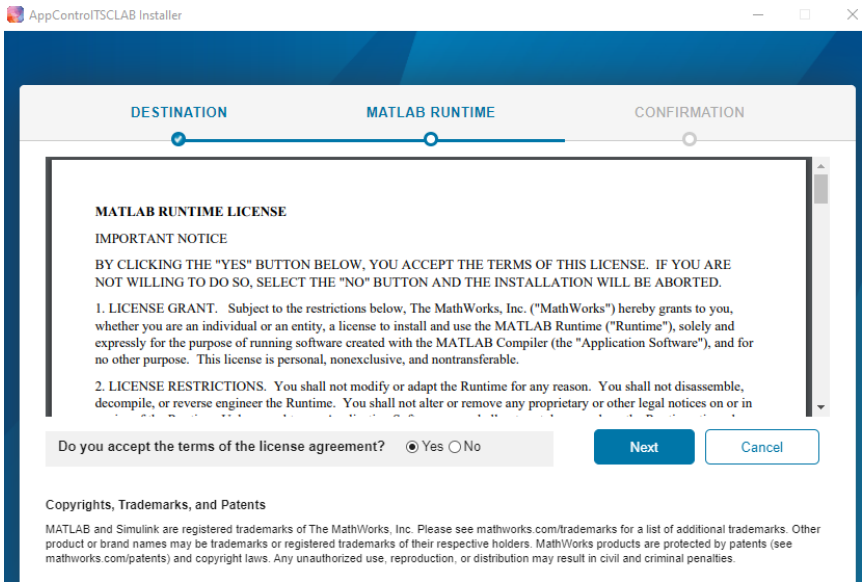
- Seleccione la ruta de instalación de la App en su PC y de clic en "**Next**":



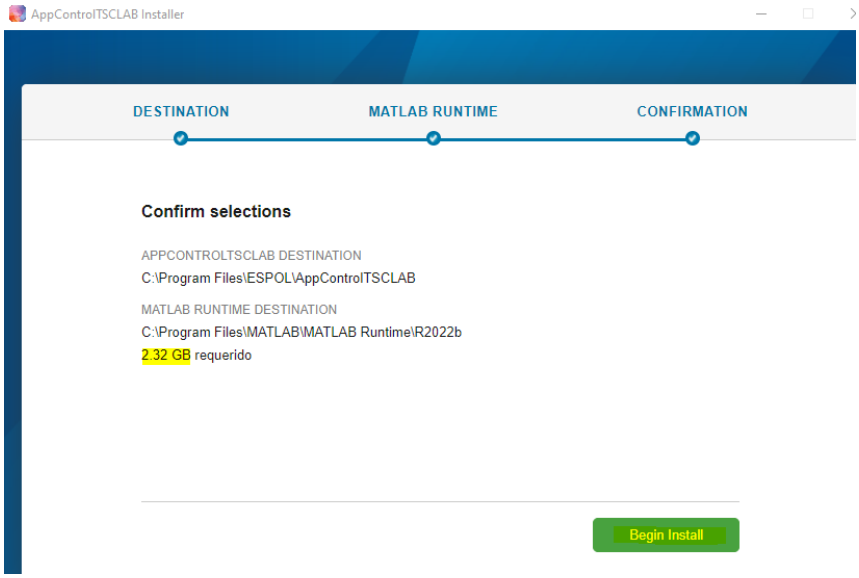
- Seleccione la ruta de instalación del "Matlab Runtime" en su PC y de clic en "**Next**":



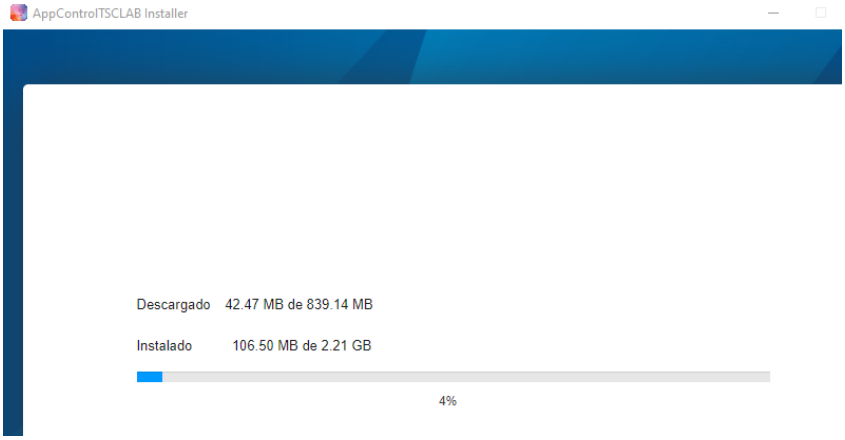
- Acepte los términos y condiciones:

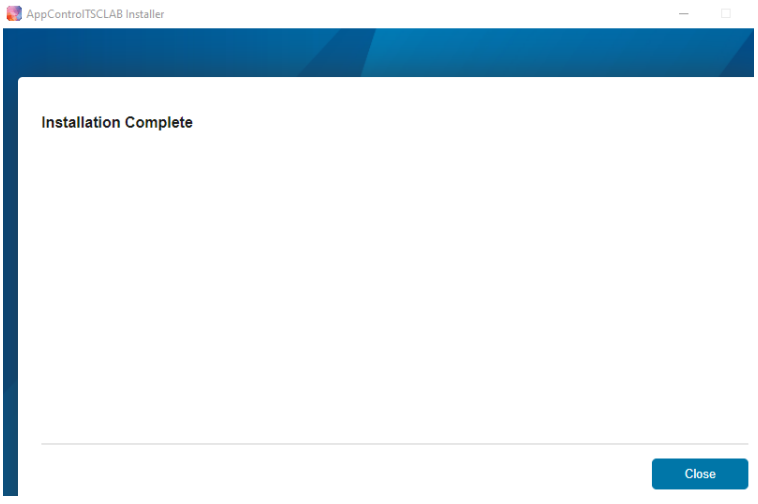


- Luego, se mostrará la información de los requisitos y configuraciones realizadas, de estar correctas de clic en "**Begin Install**":

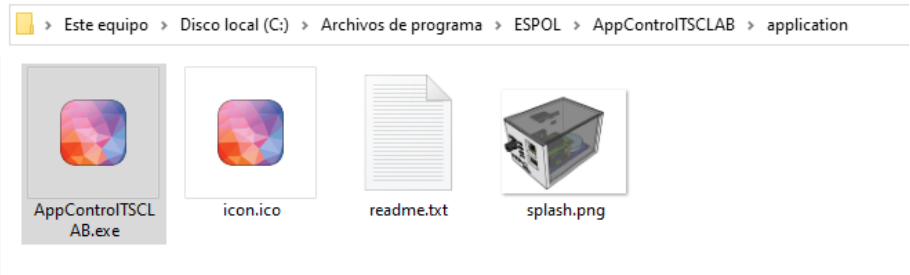


- Se iniciará la instalación, y una vez finalizada de clic en "**Close**":





- Nos vamos a la ruta que seleccionamos para la App y creamos un icono de acceso directo al escritorio y ejecutamos:



USO Y DESCRIPCIÓN DE CONTROL TSC APPDESIGNER/MATLAB.

En esta sección se muestra de manera general una descripción del funcionamiento de AppDesigner. Se hace un enfoque a la sección de comunicación serial, con el objetivo que el usuario pueda modificar según lo requiera en su aplicación o análisis.

Análisis de función para enviar datos hacia Matlab (writeToMatlab).

Analizando a modo de ejemplo el control PID por sintonización prueba-error, usando la tarjeta ESP32. La función **writeToMatlab** está diseñada para enviar una serie de datos en formato float (números de punto flotante) a través del puerto serial desde un Arduino u otro dispositivo compatible hacia una aplicación de MATLAB o algún otro software que esté esperando recibir estos datos para su análisis o procesamiento.

```
void writeToMatlab(float number1, float number2, float number3, float number4, float number5, float number6)
{
  byte *b = (byte *) &number1;   // Crea un puntero byte que apunta a la dirección de memoria de number1 (temperatura)
  byte *b3 = (byte *) &number2;  // Crea un puntero byte que apunta a la dirección de memory de number2 (H1)
  byte *b4 = (byte *) &number3;  // Crea un puntero byte que apunta a la dirección de memory de number3 (kp)
  byte *b5 = (byte *) &number4;  // Crea un puntero byte que apunta a la dirección de memory de number4 (ti)
  byte *b6 = (byte *) &number5;  // Crea un puntero byte que apunta a la dirección de memory de number5 (td)

  // Envía cada valor precedido por un caracter de identificación y seguido por caracteres de terminación.
  Serial.write("A");     // Envía un caracter "A" como indicador de inicio de trama
  Serial.write(b, 4);    // Envía los 4 bytes de number1 (temperatura)
  Serial.write(b3, 4);   // Envía los 4 bytes de number2 (H1)
  Serial.write(b4, 4);   // Envía los 4 bytes de number3 (kp)
  Serial.write(b5, 4);   // Envía los 4 bytes de number4 (ti)
  Serial.write(b6, 4);   // Envía los 4 bytes de number5 (td)
  Serial.write(13);      // Envía un caracter '\r' (retorno de carro)
  Serial.write(10);      // Envía un caracter '\n' (nueva línea)
}
```

Parámetros de entrada:

- **float number1:** hace referencia a la temperatura a ser enviada.
- **float number2:** hace referencia al voltaje de control a ser enviada.

- **Float number3:** hace referencia a la constante Kp a ser enviada.
- **float number4:** hace referencia a la constante Ti a ser enviada.
- **float number5:** hace referencia a la constante Td a ser enviada.
- **float number6:** hace referencia al Setpoint a ser enviado.

Esta función básicamente convierte los valores float en bytes y los envía uno tras otro a través del puerto serial con indicadores de inicio y fin de trama. Los valores float se convierten en bytes para poder transmitirlos de manera consistente y reconstituirlos en el otro extremo de la comunicación. El carácter "A" se utiliza como indicador de inicio de trama, y los caracteres '\r' (retorno de carro) y '\n' (nueva línea) se utilizan para indicar el final de la trama.

Análisis de función para recibir datos desde Matlab (readFromMatlab).

Esta función se encarga de leer datos enviados desde una aplicación MATLAB hacia la ESP32. Los datos enviados desde MATLAB se componen de cuatro valores flotantes, y esta función se encarga de interpretar y almacenar esos valores en las variables correspondientes.

```
void readFromMatlab()
{
  String rein = Serial.readString();
  for (int i =0; i<4; i++)
  {

    u2.b2[i] = rein[i];          // w
    u4.b4[i] = rein[i+4];       // kp
    u5.b5[i] = rein[i+8];      // ti
    u6.b6[i] = rein[i+12];    // td
  }

  instancia.value1 = u2.fval2;       // guarda w
  instancia.value2 = u4.fval4;      // guarda kp
  instancia.value3 = u5.fval5;     // guarda ti
  instancia.value4 = u6.fval6;    // guarda td
}
```

A continuación, se describe paso a paso lo que hace la función:

1. **Lectura de datos recibidos.**

La línea **String rein = Serial.readString();** lee una cadena de caracteres enviada a través del puerto serial y la almacena en la variable **rein**. Esta cadena de caracteres debe contener los valores

enviados desde MATLAB, cada uno con la longitud adecuada. La función **Serial.readString()** espera hasta que se encuentre un carácter de terminación de línea (usualmente '\n').

2. Extracción de valores individuales.

El bucle **for** itera cuatro veces (de **i = 0** a **i < 4**) para extraer cada uno de los cuatro valores individuales contenidos en la cadena recibida desde MATLAB. Cada valor flotante es enviado en formato binario de 4 bytes.

a) **u2.b2[i] = rein[i];** extrae y almacena los primeros 4 bytes en la variable **u2.b2**, que es una unión (o estructura) de tipo **byte** para acceder a los bytes individuales de un valor flotante.
b) **u4.b4[i] = rein[i+4];** hace lo mismo, pero empieza a partir del quinto byte (índice 4) para obtener el segundo valor.
c) **u5.b5[i] = rein[i+8];** obtiene el tercer valor, a partir del noveno byte (índice 8).
d) **u6.b6[i] = rein[i+12];** obtiene el cuarto valor, a partir del decimotercer byte (índice 12).

3. Conversión de bytes a valores flotantes.

Una vez que los bytes de cada valor flotante han sido extraídos y almacenados en las variables **u2.b2**, **u4.b4**, **u5.b5** y **u6.b6**, la función asume que estas variables son uniones o estructuras que permiten acceder a los valores flotantes **fval2**, **fval4**, **fval5** y **fval6** a través de la conversión de bytes a flotantes.

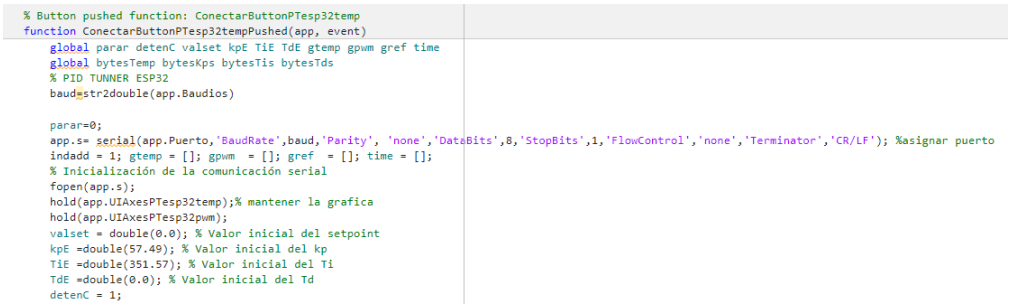
4. Almacenamiento en variables de instancia.

Finalmente, los valores flotantes obtenidos (**fval2**, **fval4**, **fval5** y **fval6**) se almacenan en las variables de la estructura **instancia**. Los valores se asignan a **instancia.value1**, **instancia.value2**, **instancia.value3** e **instancia.value4** respectivamente.

En resumen, esta función se encarga de leer una cadena de caracteres enviada desde MATLAB que contiene cuatro valores flotantes en formato binario de 4 bytes cada uno. Luego, la función extrae y convierte esos valores a flotantes, y finalmente los almacena en una estructura (llamada **instancia**) para su uso posterior en el código. Es importante que el orden de los bytes enviados y recibidos esté sincronizado y que las uniones o estructuras utilizadas para la conversión sean definidas y configuradas correctamente en tu código.

- **Análisis función para recibir datos desde ESP32 en AppDesigner.**

Se describe la función implementada en el botón conectar de AppDesigner, que permitirá recibir y escribir datos por comunicación serial entre AppDesigner y la ESP32.



```
% Button pushed function: ConectarButtonPTesp32temp
function ConectarButtonPTesp32tempPushed(app, event)
    global parar detenC valset kpE TiE TdE gtemp gpwm gref time
    global bytesTemp bytesKps bytesTis bytesTds
    % PID TUNNER ESP32
    baud=str2double(app.Baudios)

    parar=0;
    app.s= serial(app.Puerto,'BaudRate',baud,'Parity', 'none','DataBits',8,'StopBits',1,'FlowControl','none','Terminator','CR/LF'); %asignar puerto
    indadd = 1; gtemp = []; gpwm  = []; gref  = []; time = [];
    % Inicialización de la comunicación serial
    fopen(app.s);
    hold(app.UIAxesPTesp32temp);% mantener la grafica
    hold(app.UIAxesPTesp32pwm);
    valset = double(0.0); % Valor inicial del setpoint
    kpE =double(57.49); % Valor inicial del kp
    TiE =double(351.57); % Valor inicial del Ti
    TdE =double(0.0); % Valor inicial del Td
    detenC = 1;
```

A continuación, se muestra las descripciones de las secciones de la función:

1. Definición de variables globales.

Se definen varias variables globales que almacenan información necesaria para la comunicación y visualización de datos en la interfaz de la aplicación. Estas variables incluyen indicadores para detener el bucle de comunicación (parar y detenC), valores de temperatura, pwm, referencia, tiempo, y variables que almacenarán los datos en formato de bytes para ser enviados al controlador (ESP32).

2. Configuración de la comunicación serial.

Se establece la comunicación serial utilizando la función serial() con los parámetros proporcionados en la aplicación. El puerto y la velocidad (baud) son configurados según los valores ingresados en la interfaz de la aplicación.

3. **Apertura de la comunicación serial.**

Se abre la comunicación serial utilizando **fopen(app.s);**. Esto permite que la aplicación empiece a recibir y enviar datos a través del puerto serial.

```
while (detenC==1)
    if parar==1
        break;
    else
        % Lectura de los bytes recibidos
        numBytes = 23; % Número total de bytes a leer
        receivedData = fread(app.s, numBytes, 'uint8');
        if (receivedData(1)==65) % si empieza con 'A'
            indx1 = receivedData(1);     % caracter de inicio de trama 'A'
            indx2 = receivedData(2:5);   % temperatura
            indx3 = receivedData(6:9);   % pwm
            indx4 = receivedData(10:13);   % kp
            indx5 = receivedData(14:17);   % Ti
            indx6 = receivedData(18:21);   % Td
            indx7 = receivedData(22);    % fin trama '\r' comparar con ascii =>13
            indx8 = receivedData(23);    % '\n' comparar con ascii =>10

            inicio = char(indx1);        % Caracter inicial convertido a char 'A'
            temp = typecast(uint8(indx2), 'single');% convertir dato a tipo single
            v_pwm = typecast(uint8(indx3), 'single');
            kps = typecast(uint8(indx4), 'single');
            tis = typecast(uint8(indx5), 'single');
            tds = typecast(uint8(indx6), 'single');

            gtemp(indadd) = temp;% Añadir datos recibido en gtemp
            gpwm(indadd)  = v_pwm;% Añadir datos recibido en gpwm
            gref(indadd)  = valset;
            time(indadd)  = indadd;%NUEVO
            app.TemperaturaPTesp32temp.Value=gtemp(indadd); % visualizar cuadro de temperatura
            app.ReferenciaPTesp32temp.Value=gref(indadd); % visualizar cuadro de ref
            app.PWMPTesp32temp.Value=gpwm(indadd); % visualizar cuadro de ti
            indadd = indadd+1;
```

4. **Bucle principal.**

Se inicia un bucle principal (**while detenC == 1**) que se ejecutará mientras **detenC** tenga el valor 1. En cada iteración del bucle, se verifica si **parar** tiene el valor 1, lo que indicaría que se desea detener la comunicación.

5. **Lectura de datos recibidos.**

Si no se desea detener la comunicación (**parar** no es 1), se leen los datos recibidos desde el dispositivo Arduino. Se espera recibir una trama que comienza con el carácter 'A' (ASCII 65). Luego se extraen los valores de temperatura, pwm, kp, Ti y Td de los bytes recibidos.

```
            plot(app.UIAxesPTesp32temp,gtemp,'b');  % Temperatura
            plot(app.UIAxesPTesp32temp,gref,'r');       % referencia
            plot(app.UIAxesPTesp32pwm,gpwm,'b');    % PWM
            % Valores de ejemplo para enviar
            tempE = single(valset);   % Temperatura referencia a enviar
            KpsE = single(kpE);   %
            TisE = single(TiE);   %
            TdsE = single(TdE);   %

            % Conversión de los valores a bytes
            bytesTemp = typecast(tempE, 'uint8');
            bytesKps = typecast(KpsE, 'uint8');
            bytesTis = typecast(TisE, 'uint8');
            bytesTds = typecast(TdsE, 'uint8');

             % Trama de envío
            trama = [bytesTemp, bytesKps, bytesTis,bytesTds, 13, 10];
            % Envío de la trama a Arduino
            fwrite(app.s, trama, 'uint8');
            pause(1);
        end
    end
    fclose(app.s);
```

6. **Actualización de la interfaz.**

Los valores de temperatura, pwm, referencia y otros son actualizados en la interfaz de la aplicación. Estos valores se muestran en campos de texto o gráficas, de acuerdo con la interfaz de App Designer.

7. **Envío de datos a la ESP32.**

Se toman los valores de temperatura, kp, Ti y Td y se convierten a bytes utilizando la función **typecast()**. Luego, se construye una trama que contiene estos bytes y caracteres de terminación, y se envía al Arduino a través de **fwrite()**.

8. **Pausa.**

Se realiza una pausa de 1 segundos antes de la próxima iteración del bucle. Esto controla la velocidad de la comunicación y evita un uso excesivo de recursos del sistema.

9. **Cierre de la comunicación serial.**

Una vez que se sale del bucle, se cierra la comunicación serial con **fclose(app.s);**.

Es importante mencionar que este código está diseñado para interactuar con un dispositivo específico que envía y recibe datos a través de comunicación serial. Las partes de la aplicación que interactúan con la interfaz gráfica y los elementos visuales de la aplicación están configuradas en la interfaz visual de App Designer.

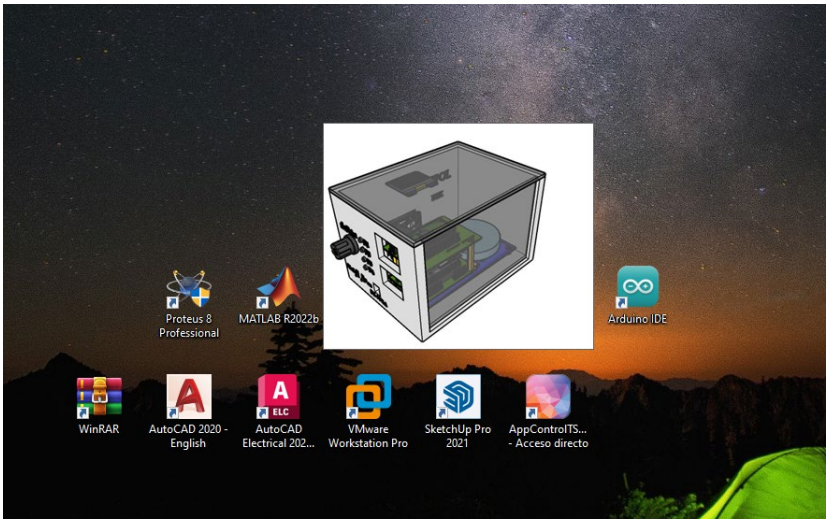
SELECCIÓN DEL CONTROLADOR.

Los controladores disponibles para la planta de velocidad y temperatura son:

Tarjeta de control	Planta	Controlador
Arduino Uno	Temperatura	PID Prueba-Error
		PID Zeigler-Nichols
		Fuzzy-Mamdani
		Realimentación de Estados
	Velocidad	PID Prueba-Error
		Fuzzy-Mamdani
Esp32	Temperatura	PID Prueba-Error
		PID Zeigler-Nichols
		Fuzzy-Mamdani
		Realimentación de Estados
	Velocidad	PID Prueba-Error
		Fuzzy-Mamdani

A manera de ejemplo, si se requiere hacer uso del controlador de Temperatura usando la tarjeta ESP32 por Realimentación de Estados seguimos los siguientes pasos:

- ✓ Abrimos la App de monitoreo y control.



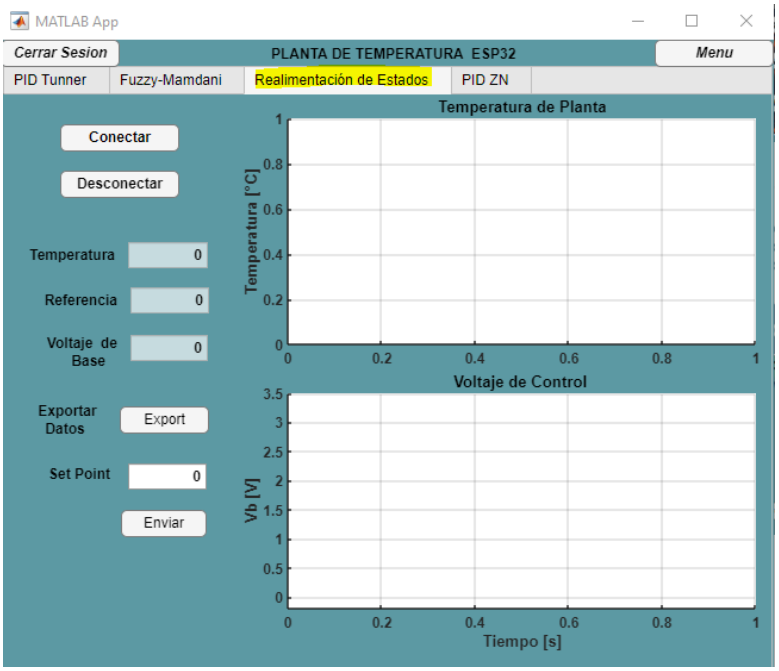
✓ Ingresar el nombre de usuario e identificación, por defecto el usuario es “us2023” y la identificación “pw2023”, finalmente presionar “INGRESAR”

✓ A continuación, se presenta la ventana de selección de planta y controlador. En este caso seleccionamos la planta de Temperatura, la tarjeta de control Esp32, el Baud Rate de 115200 y el puerto que detecte el computador.

***Nota:**

Hay que recordar que solamente se presentara puertos siempre y cuando la tarjeta de control esté conectado al Pc.

- ✓ Siempre y cuando se haya seleccionado correctamente la aplicación le permitirá avanzar hasta la ventana de controladores. Una vez en la ventana de controladores, en este caso se selecciona la pestaña de Realimentación de Estados.

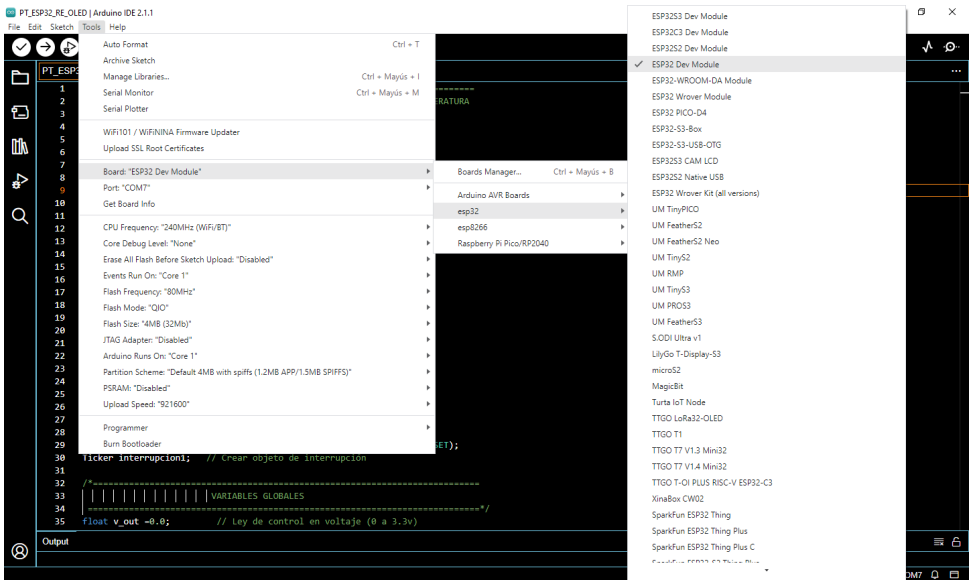


***Nota:**

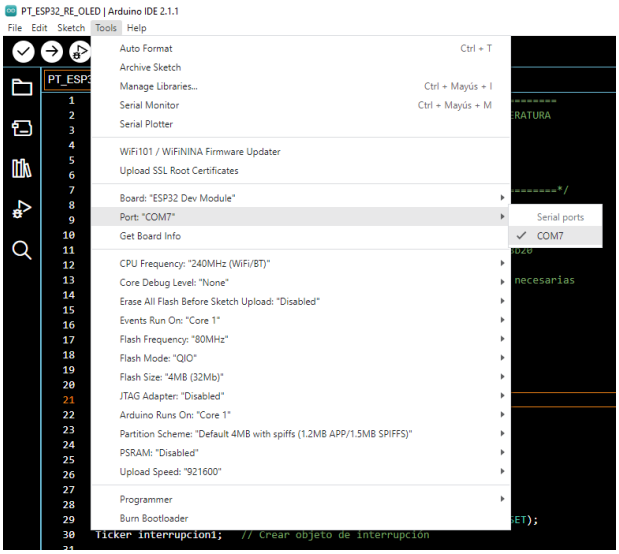
Es necesario "cargar" el código antes de seleccionar en "**Conectar**" y asegurarse de tener seleccionado el **Switch Modo Interfaz**.

❖ Cargar el código en la ESP32.

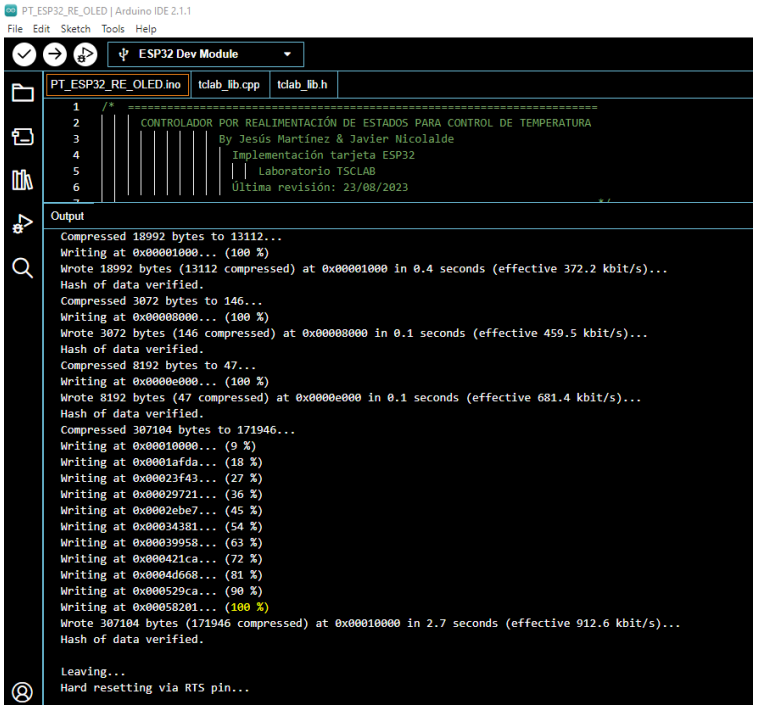
✓ Abrimos la carpeta donde se aloja los códigos, y abrimos Arduino IDE. Seleccionamos la tarjeta usada y el puerto:



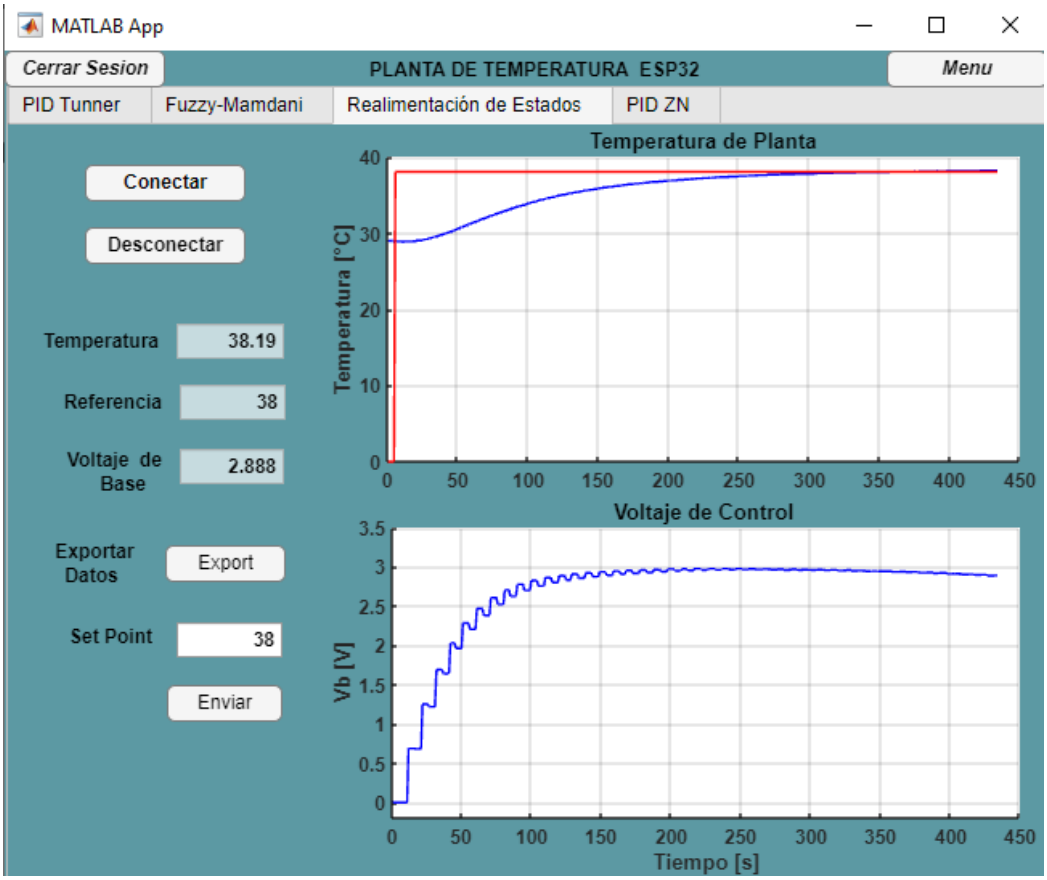
✓ Seleccionamos el puerto:



✓ Enviamos a cargar el código en la tarjeta de control.

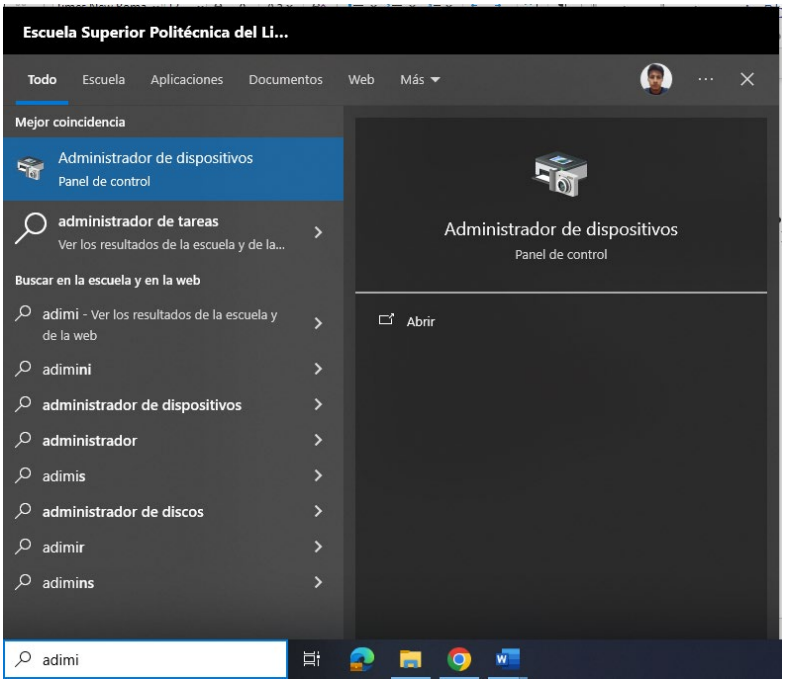


✓ Una vez, cargado el código en la tarjeta se procede a “conectar” para que inicie la comunicación serial y se presente las gráficas en la aplicación.

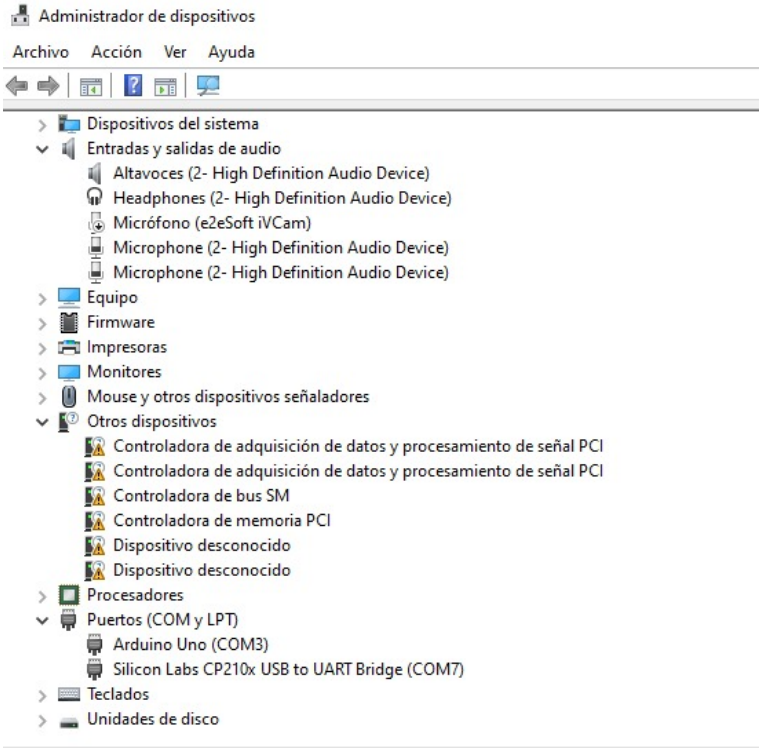


Configuración de puertos en PC.

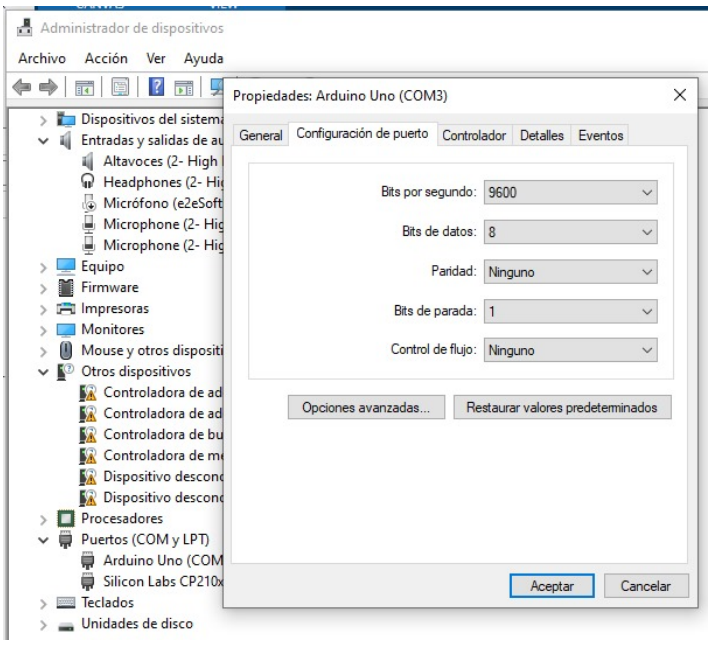
- ✓ Ir a inicio, y abrir el administrador de dispositivos.



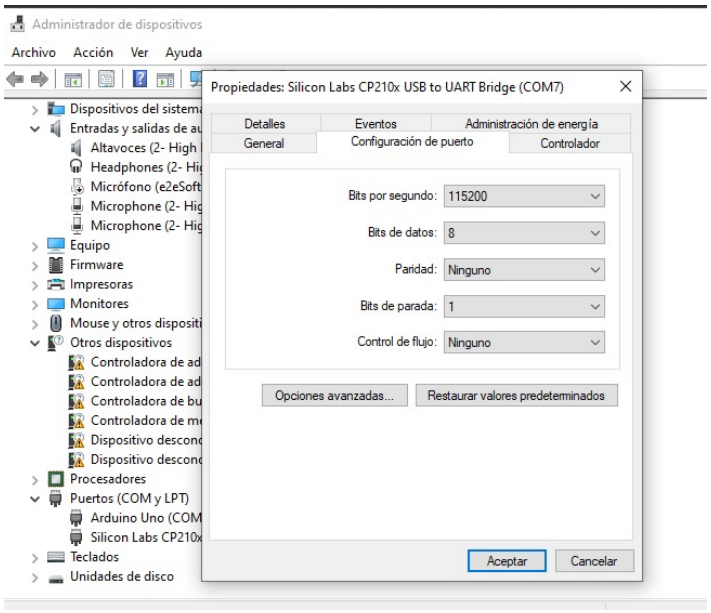
✓ En la opción de Puertos (COM y LPT), y se presentaran los puertos COM disponibles actualmente.



✓ Dar clic derecho en Arduino Uno (COM3 en este caso) y se presentaran las propiedades de comunicación las cuales pueden ser configuradas de acuerdo con la tarjeta de control que este conectada. Baud Rate de 9600 con tarjeta Arduino Uno, mientras que los demás parámetros quedan con la configuración por defecto.



✓ Además, para el uso de la tarjeta de control ESP32 se debe realizar el mismo proceso anterior y en el cual solamente debe cambiar el Baud Rate que es 115200.



✓ Mediante el botón “aceptar” se guardan todas las configuraciones realizadas en los puertos.

LIMITACIONES/ RANGOS DE OPERACIÓN DE CONTROLADORES

- **ESP32-TSCLAB**
  - **CONTROL DE TEMPERATURA**

    Se ha configurado como rango mínimo de temperatura 18 °C, el rango máximo que se permite con la ESP32 es de 38 °C. Esto se debe a que la ESP32 trabaja con 3.3V.

  - **CONTROL DE VELOCIDAD**

    Se ha configurado como rango mínimo de velocidad 0 RPM, el rango máximo que se permite con la ESP32 es de 3500 RPM.

- **ARDUINO UNO-TSCLAB**
  - **CONTROL DE TEMPERATURA**

    Se ha configurado como rango mínimo de temperatura 18 °C, el rango máximo que se permite con Arduino Uno es de 50 °C. Esto se debe a que Arduino trabaja con 5V.

  - **CONTROL DE VELOCIDAD**

    Se ha configurado como rango mínimo de velocidad 0RPM, el rango máximo que se permite con Arduino Uno es de 4000 RPM. Esto se debe a que Arduino trabaja con 5V.