



FACULTAD DE INGENIERÍA

UNIVERSIDAD DE BUENOS AIRES

FACULTAD DE INGENIERÍA

DEPARTAMENTO DE ELECTRÓNICA

Tesis de grado en Ingeniería Electrónica

SISTEMA AUTOMÁTICO DE CALIBRACIÓN DE ANEMÓMETROS EN EL TÚNEL DE VIENTO DEL SERVICIO METEOROLÓGICO NACIONAL

Autor: Cristian Zozimo Aranda Cordero (93631)

Directores: Mg. Ing. Leonardo Martín Carducci (FIUBA)

Dr. Ing. Lucas Sambuco (FIUBA)

Ing. Claudio Arencibia (SMN)

Jurado: Nombre y apellido

Nombre y apellido

Nombre y apellido

Buenos Aires, 26 de agosto de 2024

Resumen

En este trabajo se desarrolló un sistema automatizado para la calibración de instrumentos de medición de viento de tipo ultrasónico, utilizando el túnel de viento del Servicio Meteorológico Nacional. El sistema incluye un datalogger desarrollado con la placa EDU-CIAA, responsable de la adquisición de datos de viento de los sensores, así como de la comunicación y el control del motor del túnel, con un controlador de lazo cerrado PID, reemplazando el sistema manual de medición. Además, se desarrolló una aplicación web con el framework Django y un servidor WebSocket con Django Channels para configurar el datalogger, cargar información relevante de los sensores y del túnel de viento, presentar las mediciones en tiempo real, y procesar los datos utilizando un modelo por comparación, para calcular la incertidumbre y la corrección de las mediciones. Toda la información se guarda de manera estructurada en una base de datos. Este trabajo permitirá al laboratorio del Servicio Meteorológico Nacional alcanzar la capacidad de calibrar sensores de viento, satisfaciendo así la demanda de la red de estaciones automáticas y los sistemas de anemómetros instalados en el país.

Abstract

This work developed an automated system for calibrating ultrasonic wind measurement instruments using the wind tunnel of the National Meteorological Service. The system includes a datalogger developed with the EDU-CIAA board, responsible for acquiring wind data from the sensors, as well as for communication and control of the tunnel motor with a closed-loop PID controller, replacing the manual measurement system. Additionally, a web application was developed using the Django framework and a WebSocket server with Django Channels to configure the datalogger, load relevant information about the sensors and the wind tunnel, display real-time measurements, and process the data using a comparative model to calculate uncertainty and measurement corrections. All information is stored in a structured manner in a database. This work will enable the laboratory of the National Meteorological Service to achieve the capability to calibrate wind sensors, thus meeting the demand of the network of automatic stations and anemometer systems installed across the country.

Agradecimientos

Índice general

1. Implementación del Datalogger	1
1.1. Desarrollo del hardware	2
1.1.1. Módulo de alimentación eléctrica	3
1.1.2. Módulo de adquisición de datos de viento	5
1.1.3. Módulo de comunicación Ethernet/SPI	7
1.1.4. Circuito PWM	8
1.1.4.1. Simulación del circuito	10
1.1.4.2. Implementación del circuito	13
1.1.5. Circuito de adquisición de tensión del variador	19
1.1.5.1. Simulación del circuito	19
1.1.5.2. Implementación del circuito	21
1.1.6. Circuito de LEDs indicadores	22
1.2. Diseño y construcción del PCB	24
1.3. Desarrollo del firmware	30
1.3.1. Configuración y conexión con los servidores	33
1.3.1.1. Servidor NTP	34
1.3.1.2. Servidor WebSocket	36
1.3.2. Muestreo, adquisición y procesamiento	40
1.3.3. Transmisión al servidor	44
1.3.4. Sistema de control PID	46
A. Datalogger y sensores	49
A.1. Principio de medición para sensores de ultrasonido	49
A.2. Especificaciones técnicas y hojas de datos	51
B. Software	52
B.1. Algoritmo del generador de trayectoria	52

B.2. Comandos entre el servidor WebSocket y el datalogger	54
C. Cálculo de incertidumbre	55
C.1. Ejemplo de Presupuesto de incertidumbre para un punto	56

Índice de figuras

1.1.	Diagrama en bloques del sistema electrónico.	1
1.2.	Placa de desarrollo EDU CIAA	2
1.3.	Diagrama en bloques de la EDU-CIAA	2
1.4.	En (a) se muestra el sensor Vaisala modelo WMT700, en (b) se muestran el sensor Delta OHM modelo HD51.3D.	3
1.5.	Regulador de voltaje DC-DC basado en el chip LM2596.	4
1.6.	Esquemático del sistema de alimentación con módulo LM2596, diodo Zener, divisor resistivo y filtros de bypass para el LM358.	5
1.7.	Esquema de conexión de datos y alimentación del sensor HD51.3D con un Data-logger o un PLC. [2]	6
1.8.	Puerto RS-485 integrado en la EDU-CIAA	6
1.9.	Módulo convertidor de RS485 a UART (TTL)	6
1.10.	Esquemático del módulo MAX485 para la recepción de datos del sensor bajo calibración.	7
1.11.	Módulo W5100 utilizado para realizar la conectividad a internet a través de la interfaz SPI del microcontrolador.	8
1.12.	Esquemático del módulo W5100 para la comunicación bidireccional con distintos servidores.	8
1.13.	Esquemático circuital del tablero de control del tunel de viento.	9
1.14.	Circuito diseñado para obtener distintos niveles de tensión mediante la variación del ciclo de trabajo de una señal cuadrada (PWM).	11
1.15.	Señales a la salida del primer filtro LP (curva verde) y segundo filtro LP (curva azul).	11
1.16.	Señales simuladas para un ciclo de trabajo del 100 %.	12
1.17.	Señales simuladas para un ciclo de trabajo al 75 % de la señal de entrada.	13
1.18.	Señales simuladas para un ciclo de trabajo al 55 % de la señal de entrada.	13

1.19.	Banco de medición para caracterizar el circuito PWM.	14
1.20.	Medición de la señal PWM (rojo) con un ciclo de trabajo de (1 %) y la señal continua a la salida del amplificador operacional (azul) iguala 53,5 mV.	15
1.21.	Medición de la señal PWM (rojo) con un ciclo de trabajo de (25 %) y la señal continua a la salida del amplificador operacional (azul) iguala 900 mV.	15
1.22.	Medición de la señal PWM (rojo) con un ciclo de trabajo de (50 %) y la señal continua a la salida del amplificador operacional (azul) iguala 1,8 V.	16
1.23.	Medición de la señal PWM (rojo) con un ciclo de trabajo de (75 %) y la señal continua a la salida del amplificador operacional (azul) iguala 2,65 V.	16
1.24.	Medición de la señal PWM (rojo) con un ciclo de trabajo de (100 %) y la señal continua a la salida del amplificador operacional (azul) iguala 3,56 V.	17
1.25.	Mediciones para ciclos de trabajo en modo Ascendente, del nivel de tensión en el variador y la velocidad de viento.	18
1.26.	Mediciones para ciclos de trabajo en modo descendente, del nivel de tensión en el variador y la velocidad de viento.	18
1.27.	Circuito diseñado para tomar muestras de la señal que se suministra al variador de velocidad y se las envía a una canal analógico-digital de la EDU-CIAA.	19
1.28.	Señales simuladas a la salida del variador de velocidad y a la entrada del pin analógico-digital de la EDU-CIAA para un ciclo de trabajo del 100 %.	20
1.29.	Señales simuladas a la salida del variador de velocidad y a la entrada del pin analógico-digital de la EDU-CIAA para un ciclo de trabajo del 55 %.	20
1.30.	Mediciones a la entrada pin <i>VADC</i> del variador y en el pin <i>ADC_CH2</i> de la EDU-CIAA, para un barrido ascendente del ciclo de trabajo.	21
1.31.	Mediciones a la entrada pin <i>VADC</i> del variador y en el pin <i>ADC_CH2</i> de la EDU-CIAA, para un barrido descendente del ciclo de trabajo.	22
1.32.	Esquemático de los circuitos que encienden y apagan dos LEDs, verde y amarillo, cada vez que se recibe un dato de los anemómetros, tanto del patrón como del que está bajo calibración.	23
1.33.	Esquemático del circuito de un LED dual que se enciende en rojo cuando el sistema no está conectado al servidor Websocket y en verde cuando hay una conexión estable con el servidor.	24
1.34.	Template disponible para la realización de ponchos para la EDU-CIAA.	25
1.35.	Diseño del esquemático que integra los módulos desarrollados.	26

1.36. Diseño PCB del <i>shield</i> que contiene todos los módulos descritos en las secciones anteriores.	27
1.37. Vista en 3D de la capa BOTTOM.	28
1.38. Vista en 3D de la capa TOP.	29
1.39. Vista superior de la placa <i>shield</i> conectada a la EDU-CIAA	30
1.40. Lanzador de aplicaciones integradas para el desarrollo de firmware para la EDU-CIAA.	31
1.41. Estructura de carpetas, del proyecto Datalogger	31
1.42. Lista de regiones con su respectiva prioridad.	32
1.43. Maquina de estados principal, que gestiona la ejecución del resto de las maquinas de estado.	33
1.44. Modos de configuración para el módulo W5100.	34
1.45. Máquina de estados para configurar el RTC del datalogger a través de un servidor NTP.	35
1.46. Máquina de estados para establecer una conexión bidireccional con un servidor WebSocket.	38
1.47. Máquina de estados para mantener la conexión con el servidor y elevar un raise en caso de que se reciba información del servidor.	39
1.48. Máquina de estados para procesar los mensajes recibidos del servidor WebSocket.	40
1.49. Máquina de estados para tomar muestras de los sensores viento.	42
1.50. Máquina de estado para tomar muestras de la tensión del alimentacion y del variador.	43
1.51. Máquina de estados para acumular muestras en una tabla y realizar un preprocesamiento de esos datos.	44
1.52. Máquina de estados para gestionar la transmisión de datos al servidor WebSocket.	45
1.53. Máquina de estados que ejecuta el control PID con valores de referencia enviados por el servidor.	48
A.1. Esquema de los transductores y el tiempo de viaje de un pulso ultrasónico. [2]	49

Índice de tablas

1.1. Mediciones de tensión y corriente de la fuente del variador para distintas resistencias de carga.	10
1.2. Diferencias entre los tipos de máquinas de estados.	32
A.1. Especificaciones de sensor de viento, marca Delta Ohm, modelo HD51.3.	51
B.1. Listado de comandos transmitidos por el servidor WebSocket al datalogger para configurar los parámetros seleccionados por el usuario.	54
C.1. En la tabla se indica el calculo de incertidumbre para un punto en particular 10 m s^{-1}	56

Capítulo 1

Implementación del Datalogger

Este capítulo se centra en el diseño y la programación del datalogger, que será utilizado en el banco de medición para calibrar los anemómetros de ultrasonido. Este sistema está basado en la placa de desarrollo EDU-CIAA [8], con el agregado de un *shield* específico para la aplicación. El sistema se describe en la Figura 1.1 y consta de distintos módulos, a saber: un módulo para la adquisición de muestras de viento mediante el protocolo RS485, un módulo para la recepción y transmisión de datos a través de Ethernet, un módulo de alimentación de 12 V, y un módulo PWM con una etapa de filtrado seguida de otra de amplificación para entregar la señal de control del túnel de viento. Además, se toman muestras de esta señal y de la fuente para monitorear por software el comportamiento del sistema.

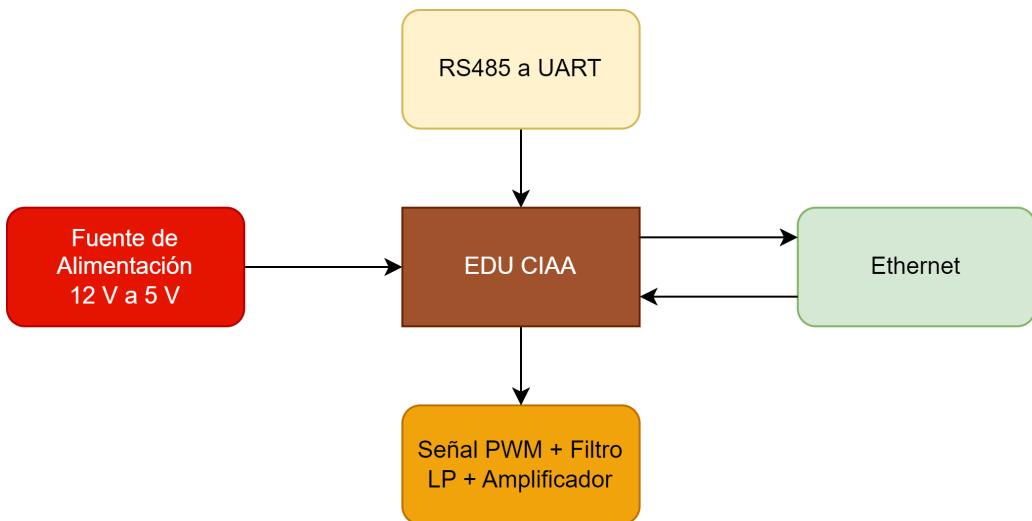


Figura 1.1: Diagrama en bloques del sistema electrónico.

1.1. Desarrollo del hardware

Para el desarrollo del Hardware, en una primera etapa se realizó un proceso de investigación de hojas de datos, manuales, artículos y videos de internet para ver qué módulos y componentes comprar para la implementación de la aplicación. Además, se utilizaron herramientas como LT Spice para la simulación de circuitos y KiCad para el diseño de PCB.

La placa de desarrollo EDU-CIAA-NXP, se muestra en la Figura 1.2, es una versión de bajo costo de la CIAA-NXP [8] pensada para la enseñanza universitaria, terciaria y secundaria.

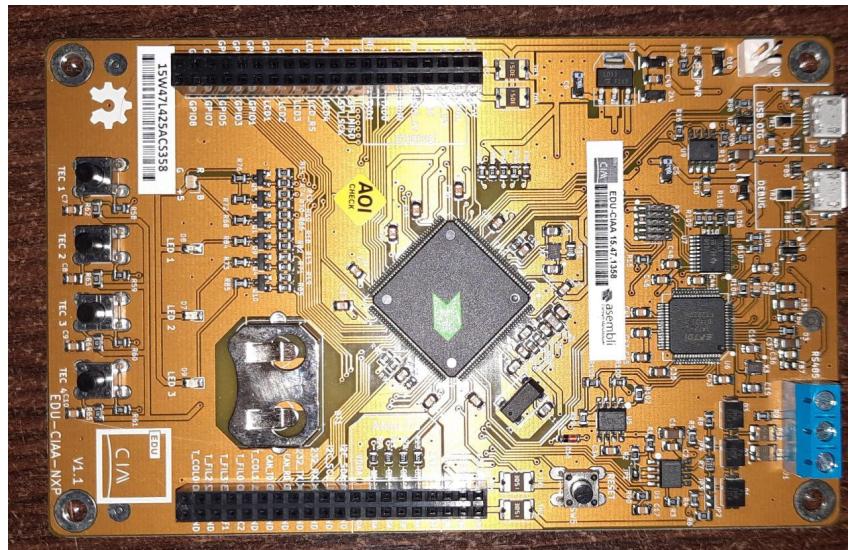


Figura 1.2: Placa de desarrollo EDU CIAA

En la Figura 1.3 se describe un diagrama en bloques de los periféricos de la placa de desarrollo.

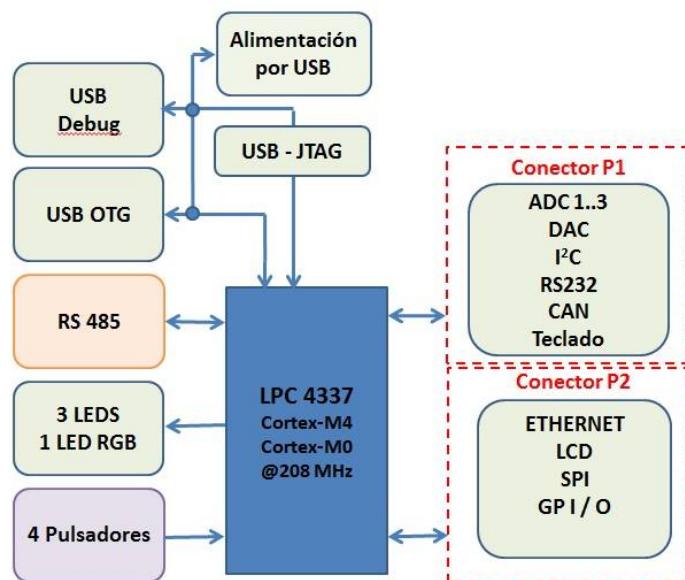


Figura 1.3: Diagrama en bloques de la EDU-CIAA

El SMN proporcionó los sensores de viento de ultrasonido. Se trabajó con los siguientes anemómetros: el modelo WMT700 de la marca VAISALA, Figura 1.4a , utilizado como anemómetro patrón, y el modelo HD51.3D de la marca Delta OHM, Figura 1.4b, utilizado como anemómetro bajo calibración. Ambos sensores operan con la interfaz eléctrica RS485 y vienen con un cable de aproximadamente 15 metros, puesto que estos sensores se instalan en torres de 10 metros de altura.



Figura 1.4: En (a) se muestra el sensor Vaisala modelo WMT700, en (b) se muestran el sensor Delta OHM modelo HD51.3D.

Algunas especificaciones técnicas de este tipo de anemómetros se muestran en la tabla A.1. Las más relevantes para el proceso de calibración son, el rango de medición, la resolución, la precisión, los protocolos de comunicación, la alimentación eléctrica y el intervalo de tiempo mínimo con el que puede tomar muestras.

1.1.1. Módulo de alimentación electrica

Los anemómetros funcionan con una alimentación de 12 V para el sistema principal y 24 V para el sistema de calefacción, ya que estos sensores pueden ser expuestos a entornos con temperaturas por debajo de cero grados. Por otro lado, los módulos Ethernet, RS-485 y los amplificadores operacionales trabajan con 5 V, mientras que el resto de la placa de desarrollo opera con 3,3 V. Por esta razón, se decide alimentar todo el sistema con una fuente de 12 V,

para garantizar el funcionamiento de los anemómetros en condiciones normales de temperatura dentro del laboratorio. Para lograr estos niveles de tensión, se optó por un regulador basado en el chip LM2596S (Step-Down), como se indica en la Figura 1.5. Se conectan 12 V a la entrada y se ajusta el potenciómetro hasta obtener 5 V a la salida. Esta salida estará disponible para conectar el resto de los módulos. Cabe aclarar que, si bien la EDU-CIAA tiene una salida de 5 V, esta no proporciona la corriente suficiente para alimentar todos los módulos, ya que está limitada por los 500 mA que entrega un puerto USB 2.0. En contraste, el regulador DC-DC basado en el LM2596 puede entregar hasta 3 A, satisfaciendo así las necesidades de corriente de todos los módulos conectados.



Figura 1.5: Regulador de voltaje DC-DC basado en el chip LM2596.

En la Figura 1.6 se observa el esquemático del sistema, donde se muestra cómo se conecta el módulo LM2596 con el resto de los componentes. Este esquemático incluye un divisor resistivo y un diodo Zener de 3,3 V para limitar y adaptar la tensión de 12 V a un rango de 3,3 V, ya que se toman muestras de la tensión de alimentación a través del canal analógico-digital ADC_CH1 de la EDU-CIAA. Esta información de la tensión se envía junto con los datos de los anemómetros, permitiendo un monitoreo en tiempo real del sistema de alimentación. Además, se han agregado filtros de bypass a la alimentación del amplificador operacional LM358 para asegurar una operación estable y con bajo ruido.

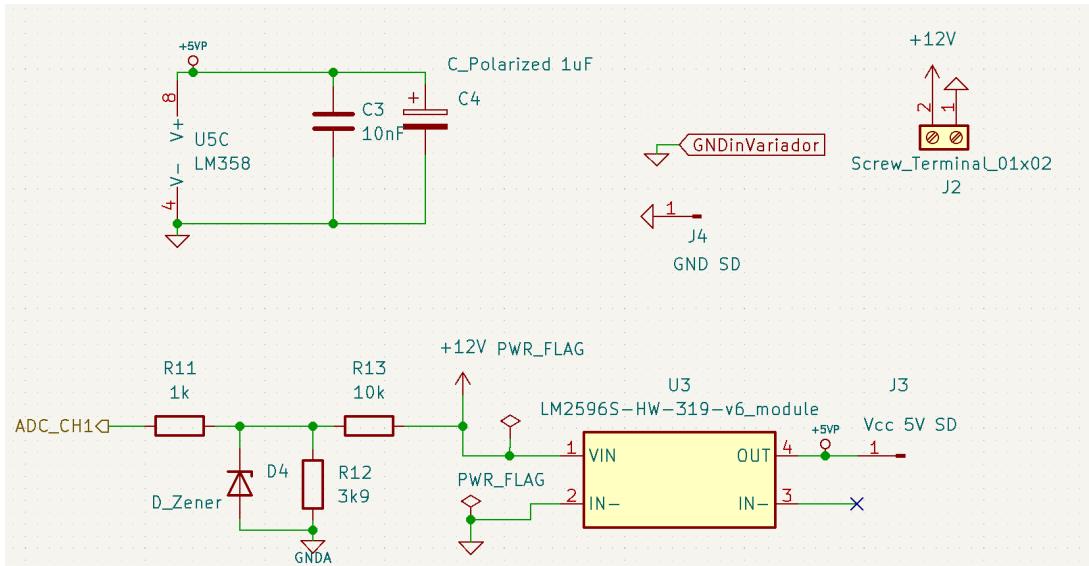


Figura 1.6: Esquemático del sistema de alimentación con módulo LM2596, diodo Zener, divisor resistivo y filtros de bypass para el LM358.

1.1.2. Módulo de adquisición de datos de viento

En la sección ?? del capítulo ?? describimos los distintos tipos de sensores para medir la intensidad y dirección del viento. En particular se trabajó con anemómetros de ultrasonido cuya interfaz electrónica utiliza el protocolo RS-485 [3], el mismo es un protocolo diferencial ampliamente utilizado en entornos industriales debido a su robustez y confiabilidad. Utiliza tres cables (A, B y GND) y transmite datos mediante la diferencia de tensión entre A y B, ofreciendo alta inmunidad al ruido. En la Figura 1.7 se muestra un esquema de conexión entre el sensor y el sistema. En particular, nos interesa dos cables para la alimentación V+ (12 V) y V- (GND), y tres cables para los datos, DATA+ (A), DATA- (B) y GND. Puesto que tanto el sensor patrón como el sensor bajo calibración trabajan con el puerto RS485 y la placa de desarrollo EDU-CIAA cuenta con un solo puerto RS485, se va a utilizar un adaptador para así tener disponible dos puertos RS485.

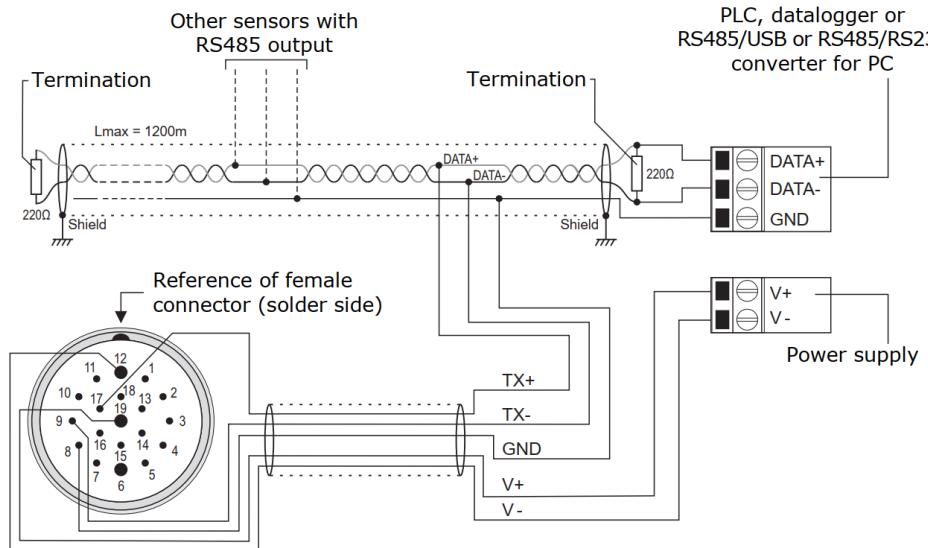


Figura 1.7: Esquema de conexión de datos y alimentación del sensor HD51.3D con un Datalogger o un PLC. [2]

En la Figura 1.8, se muestra el puerto RS485 integrado de la EDU-CIAA, a esta bornera se conectó directamente el anemómetro patrón WMT700. El sensor envía datos en modo ráfaga cada segundo, y este puerto permite una comunicación directa y eficiente sin necesidad de adaptadores adicionales.

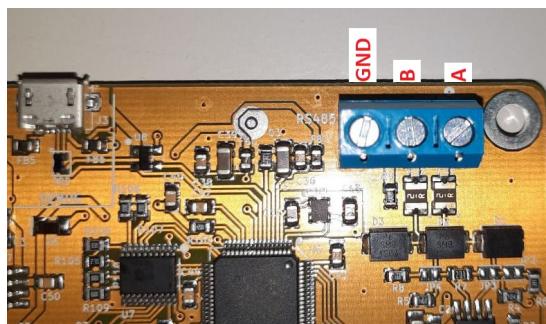


Figura 1.8: Puerto RS-485 integrado en la EDU-CIAA

Para conectar el anemómetro bajo calibración se utilizó el módulo MAX485, Figura 1.9, que convierte las señales RS-485 a niveles de la UART (TTL).

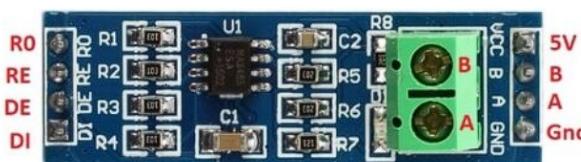


Figura 1.9: Módulo convertidor de RS485 a UART (TTL)

El módulo se alimentó con 5 V suministrados por el regulador LM2596S. Desde un extremo, se conectaron los cables A y B del sensor RS-485, mientras que del otro lado se conectó a la UART-RS232 de la EDU-CIAA. Para adaptar la señal TTL a la placa de desarrollo, se utilizó un resistor de $1\text{ k}\Omega$ entre el pin RO del MAX485 y la UART-RS232. Los pines DE y RE del MAX485 se conectaron a GND para permitir la recepción de datos desde el sensor hacia la placa. El pin DI no se conectó, ya que el sensor no requiere recibir datos o comandos desde la EDU-CIAA.

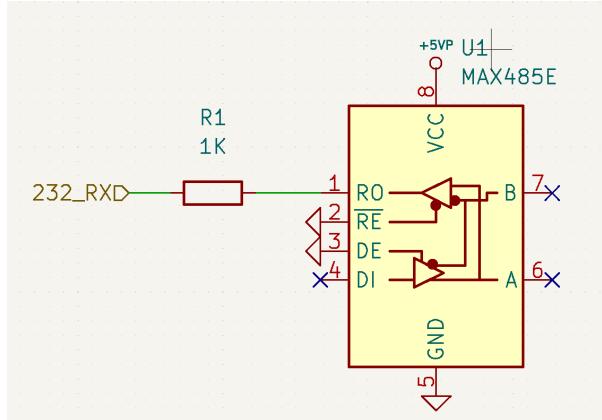


Figura 1.10: Esquemático del módulo MAX485 para la recepción de datos del sensor bajo calibración.

1.1.3. Módulo de comunicación Ethernet/SPI

Para comunicar el sistema embebido con distintos servidores, como el servidor WebSocket desarrollado en la sección ??, así como servidores NTP para sincronizar la fecha y hora, se utilizó el módulo W5100, Figura 1.11. Este módulo emplea el protocolo TCP/IP para proporcionar una comunicación estable. Se alimenta con 5 V y utiliza una interfaz SPI para comunicarse con la placa de desarrollo. Una de las principales ventajas del módulo W5100 [9] es su capacidad para manejar cuatro sockets independientes, lo que permite conectarse a hasta cuatro servidores distintos de manera concurrente. Esto mejora la eficiencia y la flexibilidad del sistema, facilitando la integración con múltiples servicios de red simultáneamente.

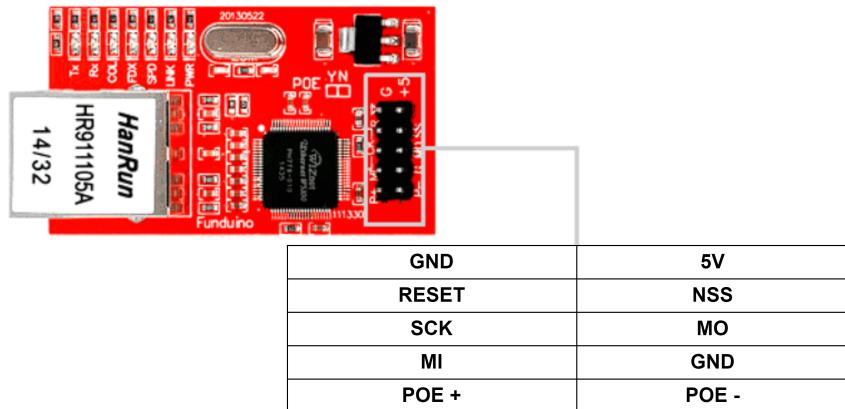


Figura 1.11: Módulo W5100 utilizado para realizar la conectividad a internet a través de la interfaz SPI del microcontrolador.

Como se muestra en la Figura 1.12, el módulo W5100 se alimentó con 5 V y se conectaron los pines del módulo a la EDU-CIAA de la siguiente manera: el pin MO a SPI_MOSI, el pin MI a SPI_MISO, y el pin SCK a SPI_SCK. Además, se conectó el pin NSS al GPIO1 para poder reiniciar el módulo cuando sea necesario por software. El resto de los pines del módulo quedaron sin conectar. Con esta configuración, se logró una comunicación bidireccional, permitiendo transmitir datos de los sensores de viento desde el datalogger hacia la aplicación web, así como recibir comandos desde la aplicación web hacia el datalogger.

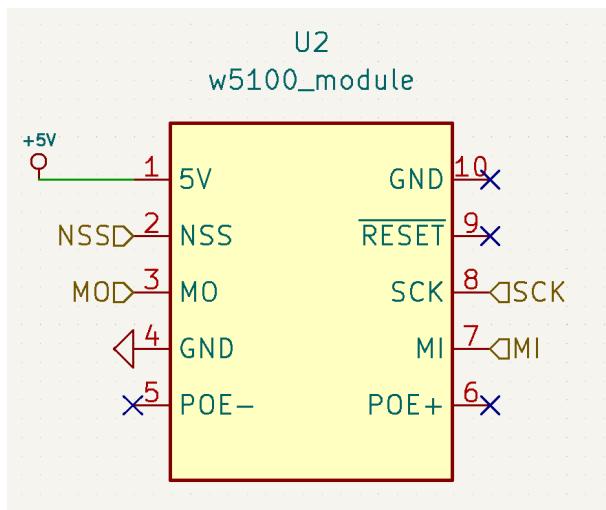


Figura 1.12: Esquemático del módulo W5100 para la comunicación bidireccional con distintos servidores.

1.1.4. Circuito PWM

En esta sección, se presenta el desarrollo de un circuito que permite controlar la potencia del motor del túnel de viento a través de modulación por ancho de pulso (PWM, por sus siglas en

inglés). Primero, se describe el funcionamiento original del variador de velocidad. En la Figura 1.13 se muestra un esquema del tablero de control que se conecta con el variador de velocidad del túnel de viento. El tablero posee dos pulsadores: uno de marcha (BT1-NC) y otro de parada (BT2-NA), que permiten encender y apagar, respectivamente, el conjunto variador-motor del túnel de viento. La velocidad del viento se regula manualmente mediante dos potenciómetros lineales: $P1 = 1\text{k}\Omega$ y $P2 = 500\Omega$. Además, el circuito incluye una resistencia $R1 = 1,2\text{k}\Omega$ conectada a una fuente de alimentación $VCC = 10\text{V}$, la cual es proporcionada por el tablero. Estos componentes, junto con los potenciómetros, conforman un divisor de tensión variable que permite ajustar el voltaje en un rango de 0V a 10V . Esta tensión se aplica al pin de entrada analógico-digital $VADC$ del tablero. Este nivel de baja tensión obtenido se amplifica y adapta a través de su variador a los niveles de tensión continua requeridos por el motor en el rango de 0V a 300V . El nivel de tensión aplicado al motor se puede visualizar en un voltímetro de corriente continua, V , que incluye el tablero.

Se realizó una medición con un amperímetro en serie con la resistencia $R1$ y se obtuvo que la corriente que pasa por $R1$ cuando se ajusta el potenciómetro para obtener una $VADC = 3,56\text{V}$ es de $140\mu\text{A}$. Cuando se ajusta el potenciómetro para obtener una $VADC = 5\text{V}$, la corriente es de $200\mu\text{A}$. Con estas mediciones podemos concluir que el pin $VADC$ del variador tiene una resistencia de entrada de alta impedancia, ya que el consumo de corriente no supera los $500\mu\text{A}$.

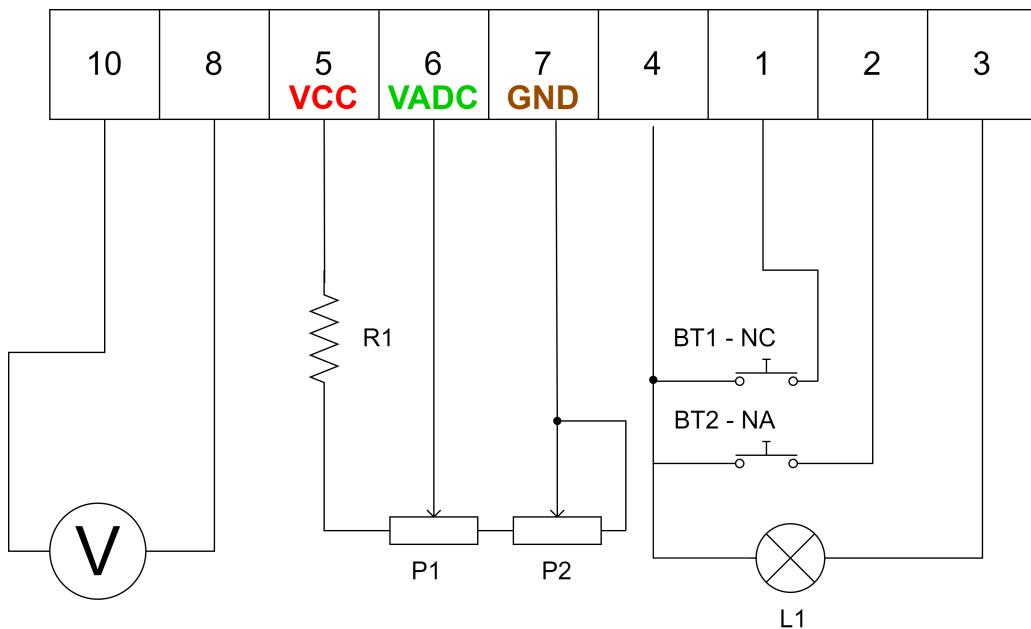


Figura 1.13: Esquemático circuital del tablero de control del tunel de viento.

Para lograr controlar el nivel de tensión $VADC$ por medio de software, se remplazó los dos potenciómetros y el resistor $R1$ por un circuito diseñado en la siguiente sección. Por otro lado,

se hizo un relevamiento de la corriente máxima que puede entregar la fuente del variador VCC conectando distintos valores de resistencia entre VCC y GND . Los resultados se muestran en la tabla 1.1. Como se observa en la tabla, a medida que disminuimos la carga R de prueba, tanto la tensión, como la corriente que puede entregar la fuente VCC disminuyen. Se alcanza la corriente maxima de 100 mA a partir de una carga de 100Ω , cuya tensión de la fuente cae a 2,29 V. Tomando como referencia el consumo de corriente y la tensión que precisa el datalogger, esta fuente VCC del tablero, no es capaz de suministrar la corriente necesaria y mantener la tensión constante, por esta razón se optó por alimentar el sistema como se explicó en la sección 1.1.1

R [Ω]	VCC [mV]	I_R[mA]
47K	9,9	2
10K	9,52	5,9
1K	9,29	9,4
560	7,55	13,48
330	5,63	16,97
100	2,29	100
47	0,859	100

Tabla 1.1: Mediciones de tensión y corriente de la fuente del variador para distintas resistencias de carga.

1.1.4.1. Simulación del circuito

En la Figura 1.14 se muestra el diseño del circuito del PWM [5], que reemplaza al divisor de tensión formado por los potenciómetros $P1$, $P2$ y la resistencia $R1$, mostrado en la Figura 1.13. La señal de entrada es una señal PWM (cuadrada) obtenida desde un pin de la EDU-CIAA. Esta señal, para un ciclo de trabajo del 100 %, tiene una amplitud de 3,3 V acorde con los niveles digitales de tensión de la EDU-CIAA. Luego, esta señal pasa por dos etapas de filtrado, cada una con una frecuencia de corte de 1592 Hz. La tensión vFilter1 (señal verde) en la primera etapa del filtro pasabajo tiene un tiempo de crecimiento corto, del orden de 8 ms, pero presenta mayor rizado. La segunda etapa del filtro genera la tensión voutFilter (señal azul), que tiene un mayor tiempo de crecimiento, del orden de 12 ms, pero con un rizado mucho menor. Con esta doble etapa de filtro pasa bajo, se obtiene una señal continua con menor rizado, como se muestra en la Figura 1.15.

Posteriormente, la señal rectificada ingresa a una etapa de amplificación no inversora, implementada con el amplificador operacional LM358. Este nivel de continua es el que se conecta al pin $VADC$ del variador de velocidad del túnel de viento, pero para la simulación se reemplaza

con una carga de $1\text{ M}\Omega$. La señal a la salida del amplificador tiene una ganancia de $\frac{3,5\text{ V}}{3,3\text{ V}} \approx 1,06$. Se decidió amplificar hasta 3,5 V porque se encontró que la máxima velocidad que logra alcanzar el túnel, de 25 m s^{-1} , se alcanza con 3,5 V a la entrada del *VADC*. Esta limitación viene dada por la mecánica del motor, por lo tanto, en estas condiciones no es necesario amplificar a valores mayores de 3,5 V, ya que el túnel no incrementa su velocidad de viento.

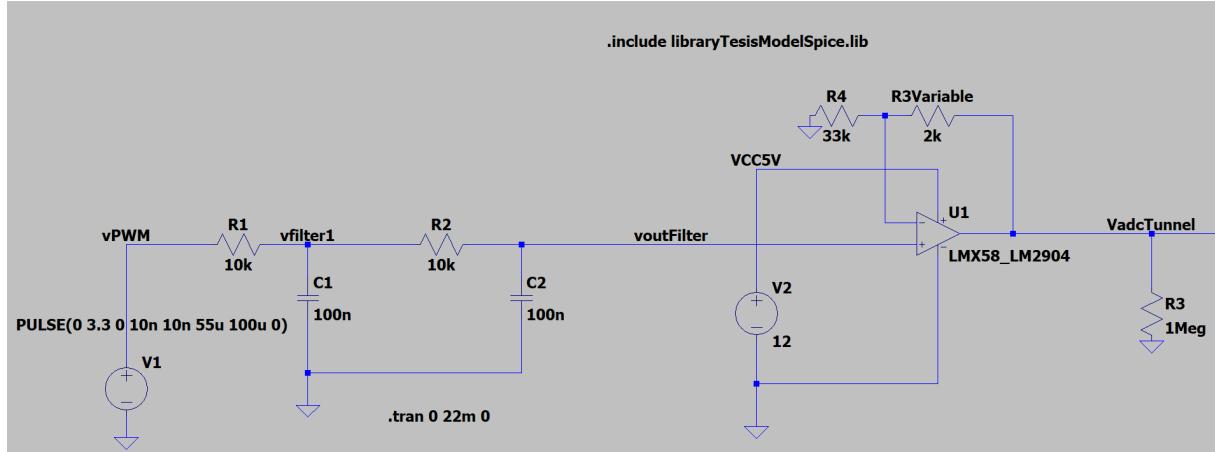


Figura 1.14: Circuito diseñado para obtener distintos niveles de tensión mediante la variación del ciclo de trabajo de una señal cuadrada (PWM).

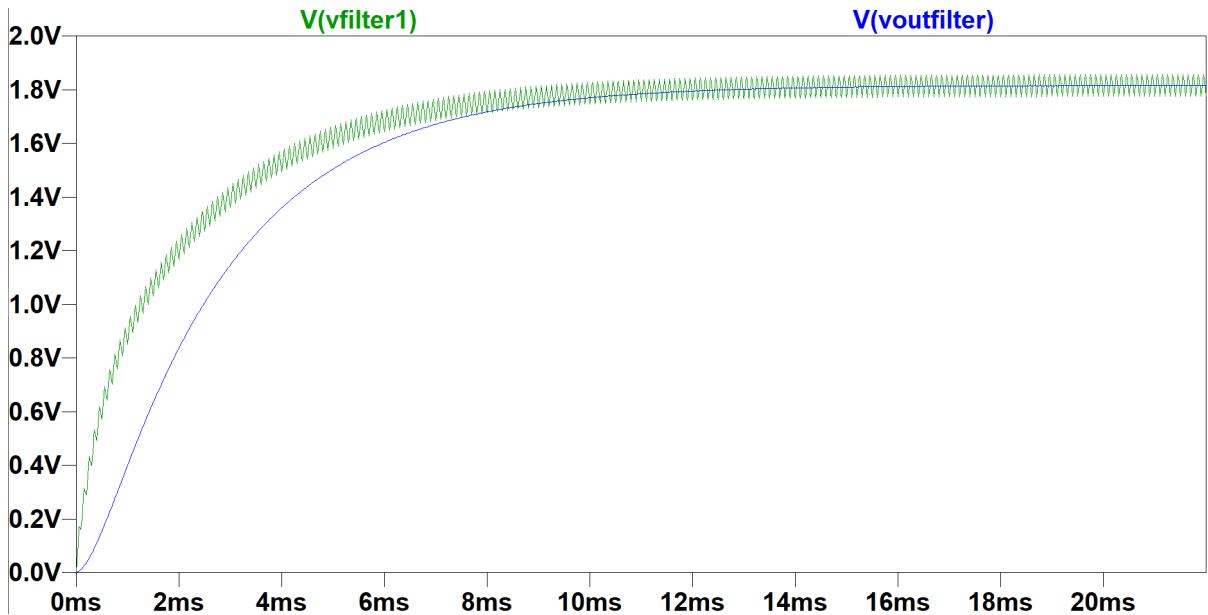


Figura 1.15: Señales a la salida del primer filtro LP (curva verde) y segundo filtro LP (curva azul).

Se dejó un resistor variable (Preset) $R3 = 10\text{ k}\Omega$ para poder modificar la ganancia del amplificador si fuera necesario. En este caso se lo ajustó para alcanzar 3,5 V a la salida del operacional. Sin embargo, para cuando sea mejorada la mecánica y transmisión del motor del túnel de viento,

podrá ser recalibrado para llegar a mayores niveles de tensión, y por ende mayores velocidades de viento(talvez lo pongo abajo).

En la Figura 1.16 se muestra el resultado de la simulación para un pulso vPWM (señal verde) de $100\text{ }\mu\text{s}$ de periodo (10 kHz) y una amplitud de 3,3 V con un ciclo de trabajo del 100 %. La señal vOutFilter (señal en azul) es la que ingresa al amplificador operacional alcanzando su valor máximo en 3,3 V y la señal vAdcTunnel (señal en rojo) se mide a la salida del amplificador operacional alcanzando un tension máxima de 3,5 V lo que produce la máxima velocidad del motor.

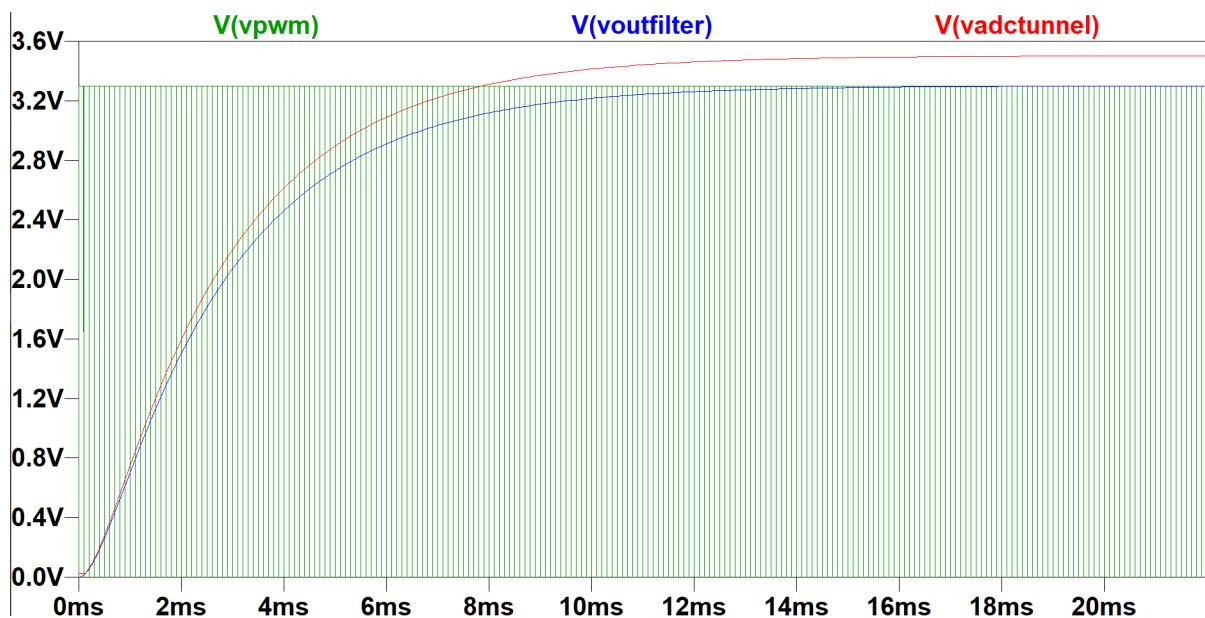


Figura 1.16: Señales simuladas para un ciclo de trabajo del 100 %.

En la simulación, al regular el ciclo de trabajo de la señal de entrada, se obtuvieron distintos niveles de tensión continua. Esto se puede observar en las Figuras 1.17 y 1.18, que muestran los resultados para ciclos de trabajo del 75 % y 55 %, respectivamente.

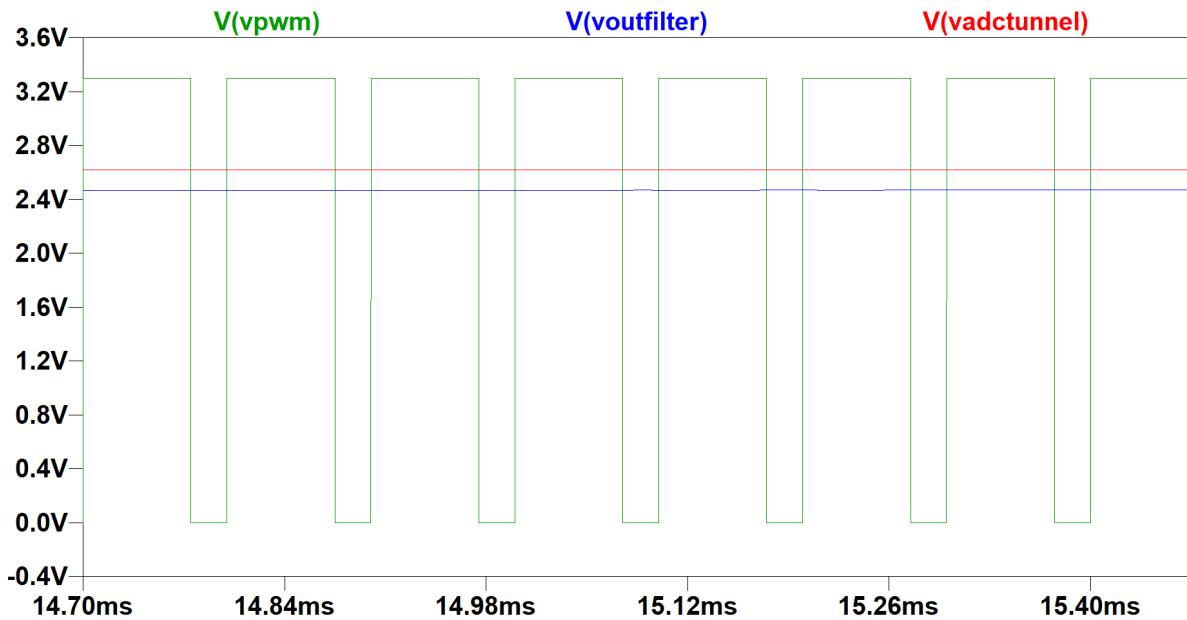


Figura 1.17: Señales simuladas para un ciclo de trabajo al 75 % de la señal de entrada.

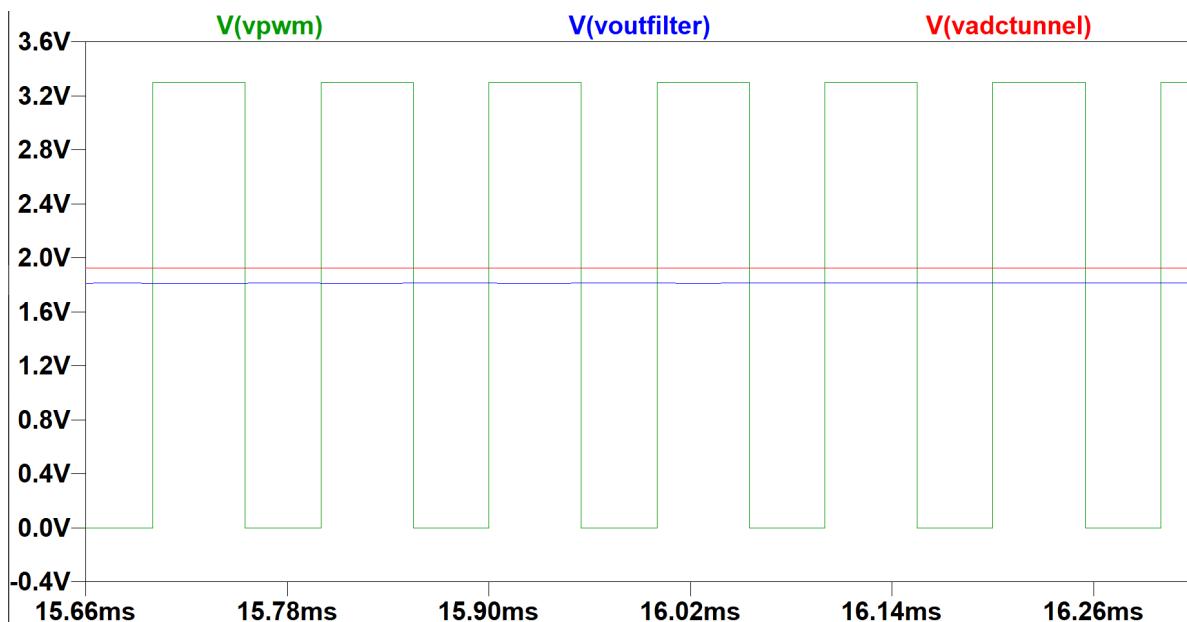


Figura 1.18: Señales simuladas para un ciclo de trabajo al 55 % de la señal de entrada.

1.1.4.2. Implementación del circuito

(cambiar a ensayos del circuito) Se implementó el circuito de la Figura 1.14 en un *proto-board* de prueba con componentes *Through-Hole Technology* (THT), conectado a la EDU-CIAA, como se muestra en la Figura 1.19. Para comprobar y caracterizar el comportamiento del PWM, se utilizaron distintos valores de ciclo de trabajo programados desde el firmware del microcontrolador. El ciclo de trabajo de la señal cuadrada se modificó desde 1 hasta 255, equivalente a

un rango de 0 % a 100 %.

Para realizar estas mediciones, se utilizó un osciloscopio RIGOL DS2302A. En el canal 1 se midió la señal PWM que sale de un pin digital de la placa de desarrollo y se conecta a la entrada del circuito PWM, esta señal tiene una frecuencia de 10 kHz con amplitud de 3,3 V . En el canal 2 se midió la señal continua a la salida del amplificador operacional, ajustando el potenciómetro *R3Variable* del esquema de la Figura 1.14, para obtener 3,5 V cuando el ciclo de trabajo es 100 %.

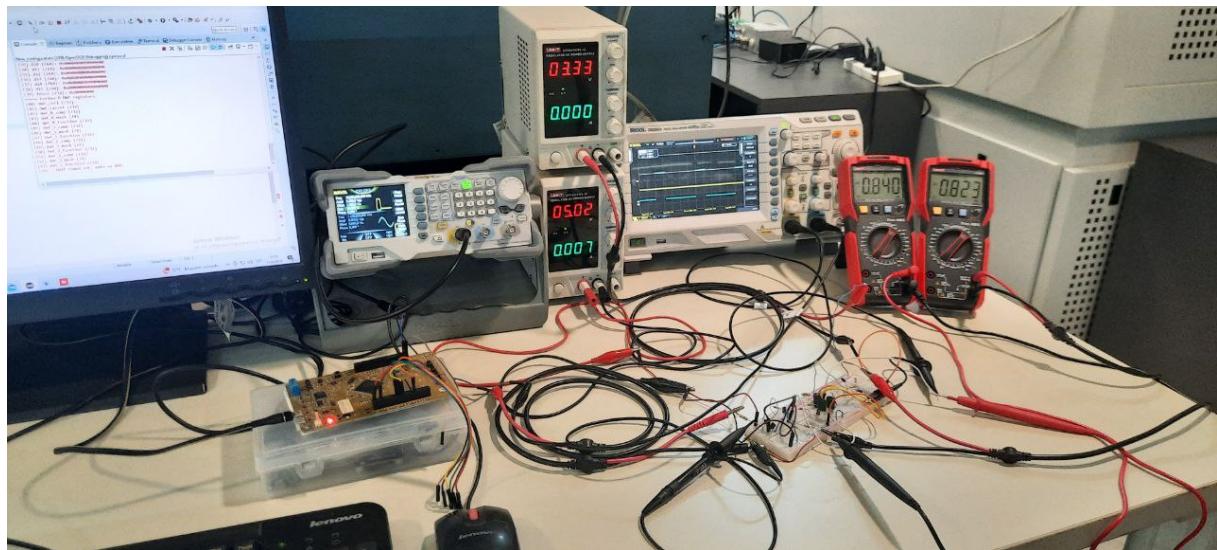


Figura 1.19: Banco de medición para caracterizar el circuito PWM.

Los resultados de las mediciones realizadas con el osciloscopio se muestran en las Figuras 1.20, 1.21, 1.22, 1.23 y 1.24, equivalente a ciclo de trabajo de 1%, 25%, 50%, 75% y 100% respectivamente.



Figura 1.20: Medición de la señal PWM (rojo) con un ciclo de trabajo de (1 %) y la señal continua a la salida del amplificador operacional (azul) iguala 53,5 mV.

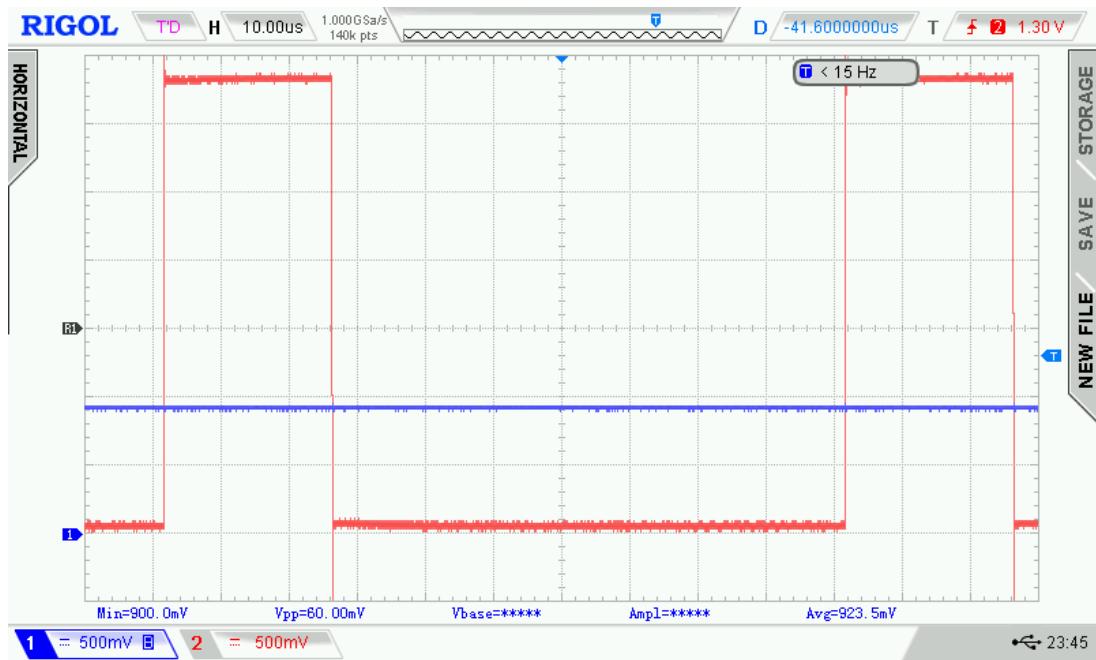


Figura 1.21: Medición de la señal PWM (rojo) con un ciclo de trabajo de (25 %) y la señal continua a la salida del amplificador operacional (azul) iguala 900 mV.

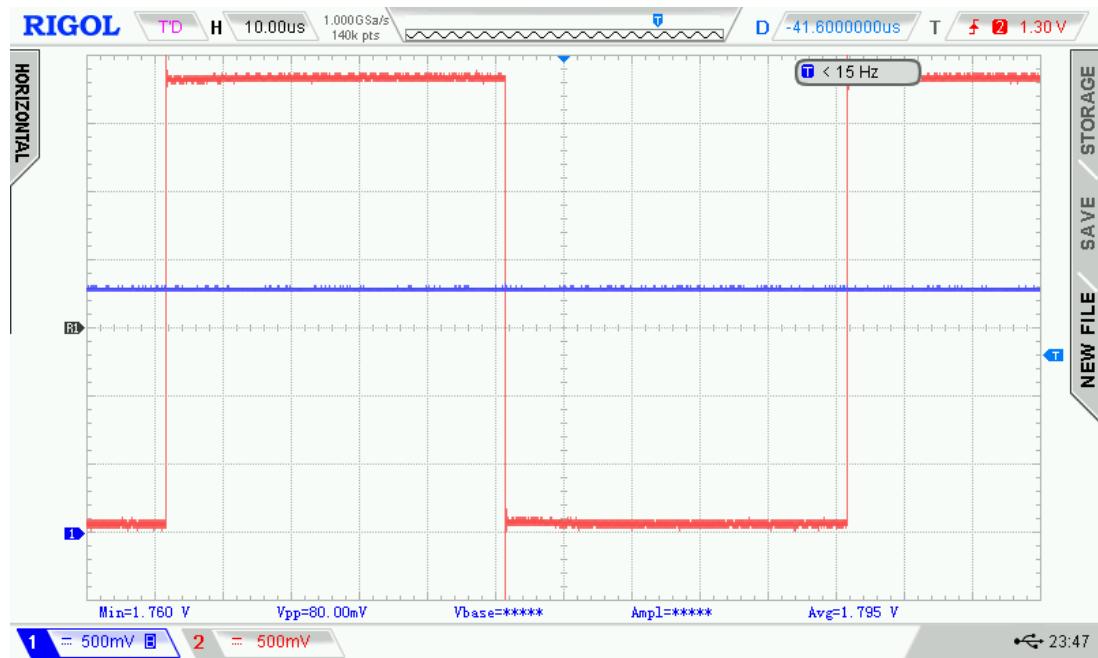


Figura 1.22: Medición de la señal PWM (rojo) con un ciclo de trabajo de (50 %) y la señal continua a la salida del amplificador operacional (azul) iguala 1,8 V.

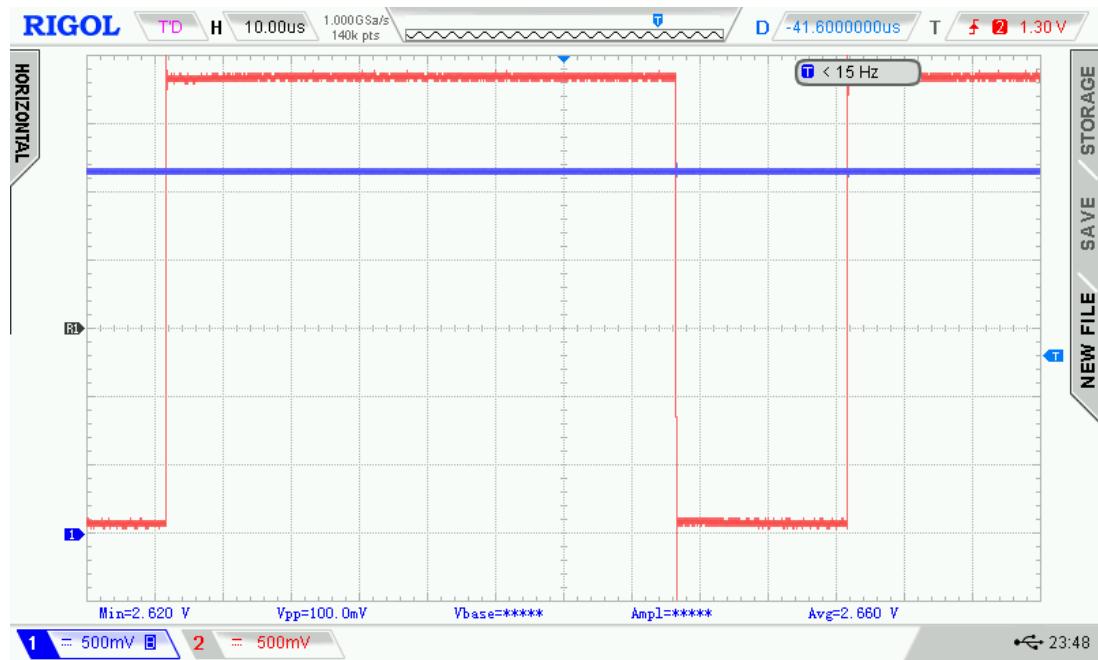


Figura 1.23: Medición de la señal PWM (rojo) con un ciclo de trabajo de (75 %) y la señal continua a la salida del amplificador operacional (azul) iguala 2,65 V.

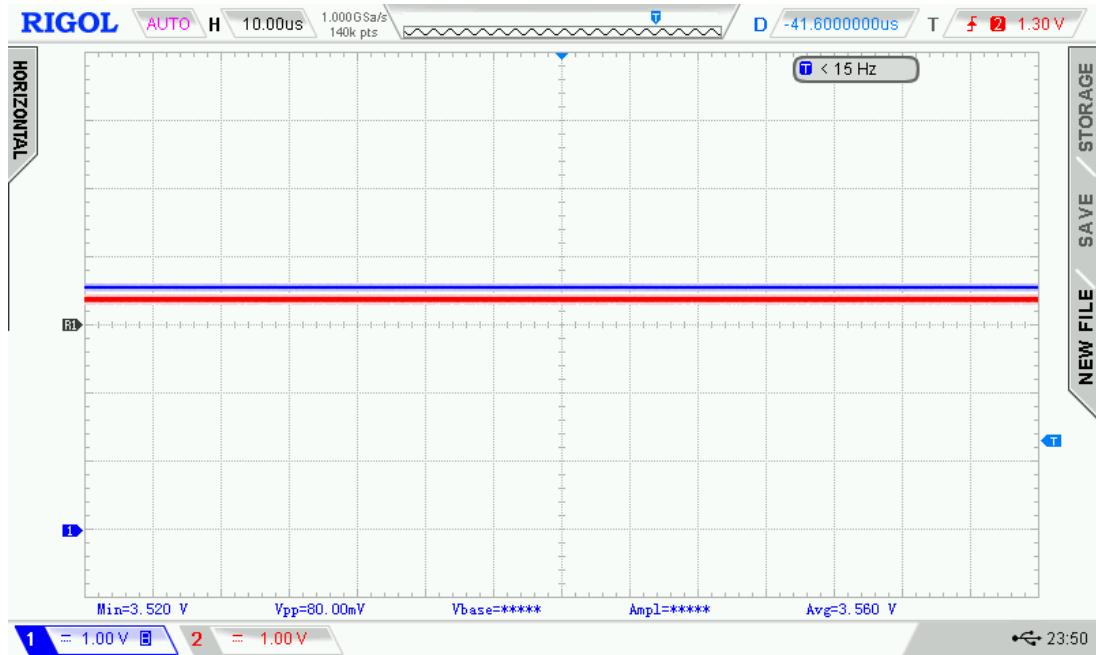


Figura 1.24: Medición de la señal PWM (rojo) con un ciclo de trabajo de (100 %) y la señal continua a la salida del amplificador operacional (azul) iguala 3,56 V.

Luego de caracterizar el circuito PWM, se dio comienzo a las pruebas en el túnel de viento. Se conectó la tensión de la salida del operacional al pin de entrada $VADC$ del variador de velocidad. A continuación, se realizó un barrido del ciclo de trabajo desde 0 hasta 255 (0 % a 100 %), midiendo con un voltímetro, UNI-T modelo UT89X, la tensión entregada al túnel y, con el anemómetro VAISALA WMT700 dentro del túnel, la velocidad en metros por segundo obtenida para cada ciclo de trabajo. En la Figura 1.25 se muestran los resultados de un barrido ascendente, y en la Figura 1.26, los resultados de un barrido descendente.

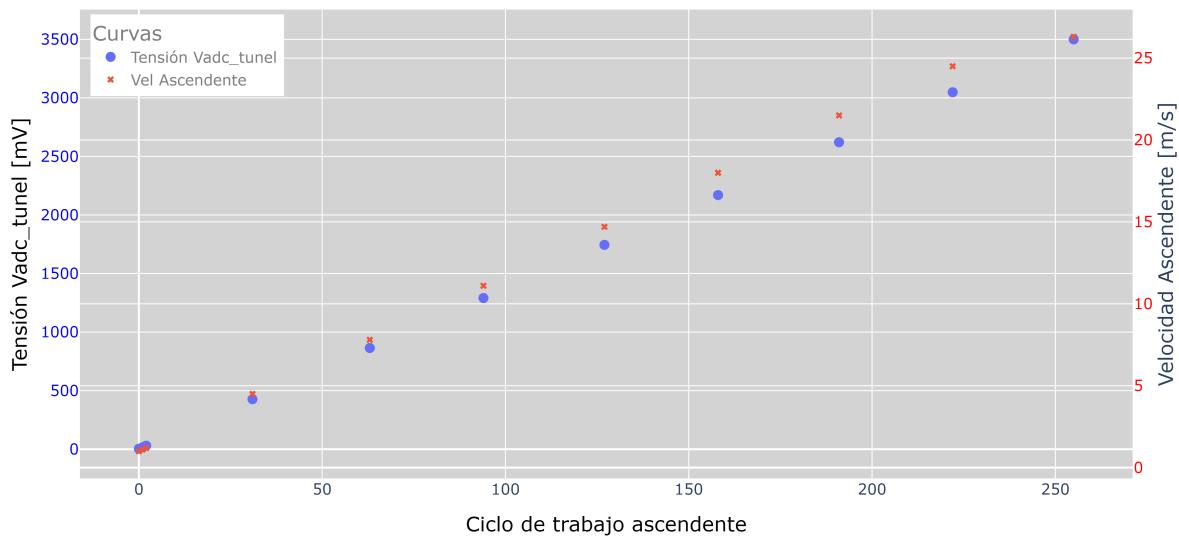


Figura 1.25: Mediciones para ciclos de trabajo en modo Ascendente, del nivel de tensión en el variador y la velocidad de viento.

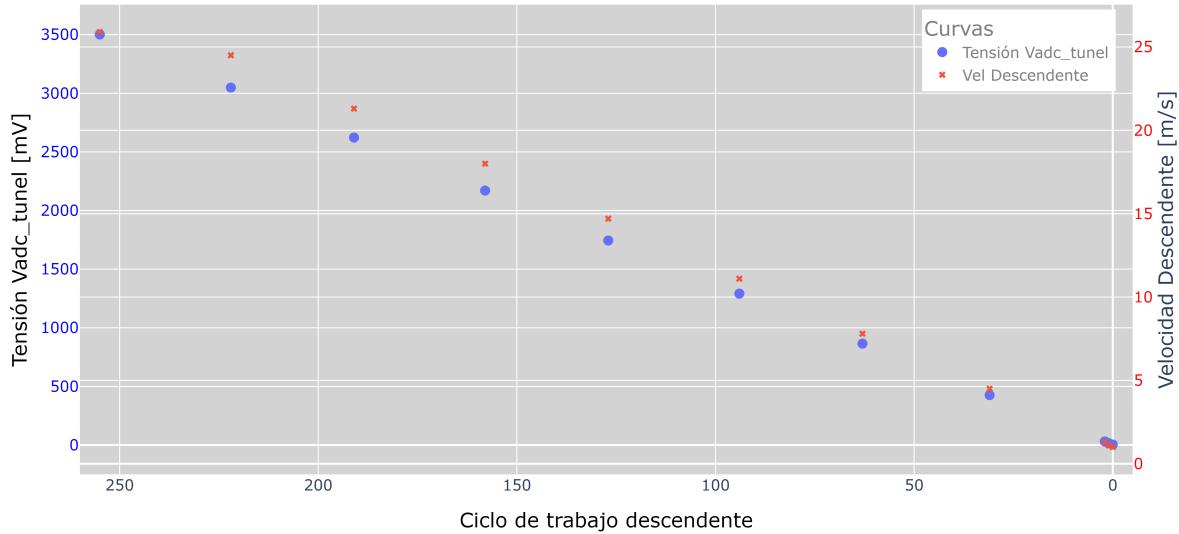


Figura 1.26: Mediciones para ciclos de trabajo en modo descendente, del nivel de tensión en el variador y la velocidad de viento.

A partir de los resultados obtenidos para el ciclo de trabajo versus la tensión en el variador y el ciclo de trabajo versus la velocidad del viento medida con el anemómetro, se determina una velocidad máxima de viento de 26 m s^{-1} para una tensión de 3,5 V. Además, dado que se tienen 255 niveles de tensión, y considerando que tanto el viento como la tensión *VADC* del variador tienen respuesta lineal en el rango de 0 a 255, se obtiene una resolución en tensión de

$\frac{3,5\text{ V}}{255} = 13,7\text{ mV}$ y una resolución en velocidad del viento de $\frac{26\text{ m s}^{-1}}{255} = 0,101\text{ m s}^{-1}$. Con estas mediciones y la caracterización del circuito PWM, éste quedó listo para ser integrado en un PCB. Este circuito será la interfaz electrónica para un controlador PID, el cual se explica en la sección 1.3.4.

1.1.5. Circuito de adquisición de tensión del variador

En esta sección se describe el diseño de un circuito que permite tomar muestras de la tensión entregada al variador del túnel de viento. El circuito funciona como un voltímetro que mide la tensión instantánea y la envía a un canal analógico-digital (ADC_CH2) de la EDU-CIAA.

Debido a que la señal de continua, para un ciclo de trabajo del 100 %, es de 3,5 V, fue necesario adaptar esta señal a 3,3 V, ya que esta es la tensión máxima de entrada para los canales ADC de la EDU-CIAA.

1.1.5.1. Simulación del circuito

En la Figura 1.27 se muestra el diseño del circuito, la señal de continua VadcTunnel se conecta al segundo operacional del chip LM358, en modo seguidor para adaptar la impedancia y no cargar a la etapa anterior, luego a la salida del operacional se conecta un divisor de tensión para adaptar la tensión a 3,3 V, además se agrega una etapa de protección con dos diodos conectados entre la fuente de 3,3 V y GND para limitar el voltaje en caso de que la tensión a la salida del operacional sea superior 3,3 V. Finalmente, se pone una resistencia de $1\text{ k}\Omega$ para limitar la corriente en VadcCIAA y para simular la resistencia de entrada del ADC, agregamos un resistor de $1,2\text{ M}\Omega$ [1].

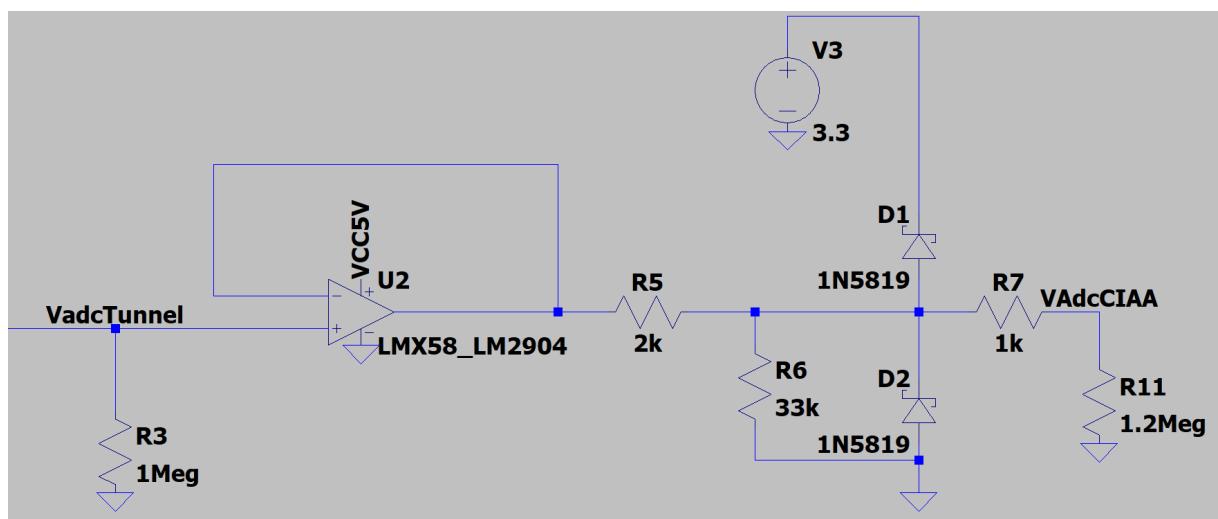


Figura 1.27: Circuito diseñado para tomar muestras de la señal que se suministra al variador de velocidad y se las envía a una canal analógico-digital de la EDU-CIAA.

Los resultados de la simulación del circuito descrito, se muestran en la Figura 1.28 para un ciclo de trabajo del 100 % y en la Figura 1.29 para un ciclo de trabajo del 55 %. En ambos casos, se puede apreciar que la señal vadccIAA (verde) está por debajo de la señal vadctunnel (azul), lo cual garantiza que no se superen los 3,3 V a la entrada del pin analógico-digital de la placa de desarrollo.

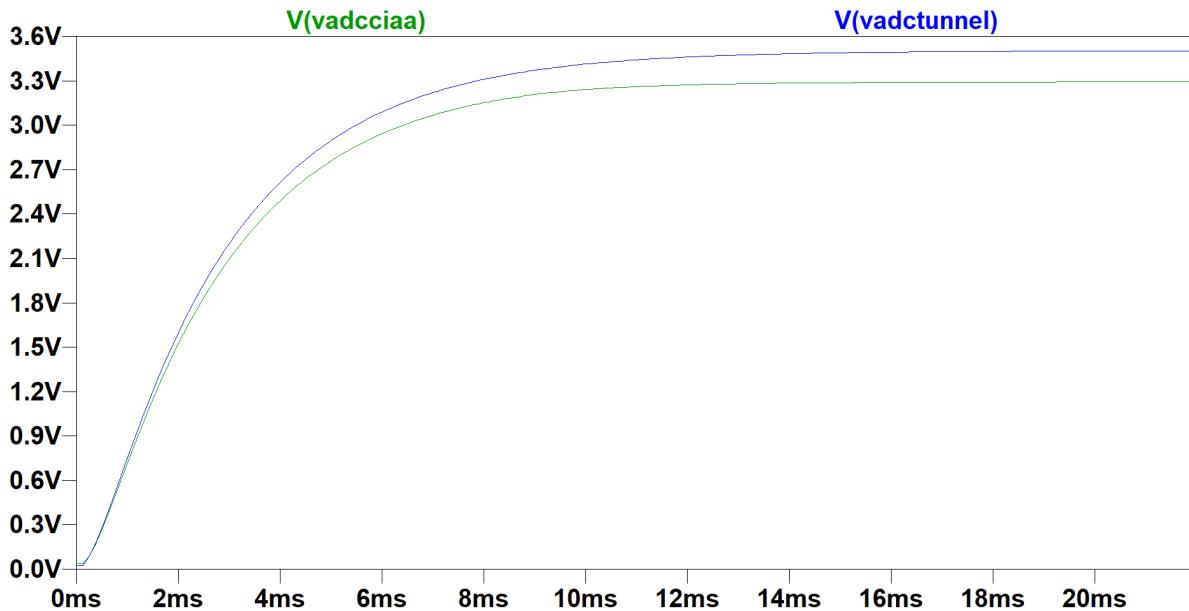


Figura 1.28: Señales simuladas a la salida del variador de velocidad y a la entrada del pin analógico-digital de la EDU-CIAA para un ciclo de trabajo del 100 %.

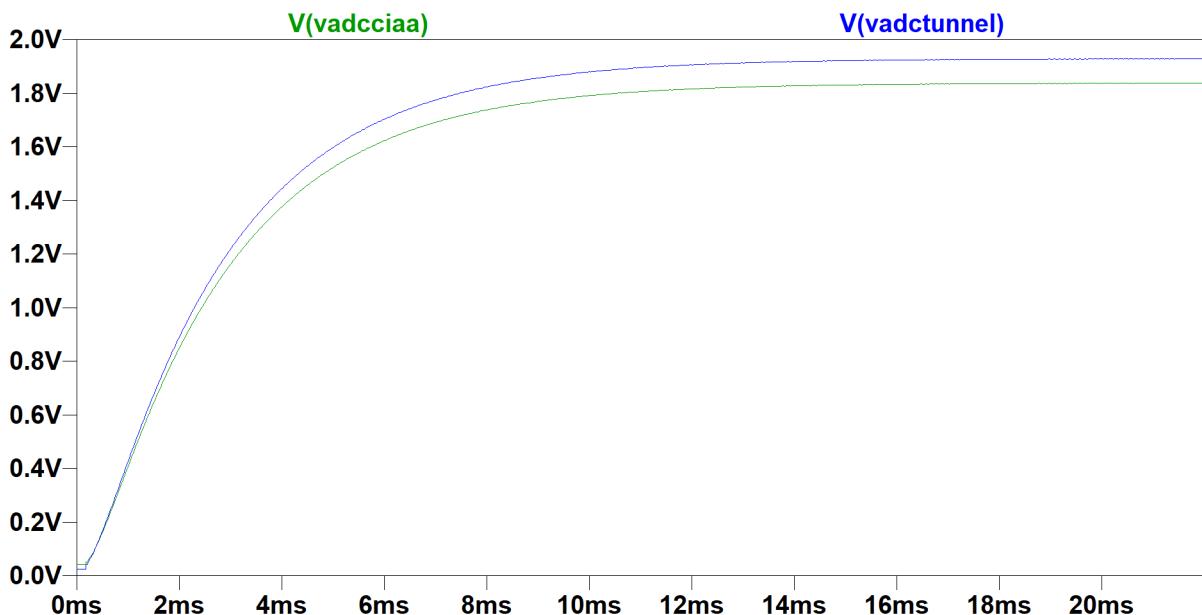


Figura 1.29: Señales simuladas a la salida del variador de velocidad y a la entrada del pin analógico-digital de la EDU-CIAA para un ciclo de trabajo del 55 %.

1.1.5.2. Implementación del circuito

Se implementó el circuito de la Figura 1.27. Para ajustar los niveles de tensión de 3,5 V a 3,3 V, se agregó en el divisor de tensión, un resistor variable con un *preset* de $R5 = 10\text{ k}\Omega$. Luego se conectó los diodos Schottky 1N5819 [6] a la fuente de 3,3 V de la EDU-CIAA y se agregó una resistencia limitadora de tensión de $R7 = 1\text{ k}\Omega$, esta señal de continua se muestrea con una resolución de $\frac{3,3\text{ V}}{1024} = 3,22\text{ mV}$ en el canal ADC_CH2 de la placa de desarrollo.

Por ultimo, para caracterizar este circuito se conectó dos voltímetros, uno en la señal de entrada del variador y otro en la señal que ingresa al ADC de la EDU-CIAA, los resultados de estas mediciones se observan en las Figura 1.30 para un barrido ascendente y 1.31 para un barrido descendente.

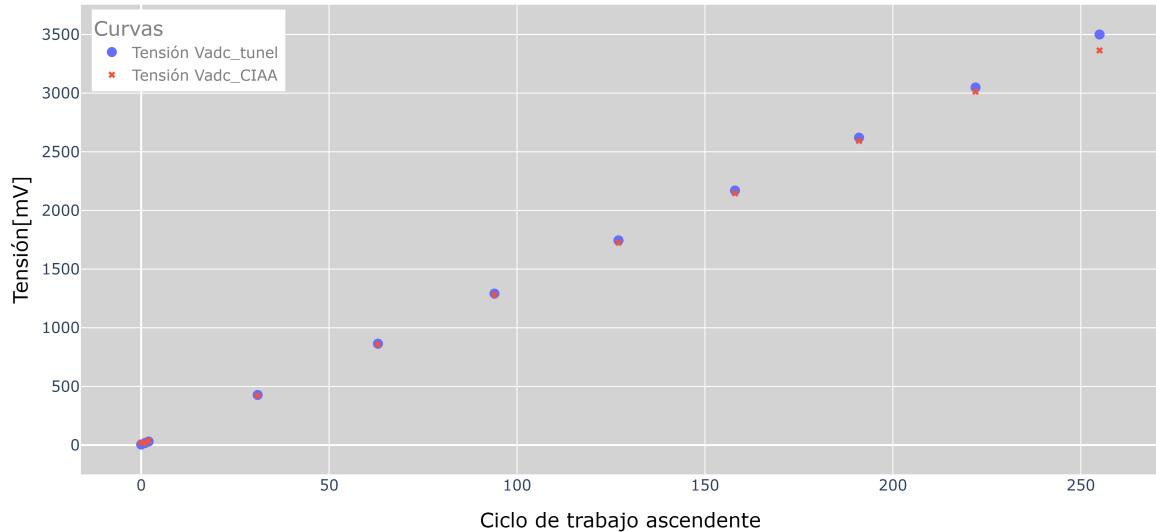


Figura 1.30: Mediciones a la entrada pin *VADC* del variador y en el pin ADC_CH2 de la EDU-CIAA, para un barrido ascendente del ciclo de trabajo.

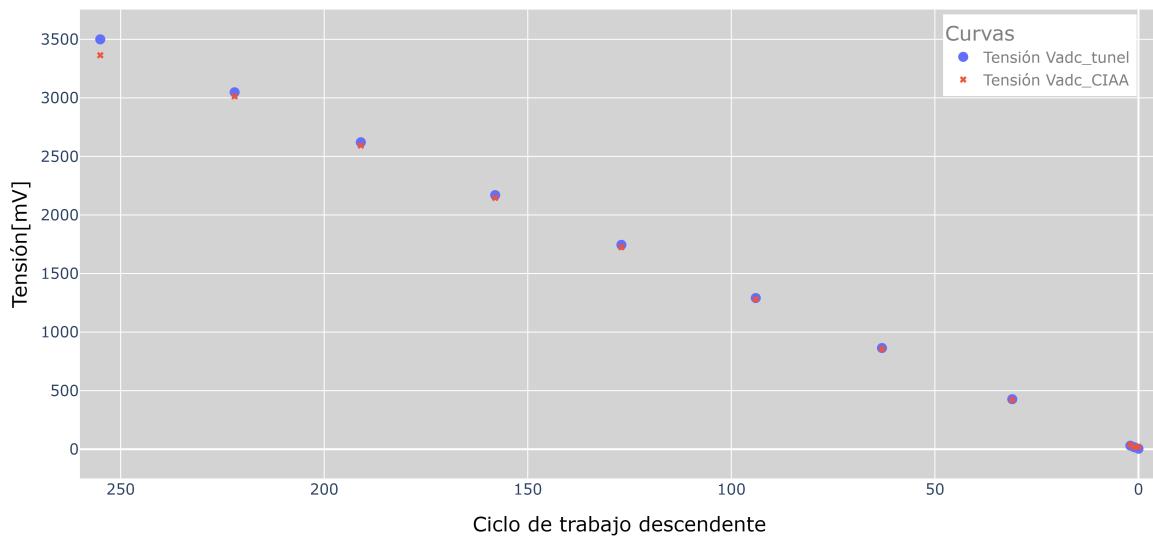


Figura 1.31: Mediciones a la entrada pin *VADC* del variador y en el pin *ADC_CH2* de la EDU-CIAA, para un barrido descendente del ciclo de trabajo.

Este circuito permite monitorear por software los niveles de tensión que estamos entregando al variador de velocidad y su correspondiente ciclo de trabajo.

1.1.6. Circuito de LEDs indicadores

Se diseñó el circuito de la Figura 1.32 para encender y apagar LEDs como indicadores de la recepción de datos de los anemómetros patrón y bajo calibración. Se utilizaron dos LEDs de colores diferentes: uno verde y otro amarillo. El LED verde se asoció al canal RS-485-1 y el LED amarillo al canal RS-485-2.

Cada circuito permite que el LED correspondiente parpadee cada vez que se recibe un dato del anemómetro, con un intervalo predeterminado por software. Si el LED no parpadea, indica que el anemómetro no está enviando datos.

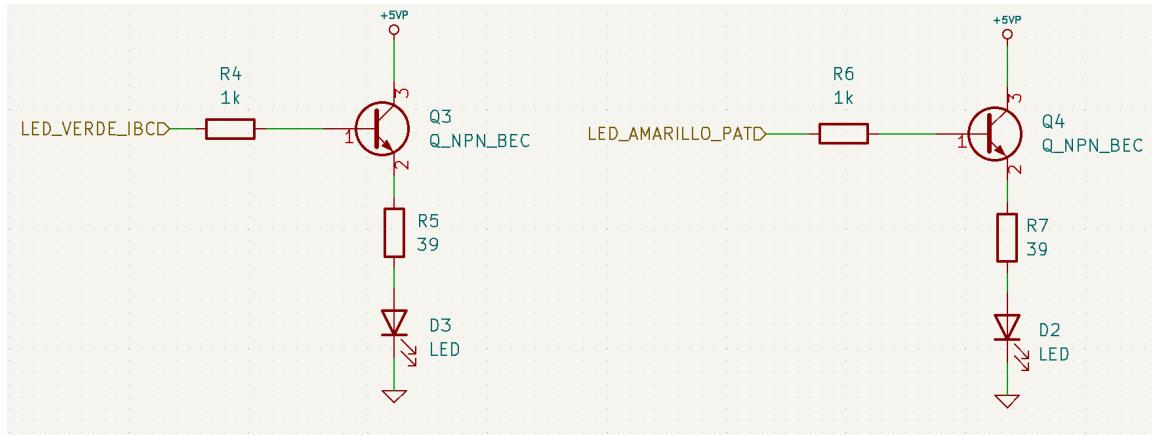


Figura 1.32: Esquemático de los circuitos que encienden y apagan dos LEDs, verde y amarillo, cada vez que se recibe un dato de los anemómetros, tanto del patrón como del que está bajo calibración.

En La Figura 1.33 se muestra un circuito similar al previamente descrito, en el que se utiliza un LED dual como indicador de la conectividad del sistema de la placa de desarrollo con el servidor Websocket descrito en la sección ???. El LED dual se enciende en rojo cuando el sistema de la placa de desarrollo está desconectado del servidor. Al establecerse la conexión a través de Ethernet, el LED cambia a verde. Este cambio de color ocurre automáticamente según el estado de la conectividad con el servidor. El LED dual tiene tres pines: dos ánodos y un cátodo común conectado a masa. Los ánodos están conectados a circuitos que controlan el encendido y apagado del LED según las señales de los pines digitales de la placa de desarrollo.

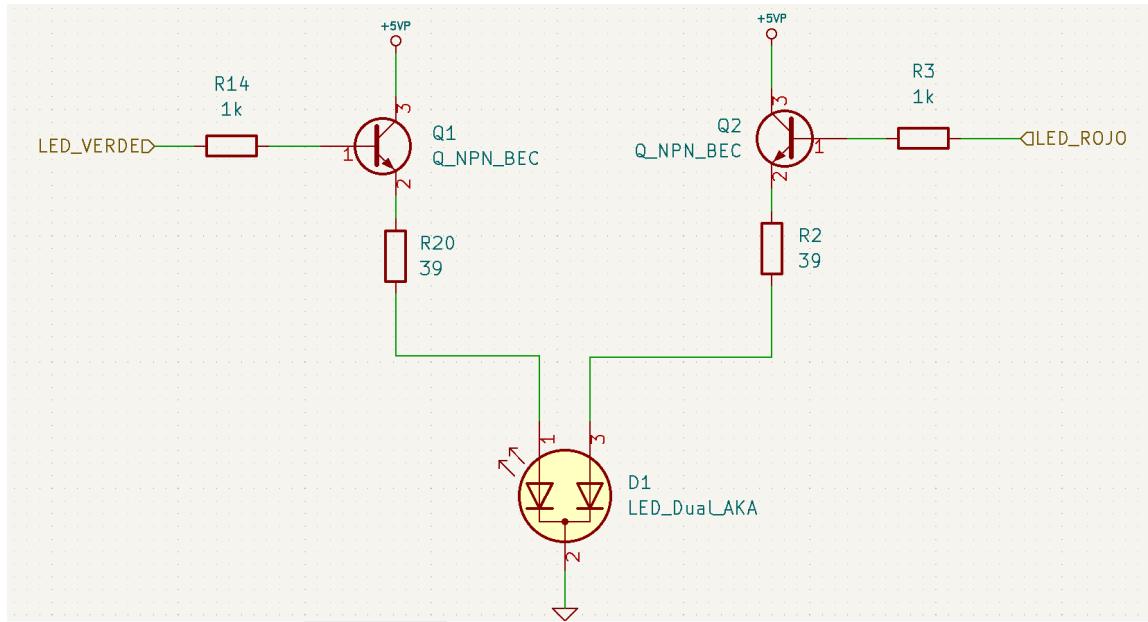


Figura 1.33: Esquemático del circuito de un LED dual que se enciende en rojo cuando el sistema no está conectado al servidor WebSocket y en verde cuando hay una conexión estable con el servidor.

1.2. Diseño y construcción del PCB

En las secciones anteriores se ha descrito el diseño y funcionamiento electrónico de los distintos módulos y circuitos que forman parte del datalogger. Para integrar todo en un solo lugar, se diseñó un *shield* que se conecta sobre la EDU-CIAA. Para ello, se trabajó con la lista de templates que ofrece el proyecto CIAA [4]. Este template tiene un diseño acorde con las dimensiones de la placa EDU-CIAA.

Para el desarrollo del PCB, se utilizó el software KiCad. Primero se cargó el template de la Figura 1.34 que incluye una hoja jerárquica con los pines disponibles en los conectores P1 y P2 y luego se agregaron etiquetas jerárquicas por cada pin utilizado en este desarrollo.

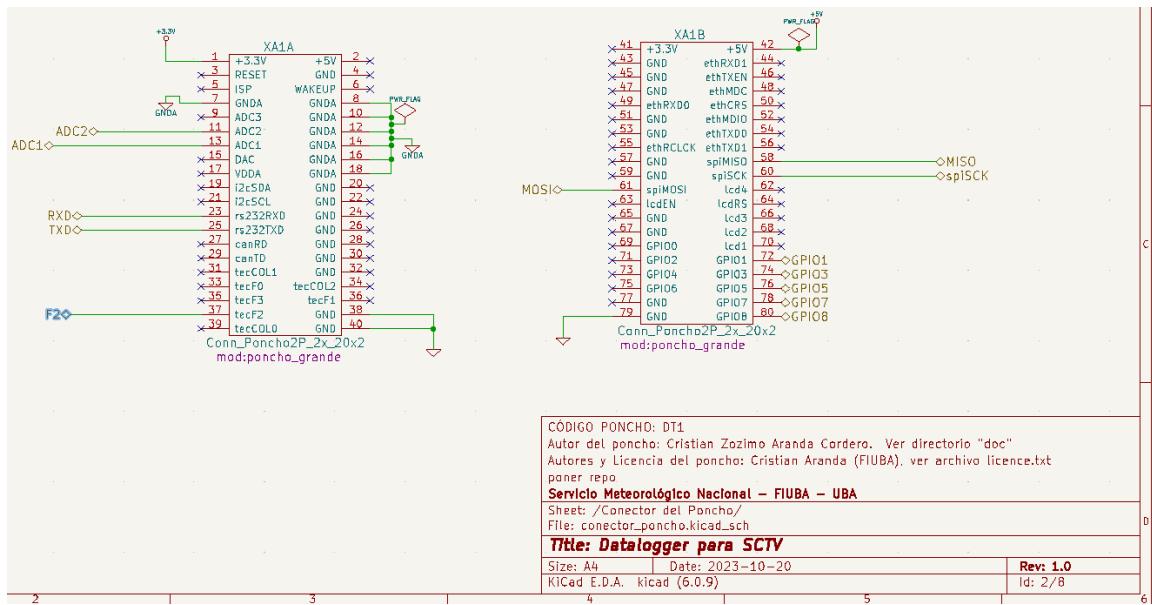


Figura 1.34: Template disponible para la realización de ponchos para la EDU-CIAA.

Por cada módulo, se agregó una hoja jerárquica que contiene el circuito esquemático como se indica en la Figura 1.35. En total, se dividió en seis hojas jerárquicas: una para el módulo RS485, otra para el módulo Ethernet W5100, otra para el circuito PWM, otra para la fuente de alimentación (power supply), otra para los LEDs y una más para el circuito que mide la tensión en el variador del túnel. Además, se diseñaron los símbolos para los módulos W5100, RS485-TTL y el convertidor DC-DC usando el editor de símbolos de KiCad, ya que estos no vienen precargados en el software. También se diseñaron sus respectivos footprints con el editor de huellas, tomando las dimensiones y la distribución de pines de cada módulo para que se puedan agregar correctamente al PCB.



Figura 1.35: Diseño del esquemático que integra los módulos desarrollados.

En la Figura 1.36 se muestra el diseño del PCB de doble capa, con la capa inferior (*Bottom*) en azul y la capa superior (*Top*) en rojo. Las dimensiones de los bordes recortados y la tira de 20 pines en la parte superior e inferior forman parte del template; el resto del circuito se diseñó en función de las necesidades de cada módulo. Las pistas tienen un ancho de entre 30 y 40 mils. Se agregaron vías de 2 mm de diámetro con un agujero de 1 mm para conectar la capa superior con la capa inferior. Además, los pads tienen un tamaño de 2 mm.

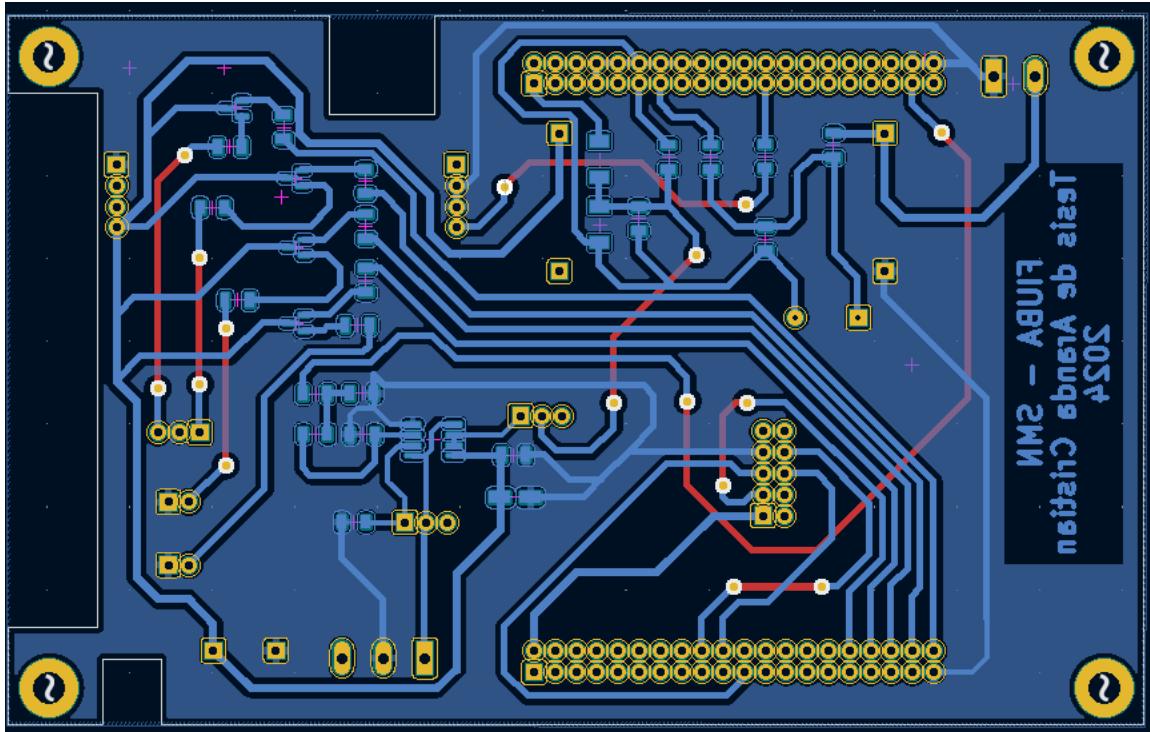


Figura 1.36: Diseño PCB del *shield* que contiene todos los módulos descritos en las secciones anteriores.

En la figura 1.37 se muestra la vista 3D de capa *Bottom* donde se diseñó los circuitos PWM, el adquisidor de tensión de la fuente y el adquisidor de tensión del variador, utilizando componentes SMD. Se emplearon diodos, capacitores y resistores con tamaño de paquete 1206 y para los transistores NPN MMBT3904 y el operacional LM358 se cargaron sus footprints SMD de la biblioteca de KiCad.

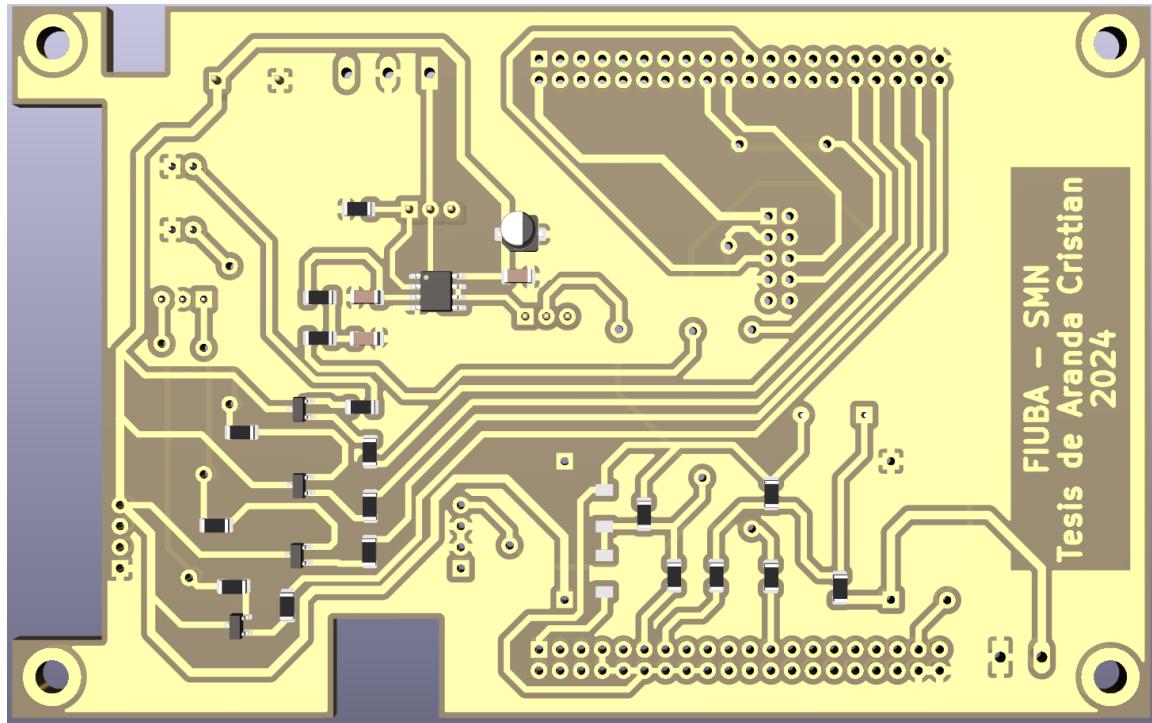


Figura 1.37: Vista en 3D de la capa BOTTOM.

Por otro lado, en la capa superior (TOP) se han añadido los tres LEDs con montaje THT y se han dispuesto dos tiras de pines 3x1 para la conexión física de los dos potenciómetros variables (presets). Además, debido a la falta de disponibilidad del diodo Zener SMD de 3,3 V para proteger el ADC-CH1 de la placa EDU-CIAA, se adquirió el mismo componente pero con montaje THT. También se han incorporado los footprints de los módulos previamente diseñados del módulo Ethernet, módulo RS485-UART (que incluye su propia bornera para la conexión del anemómetro bajo calibración), y módulo DC-DC step-down. Además, se han agregado una bornera de 2x1 para alimentar el circuito con 12 V, y otra bornera 3x1 para la conexión con el variador del túnel de viento.

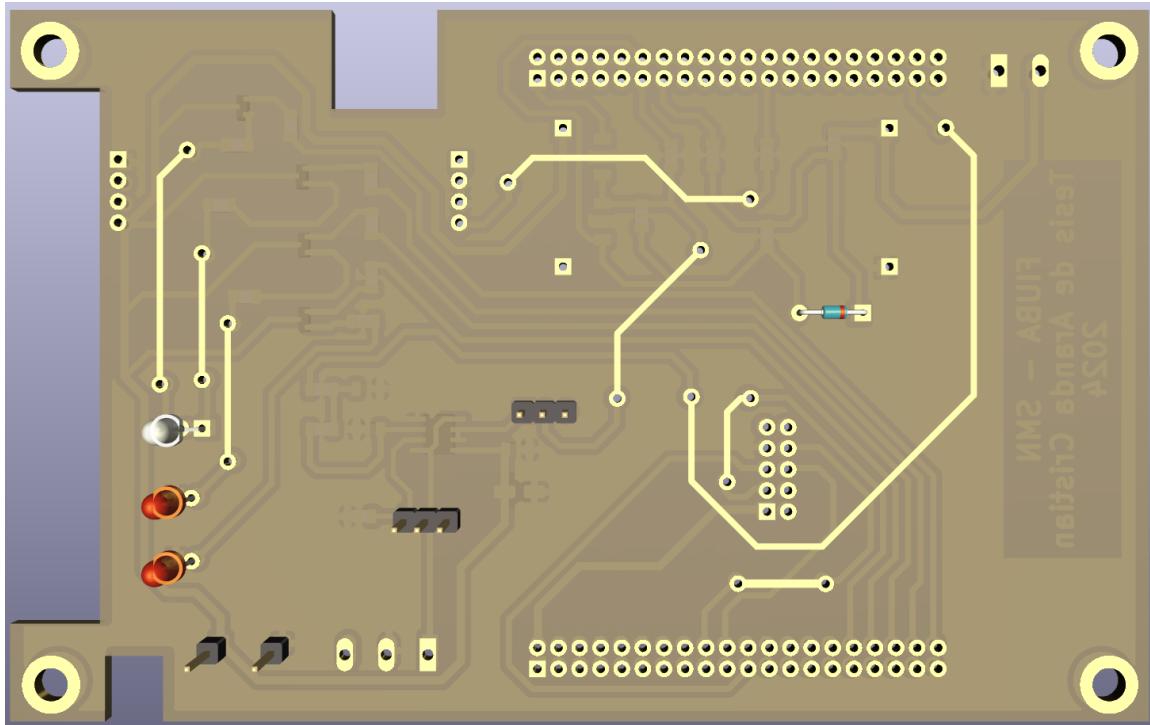


Figura 1.38: Vista en 3D de la capa TOP.

Para la construcción del *shield*, se seleccionó una placa simple faz por su disponibilidad en el Servicio Meteorológico Nacional (SMN). Debido a restricciones de tiempo, se optó por ensamblar la placa en el laboratorio del SMN, siguiendo estos pasos:

Inicialmente, se empleó una insoladora de tubos UV y papel fotosensible para transferir el diseño a la placa. Los negativos del diseño se imprimieron en papel transparente.

Después, se utilizó ácido perclórico para retirar el cobre remanente. Finalizado este procedimiento, se cortó la placa a la medida deseada y se perforaron los *pads* con una herramienta *Dremel*. Se aplicó una capa de máscara antisoldante para facilitar la soldadura de los componentes y proteger la placa. Dado que la placa era de tipo simple faz, se añadieron manualmente las pistas de la capa superior (*Top*) usando pequeños cables. El proceso concluyó con la soldadura de todos los módulos y componentes.

En la Figura 1.39 se ilustra el resultado final del montaje. Se conecta un cable USB al puerto **USB-DEBUG** para programar y depurar la placa. El sensor de viento patrón se conectará a la bornera azul del puerto **RS485 EDU-CIAA**, mientras que el anemómetro bajo calibración se conectará a la bornera verde del módulo **RS485 MAX485**. En la parte superior de la figura, se observa una bornera azul para conectar los tres cables provenientes del variador de velocidad del túnel: **VCC**, **VADC** y **GND**. En el lado izquierdo, se ha conectado un cable UTP a la red local del SMN a través de **Ethernet**, y en la bornera verde situada en la parte inferior izquierda, se ha conectado la alimentación de entrada de $V_{in} = 12\text{ V}$. Para alimentar la placa sin USB,

se dejan dos pines **Vcc_5V** y **GND** que tienen 5 V provenientes del módulo DC-DC y se conectan a la **fuente externa de 5V** con dos cables. Adicionalmente, se encuentran disponibles los dos *presets* para recalibrar tanto el amplificador operacional como el circuito encargado del muestreo de esta señal. Finalmente, se observan los indicadores LED: el LED verde parpadea al recibir datos por el puerto RS485 de la EDU-CIAA, el LED amarillo parpadea al recibir datos por el puerto RS485 del módulo, y el LED blanco cambia a verde cuando la placa está conectada al servidor, o a rojo si no lo está.

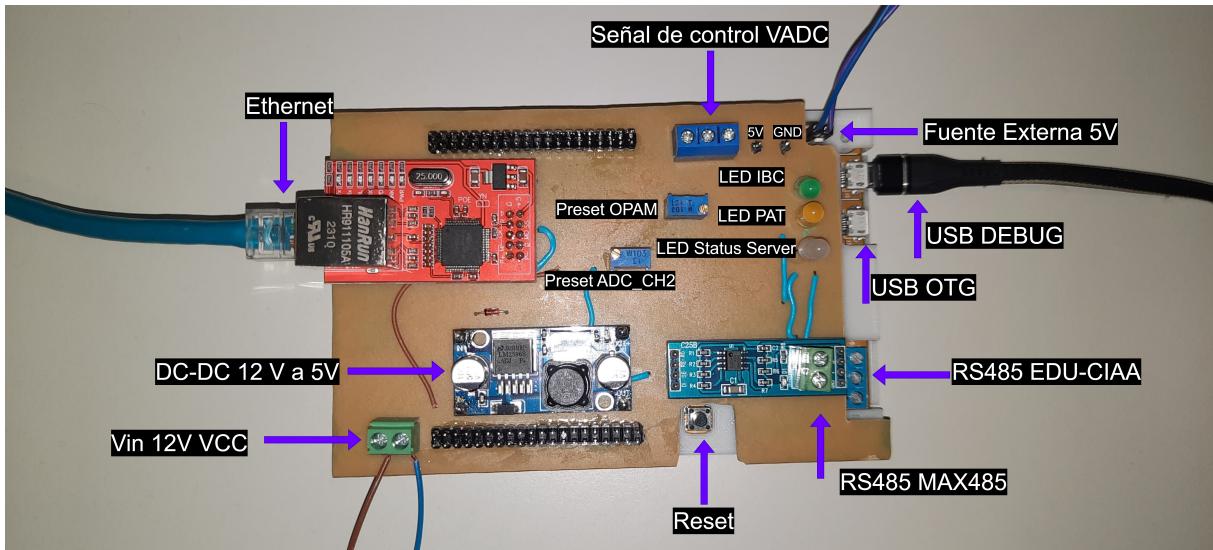


Figura 1.39: Vista superior de la placa *shield* conectada a la EDU-CIAA

1.3. Desarrollo del firmware

El proyecto CIAA brinda un framework para programar las placas de desarrollo. La última versión, el **firmware_v3** [7], es un proyecto basado en makefile para desarrollar firmware en C/C++. En particular se utilizó la biblioteca sAPI, que ofrece ejemplos para programar los periféricos del microcontrolador. Además, el proyecto incluye un lanzador de aplicaciones Figura 1.40, que trae integrado el IDE GNU MCU Eclipse como entorno de programación, OpenOCD para programación y depuración, drivers para conectar la EDU-CIAA a la PC, entre otras herramientas.



Figura 1.40: Lanzador de aplicaciones integradas para el desarrollo de firmware para la EDU-CIAA.

Se clonó el repositorio que contiene el **firmware_v3** de GitHub en el IDE y se abrió un nuevo proyecto llamado **Datalogger**, como se indica en la Figura 1.41.

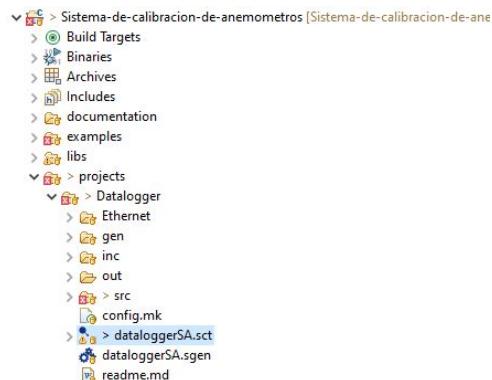


Figura 1.41: Estructura de carpetas, del proyecto Datalogger

En particular, la lógica del programa se implementó mediante el uso de máquinas de estado (*statecharts*) de Harel, empleando el software Itemis Yakindu [10]. Los *statecharts* de Harel amplían los diagramas de estados tradicionales incorporando conceptos de modularidad, jerarquía y estructura organizacional, similar a las maquinas de estado UML [11]. Estos diagramas se caracterizan por el uso de estados compuestos y subdiagramas que preservan la claridad y la organización, y facilitan la ejecución concurrente de múltiples submáquinas de estado a través de regiones ortogonales. Además, los eventos que inician con (*e_*) habilitan la comunicación por difusión, mientras que las transiciones condicionales que se denotan entre llaves ([*condición de guarda*]), los estados históricos, la lógica temporal y las acciones de entrada/salida potencian la funcionalidad de los *statecharts*, optimizando la modelización de sistemas complejos de manera

eficiente. La Tabla 1.2 presenta un análisis comparativo entre las máquinas de estado simples, como las de Mealy y Moore, y las máquinas de estado más avanzadas de UML y Harel.

Descripción	Mealy	Moore	Harel	UML
Estados y transiciones	✓	✓	✓	✓
Las transiciones producen salida	✓	✗	✓	✓
Los estados producen salida	✗	✓	✓	✓
Profundidad (jerarquías, estados compuestos)	✗	✗	✓	✓
Ortogonalidad (submáquinas de estado paralelas)	✗	✗	✓	✓
Comunicación por difusión (eventos)	✗	✗	✓	✓
Historial, acciones, retrasos, tiempos de espera, condicione de guarda	✗	✗	✓	✓

Tabla 1.2: Diferencias entre los tipos de máquinas de estados.

Se crearon doce regiones donde cada una contiene una maquina de estados. A partir de una maquina principal (*main*) el resto de estas trabajan de forma concurrente (de forma ortogonal). En la Figura 1.42 se muestran las regiones y se comunican entre sí mediante eventos y señales internas. A cada región se le asigna una prioridad en función de su posición; las que están más arriba tienen mayor prioridad de ejecución respecto a las de más abajo.

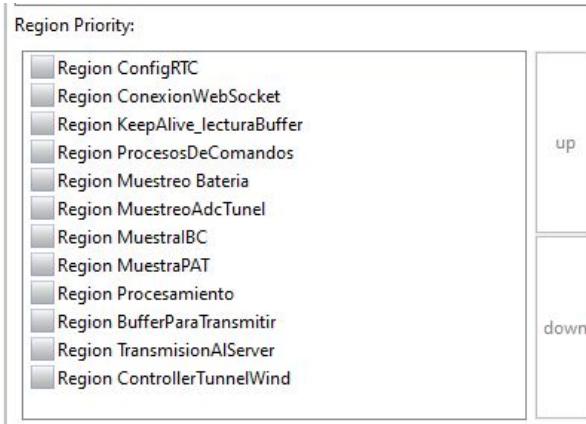


Figura 1.42: Lista de regiones con su respectiva prioridad.

Se utilizó una licencia educativa, gratuita de Yakindu, integrándola con el IDE de Eclipse para editar y generar el código a partir de los *statecharts*. En particular, se configuró el proyecto en Yakindu C/C++ para poder agregar código en C a las máquinas de estado, incluyendo la importación de archivos .h, declaración de variables, punteros globales y constantes. Para mas detalle del código se puede acceder al repositorio https://github.com/InstrumentalSMN/Sistema-de-calibracion-de-anemometros/tree/ethernet_comunication_websocket.

La máquina de estado principal (*main*) se muestra en la figura 1.43 e inicializa y configura

la placa, el SPI, las UARTs, un contador de ticks y el resto de las máquinas de estado. Luego, pasa a un estado inactivo, esperando alguna interrupción. Cuando ocurre una interrupción, se verifica el estado de un *flag* del temporizador principal del sistema. Si el *flag* está activo, se actualizan todos los *ticks* de los temporizadores. Posteriormente, el sistema escanea todos los *ticks* de los temporizadores para verificar eventos pendientes. Si hay eventos pendientes, estos se levantan en su respectiva máquina de estados. Finalmente, la máquina de estados ejecuta un ciclo para manejar los eventos levantados (*raiseEvent*). Luego, el sistema regresa al estado inactivo, esperando la próxima interrupción.

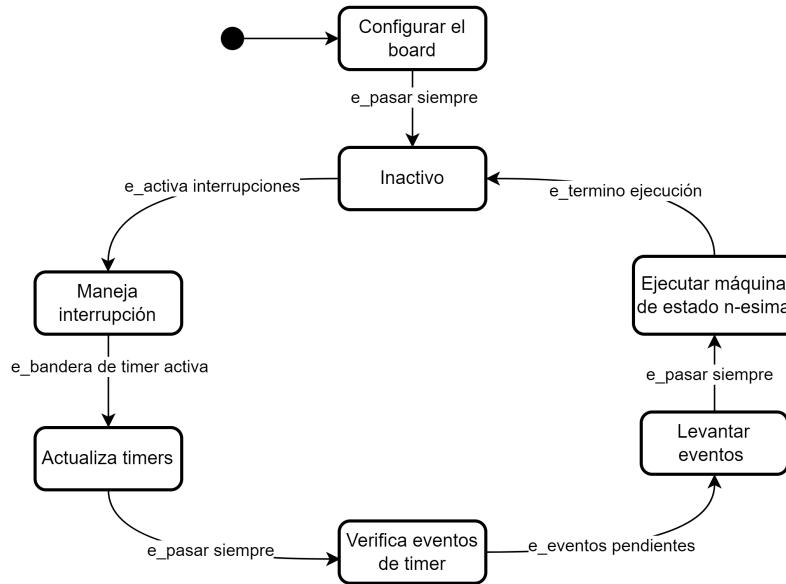


Figura 1.43: Maquina de estados principal, que gestiona la ejecución del resto de las maquinas de estado.

1.3.1. Configuración y conexión con los servidores

Como se detalló en la sección 1.1.3, se utilizó el módulo W5100 para transferir datos desde el datalogger hacia los servidores y recibir datos de los mismos. El módulo cuenta con cuatro socket independientes que permiten configurar un sistema embebido como servidor o cliente.

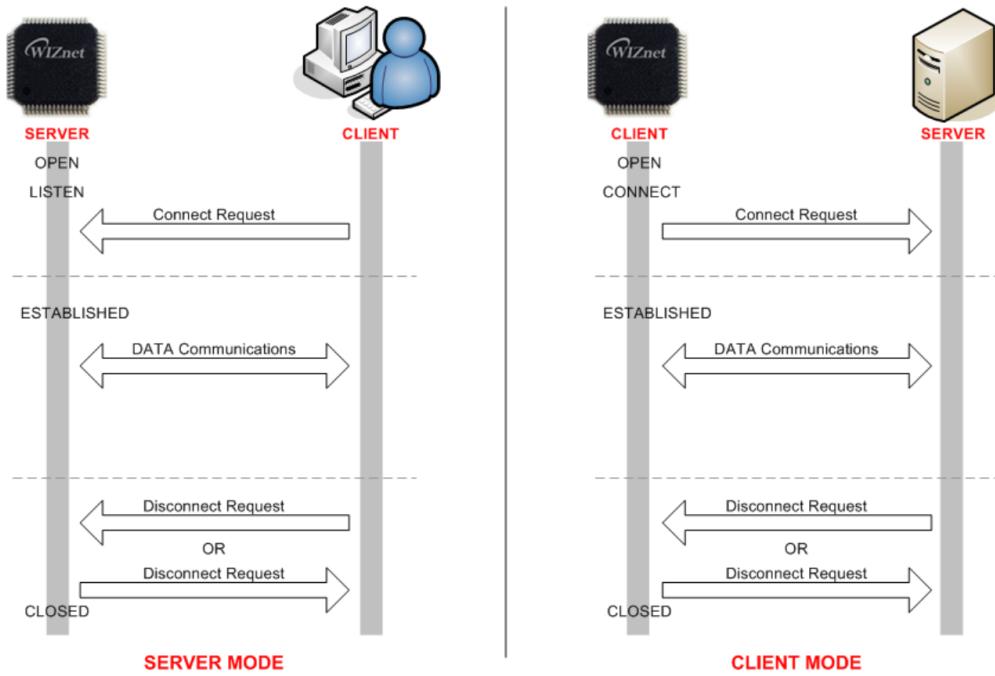


Figura 1.44: Modos de configuración para el módulo W5100.

El datalogger se comunica con los servidores en modo cliente (apertura activa), enviando una solicitud de conexión a un servidor, estableciendo la comunicación e intercambiando datos. Luego, realiza una solicitud de desconexión y se cierra el enlace. Utiliza, en la capa de transporte, el protocolo TCP y, en la capa de red, el protocolo IP. En la capa de aplicación, se utilizó el protocolo WebSocket para intercambio de datos y el protocolo NTP para configurar el RTC del datalogger. Dado que estos protocolos no están incluidos en la biblioteca del chip W5100, se implementó la conexión, el intercambio de datos y la desconexión de los servidores mediante sockets.

1.3.1.1. Servidor NTP

Para la configuración del RTC del datalogger, se estableció una conexión con los servidores NTP del Instituto Nacional de Estándares y Tecnología (*NIST*). Se seleccionó una de las múltiples IPs disponibles, específicamente la dirección 128.138.141.172 a través del puerto 13. El servidor retorna una cadena de caracteres, 59992 23-02-17 17:47:01 00 0 0 831.5 UTC(*NIST*)*, que se procesa y se guarda en una estructura que contiene las variables de hora y fecha del RTC. Para gestionar y validar esta comunicación con el servidor NTP, se configuró una máquina de estados específica dentro del sistema ilustrada en la Figura 1.45.

La máquina comienza en el estado **Inicio** y, luego de un tiempo predefinido, transiciona al estado **Configurar socket**, el cual se encarga de configurar los parámetros de red del cliente:

gateway, dirección MAC, máscara de subred e IP locales. Posteriormente, estos parámetros se cargan en uno de los cuatro sockets del módulo W5100, se configura el protocolo de transporte (TCP en este caso), se asigna un puerto y se abre la conexión para establecer comunicación con el servidor. Si la apertura y configuración son correctas, la máquina de estados transiciona al estado **Configurar servidor NTP**, donde se configura la dirección IP y el puerto, y con estos datos se solicita una conexión al servidor NTP. En caso de que la conexión esté lista, el servidor devuelve la fecha y la hora actual en formato UTC. La máquina de estados transiciona al estado **Establecer fecha y hora**, donde se procesa la información recibida y se actualiza el RTC del datalogger. Finalmente, se cierra la conexión con el servidor NTP, pasando al estado **Inactividad**. Cuando se realiza esta última transición, se eleva un evento a través de la señal `s_Iniciar configuración`, la cual será un disparador para transicionar entre estados de otra máquina de estados que gestiona la conexión con el servidor *WebSocket*. En caso de fallo en cualquiera de los estados previos, se transiciona al estado **Establecer Fecha y hora por defecto**, donde se establece por valores predeterminados.

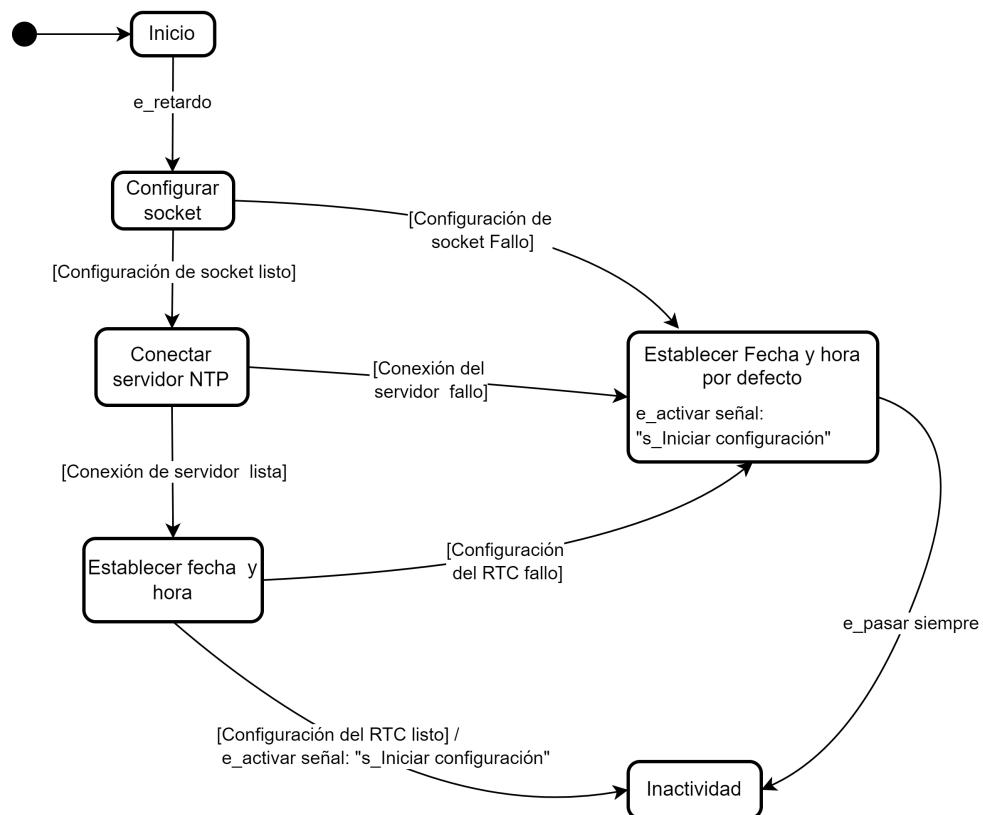


Figura 1.45: Máquina de estados para configurar el RTC del datalogger a través de un servidor NTP.

1.3.1.2. Servidor WebSocket

El protocolo WebSocket es una tecnología de comunicación que permite interacciones bidireccionales (*full-duplex*) entre un cliente y un servidor a través de un solo canal de conexión TCP. Es especialmente útil para aplicaciones web que requieren comunicación en tiempo real, como juegos en línea, chat en vivo y servicios de trading. Para establecer una conexión WebSocket, el cliente envía una solicitud de *handshake* (apretón de manos) al servidor. Esta solicitud sigue el protocolo HTTP con un *upgrade* para solicitar el cambio de protocolo a WebSocket. Aquí se muestra un ejemplo de cómo se ve una solicitud de *handshake* de WebSocket:

```
GET /mychat HTTP/1.1
Host: 10.10.13.153
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: x3JJHMBDL1EzLkh9GBhXDw==
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
Origin: http://example.com
```

El servidor, al recibir esta solicitud, responde con una respuesta de *handshake* si acepta la conexión:

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=
Sec-WebSocket-Protocol: chat
```

La clave **Sec-WebSocket-Accept** es generada por el servidor a partir de la clave **Sec-WebSocket-Key** enviada por el cliente, lo que confirma que el servidor entiende y acepta la solicitud de WebSocket. Una vez establecido el *handshake*, el cliente y el servidor pueden enviar datos en forma de *frames* de WebSocket, que pueden ser datos de texto o binarios.

A continuación se pasa a explicar cómo se implementó la comunicación. Para ello se declaran las variables de estado que se muestran en el Código 1.1 y permiten la transición de estados de la región **ConexionWebSocket** que se muestra en la figura 1.46

```
1 // Variables para la conexión al servidor WebSocket
2 ar socketConnect: uint8_t = FALSE
3 var isConnectWebServer: uint8_t=FALSE
```

```

4 | var buffertempLleno: uint8_t = TRUE
5 | var txKeep: uint8_t = FALSE

```

Código 1.1: Declaración de variables booleanas para validar la conexión con el servidor WebSocket.

La máquina de estados inicia en el estado **IDLE**. La transición al estado **CONFIG_SOCKET** se desencadena por el evento de la señal **siStartMeasure**, generado tras la correcta configuración del tiempo por la máquina de estados del servidor NTP. Esto asegura que no se inicie la medición hasta que la hora esté precisamente sincronizada. En el estado **CONFIG_SOCKET**, se invoca la función **opConfigSocket()**, responsable de configurar los parámetros de red del módulo W5100, incluyendo el gateway, la máscara de subred, la dirección MAC y una dirección IP. También se establece el protocolo TCP, asignando un número de puerto al cliente. Tras abrir el socket, la máquina de estados avanza al estado **INI_WEB_SOCKET** sin requerir validación adicional.

En el estado **INI_WEB_SOCKET**, se ejecuta la función **opInitWebSocket()**, que prepara los datos necesarios para la conexión con el servidor WebSocket, como la dirección IP del servidor y el puerto. Esta configuración permite actualizar la variable **socketConnect** a **True**, habilitando el avance al estado **CONNECT_WEBSOCKET**. Aquí, la función **opConnectToWebSocket()** gestiona el envío de la solicitud de *handshake* al servidor y procesa su respuesta. Si la conexión es exitosa, la variable **isConnectWebServer** se establece en **True**, indicando que la conexión está lista para el intercambio de datos en modo full-duplex.

Si se presenta un fallo en cualquiera de los estados previos, la máquina de estados regresa al estado **CONFIG_SOCKET** para intentar una nueva conexión con el servidor.

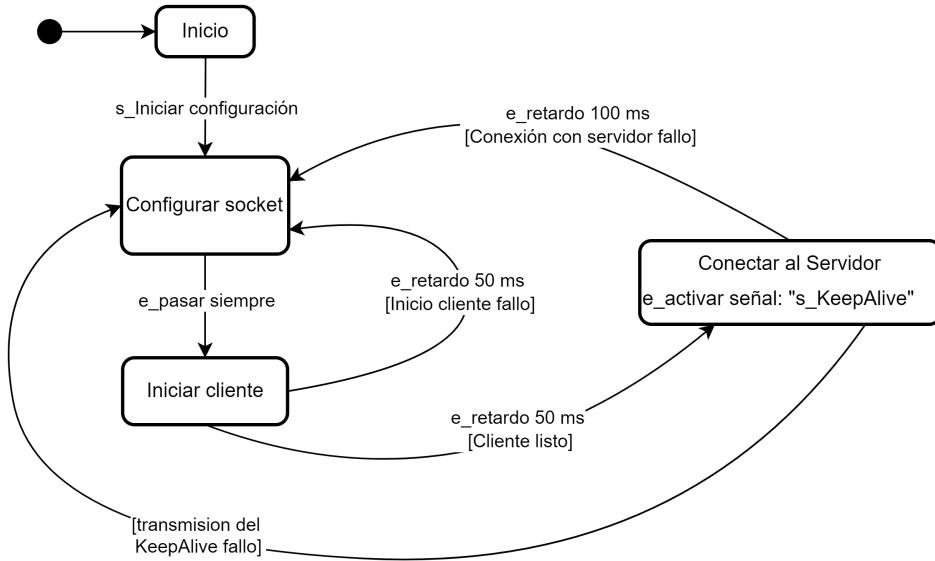


Figura 1.46: Máquina de estados para establecer una conexión bidireccional con un servidor WebSocket.

Para prevenir la desconexión automática del protocolo WebSocket en ausencia de intercambio de datos, se implementó una región denominada `KeepAlive_lecturaBuffer`. Esta región tiene un doble propósito: primero, mantener activa la comunicación con el servidor incluso cuando no hay tráfico de datos; y segundo, realizar una consulta periódica (*polling*) con un retardo no bloqueante para verificar si se ha recibido algún mensaje del servidor.

El estado final de la región `ConexionWebSocket` de la Figura 1.46, al completar su secuencia de operaciones, genera un evento con la señal `siKeepAlive`. Esta señal, en conjunción con la condición de que `isConnectWebServer` esté establecida en `True`, facilita la transición desde el estado `IDLE` hacia el estado `TX_MESSAGE`. En este último estado, se pueden transmitir mensajes al servidor, manteniendo así la conexión WebSocket activa y funcional.

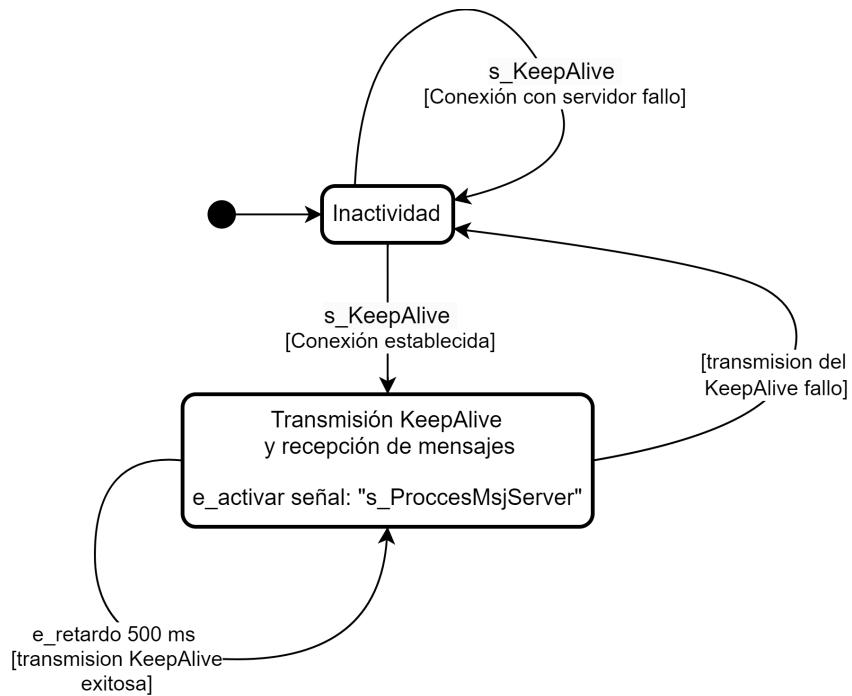


Figura 1.47: Máquina de estados para mantener la conexión con el servidor y elevar un raise en caso de que se reciba información del servidor.

Puesto que se puede recibir datos del servidor, se declaran las siguientes variables del Código

1.2

```

1 //Procesamiento de comandos
2 //Size de comandos
3 var sizeBuff: uint16_t =0
4 var ptrSizeBuff: ptrUnt16_t
5 //Estructura de configuración a través de comandos
6 var configCommand:config_t
7 var ptrConfigCommand:ptrConfig_t=configCommand.pointer

```

Código 1.2: Declaración de variables para validar y procesar la recepción de datos del servidor.

En el estado **TX_MESSAGE**, independientemente de si se han recibido datos del servidor, se genera un evento hacia la región **ProcesoDeComandos**. Este evento facilita la transición del estado **IDLE** al estado **PROCESS_COMMAND**, condicionada a que la longitud del mensaje recibido del servidor sea mayor a cero. Si se cumple esta condición, en el estado **PROCESS_COMMAND** se invoca la función **opProcessMessageFromServer(ptrSizeBuff, ptrConfigCommand)**, que se encarga de procesar los mensajes provenientes del servidor. Estos mensajes pueden incluir comandos de configuración para el datalogger o valores de referencia para el control del túnel, los cuales serán detallados en la sección 1.3.4.

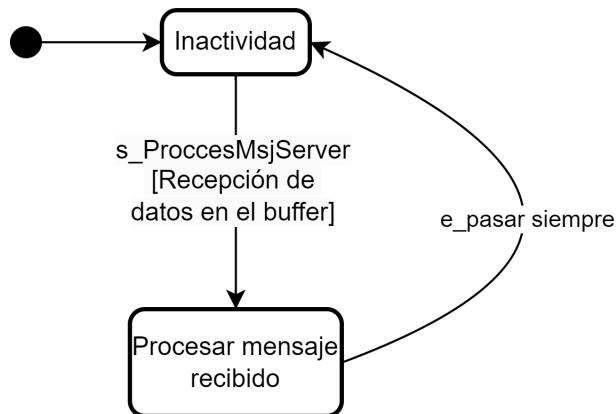


Figura 1.48: Máquina de estados para procesar los mensajes recibidos del servidor WebSocket.

Para gestionar eficientemente los comandos destinados a configurar diversas partes del datalogger con la información proporcionada por el software, se establece un diccionario de punteros a función. Este diccionario, ilustrado en el código 1.3, permite ejecutar una función específica en respuesta a cada comando o petición contenida en el mensaje recibido.

```

1 char *diccCommands []={
2     "start",
3     "stop",
4     "setIBC",
5     "setPAT",
6     "setTimes",
7     "refWindVel",
8     "NAN"};
9 bool_t (*diccProcesos [])(config_t * commandConfig)={
10     start,
11     stop,
12     setIBC,
13     setPAT,
14     setTimes,
15     refWindVel};

```

Código 1.3: Funciones que permiten configurar o modificar el comportamiento del datalogger.

1.3.2. Muestreo, adquisición y procesamiento

Para almacenar las muestras de los sensores de viento, patrón y bajo calibración, así como los niveles de tensión de la fuente y la tensión que ingresa al variador de velocidad, se han declarado las variables, punteros y constantes mostrados en el Código 1.4. Cada variable está asociada a un puntero, ya que cada función de cada estado o transición requiere recibir un puntero específico a una variable para poder modificar su valor.

```

1 //Variable interna para guardar las muestras de voltaje DC
2 var viVoltajeMuestra: real32_t
3 //Puntero Correspondiente
4 var ptrVoltajeMuestra: ptrReal32_t

```

```

5 //MuestraDelAdcDelTunnel
6 //Variable interna para guardar las muestras de voltaje DC
7 var viTunelTensionMuestra: real32_t
8 //Puntero Correspondiente
9 var ptrTunelTensionMuestra: ptrReal32_t
10 // IBC
11 var ptrAnemoIBC:ptrAmenometerSerialParam_t = configCommand.AnemoIBC.pointer
12 //PAT
13 var ptrAnemoPat:ptrAmenometerSerialParam_t=configCommand.AnemoPAT.pointer

```

Código 1.4: Variables para almacenar en cada iteración una muestra de su magnitud correspondiente.

Para configurar cada anemómetro, se ha declarado una estructura como se muestra en el fragmento de código 1.5. Esta estructura se inicializa con datos ingresados por el operador a través de la aplicación web descrita en el capítulo ???. Cada estructura requiere configurar el número de Uart, el LED de recepción y el BaudRate, valores que deben coincidir con las especificaciones de cada anemómetro. Además, se dispone de un buffer de 100 caracteres para almacenar los strings de datos recibidos, con un puntero asociado para modificar su contenido. También se incluyen dos arreglos de números reales para almacenar los datos numéricos convertidos a **real32_t** a partir de los strings de cada sensor, cada uno con su respectivo puntero para su modificación. Por último, se introduce el tipo de dato del sensor, **sensor_t**, que enumera los sensores utilizados.

```

1 typedef struct{
2     uint16_t Uart;
3     uint16_t LED;
4     uint32_t BaudRate;
5     char Buffer[100];
6     char * ptrUartBuffer;
7     real32_t DataAnemometer[7];
8     real32_t * ptrDataAnemometer;
9     sensor_t Sensor;
10 }amenometerSerialParam_t;

```

Código 1.5: Estructura de datos para anemómetro.

En las Figuras ?? y ?? se muestran las regiones que contienen las máquinas de estado **MuestraIBC** y **MuestraPAT**, respectivamente, para adquirir los datos de viento. La máquina de estado, por ejemplo, del IBC comienza en el estado **IDLE**. Cuando el comando **startMeasure** es **True**, se transiciona al estado **SENSOR_IBC**, donde se invoca la función **opBufferRS485_On(prtAnemo)**. Esta función recibe un puntero con todos los datos de configuración del anemómetro y los carga en su respectiva UART. Además, inicializa un callback y habilita la interrupción correspondiente de la UART. De esta manera, cada vez que los sensores envían datos, se activa una función que se encarga de procesar y separar los datos recibidos. Cuando **startMeasure** se configura como **false**, la máquina de estado transiciona al estado **OFF_SENSOR_IBC**, donde se desactiva la interrupción y el microcontrolador deja de tomar muestras de los anemómetros.

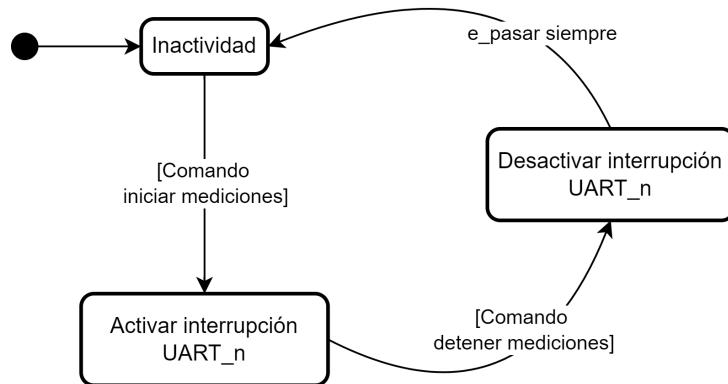


Figura 1.49: Máquina de estados para tomar muestras de los sensores viento.

El string del patrón VAISALA, modelo WMT700 tiene el formato `$-MWW,021,R,003.34,N,A*14,` mientras que el del instrumento bajo calibración DELTAOHM, modelo HD51.3 tiene el formato `28.30 359.3 998.3<CR><LF>`. Para procesar y separar estos datos, se utiliza un diccionario de punteros a funciones, como se muestra en el fragmento de Código 1.6. Desde el software, el operador preconfigura el modelo del anemómetro y, por lo tanto, a la máquina de estados, asegurándose de llamar a la función correspondiente para procesar los datos de cada anemómetro.

```

1 // Diccionario de punteros a función para procesar en función del sensor elegido
2 void (*opProcesoDatosViento[]) (amenometerSerialParam_t * data) = {
3     opProcesoDeltaOHM,
4     opProcesoWMT700,
5     opProcesoVentus
6 };
  
```

Código 1.6: Diccionario de punteros a función para procesar los datos recibidos de los anemómetros.

En las Figuras ?? y ?? se muestran las regiones que contienen las máquinas de estado **MuestreoBateria** y **MuestreoAdcTunel**, respectivamente, para adquirir los niveles de tensión de la fuente de alimentación y los niveles de tensión del variador de velocidad en las variables declaradas en el Código 1.4. Por ejemplo, para el muestreo del ADC del variador, la máquina de estado comienza en el estado **INICIO** y automáticamente transita al estado **IDLE**. Luego, siempre que la condición **startMeasure** sea igual a **True**, se produce una transición al estado **ADQ_BAT**, donde se invoca la función **opAdquirirDNB(ptrVoltajeMuestra)** para tomar una muestra del nivel de tensión. Después de transcurrido el tiempo definido en **IntervaloMuestra** en segundos, se procede a guardar las muestras y se eleva (se hace un *raise*) a través del evento (señal) **siMuestrasNuevas** que permite activar una máquina de estado de otra región. Finalmente, se regresa al estado **IDLE** para repetir el ciclo indefinidamente o hasta que la condición **startMeasure** sea **False**.

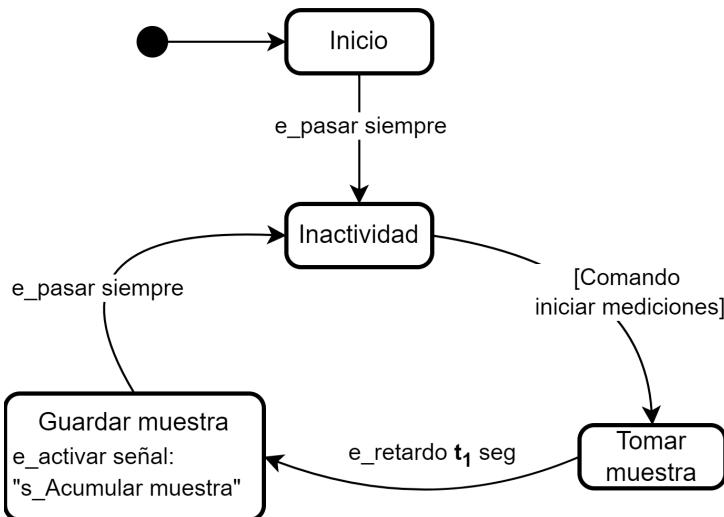


Figura 1.50: Máquina de estado para tomar muestras de la tensión del alimentacion y del variador.

En el Código 1.7 se muestra las variables declaradas para contabilizar el número de muestras y la cantidad de datos acumulados cada vez que se realiza una iteración para tomar muestras de los sensores de viento y los niveles de tensión detallados arriba.

```

1 // Adquisicion y procesamiento
2 var NumMuestra: uint16_t = 0
3 var ptrNumMuestra: ptrUnt16_t
4 var cantDatosProcesado: int32_t = 0
5 var ptrCantDatosProcesado: ptrInt32_t
  
```

Código 1.7: Variables para controlar la adquisición y procesamiento de los datos .

La región **Procesamiento** de la figura 1.51 inicia en un estado de **IDLE**. Se produce la transición al estado **ACUMULO_MUESTRAS** cuando la señal elevada **siMuestrasNuevas**, generada en la región **MuestreoBateria**¹, está activada y se cumplen dos condiciones: primero, que el número de muestras (**NumMuestra**) sea menor al cociente entre el intervalo de Tabla y el intervalo de muestra; segundo, que la condición **startMeasure** sea **True**. En este nuevo estado, se llama a la función **opAcumular(ptrNumMuestra, ptrVoltajeMuestra, ptrTunelTensionMuestra, ptrAnemoIBC, ptrAnemoPat)**, que se encarga de guardar las muestras en un arreglo, se incrementa el número de muestras en una unidad y luego se regresa al estado **IDLE**.

Por ejemplo, si el operador definió un intervalo de muestras de 1 segundo y un intervalo de tabla de 10 segundos, el cociente es 10. Cuando el número de muestras (**NumMuestra**) llega a 10, se produce la transición al estado **PROCESO_y_ALMACENO**. En este estado, se llama a la función **opProceso(ptrNumMuestra, ptrCantDatosProcesado, ptrPwmValue)**, se calcula el promedio, mínimo y máximo de las 10 muestras tomadas cada 1 segundo. Luego, se reinicia el número de muestras a cero. Finalmente, se eleva la señal **siDatoProcesado** para activar la

¹ Descrita anteriormente como un ejemplo de relación entre regiones mediante señales elevadas

región de transmisión de datos y se vuelve al estado **IDLE** para acumular las próximas 10 mediciones. Todos estos datos, incluyendo el último dato considerado como el dato instantáneo, son guardados en el microcontrolador para posteriormente ser transmitidos al servidor.

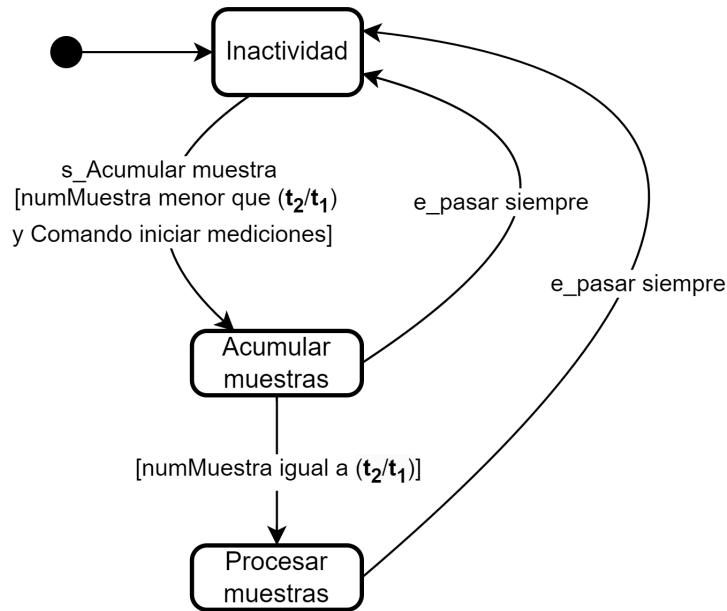


Figura 1.51: Máquina de estados para acumular muestras en una tabla y realizar un preprocesamiento de esos datos.

1.3.3. Transmisión al servidor

El protocolo WebSocket establece una comunicación en tiempo real entre clientes y servidores, adaptando el encabezado del mensaje según su tamaño. Para mensajes hasta 125 bytes, se usa un encabezado de 2 bytes. Para mensajes mayores, se añade un encabezado extendido con 2 bytes adicionales para la longitud. Independientemente del tamaño, todos los mensajes se enmascaran con una clave de 32 bits aplicada mediante XOR a la carga útil, protegiendo la transmisión contra vulnerabilidades y garantizando la integridad de los datos. Este mecanismo asegura que todos los mensajes, independientemente de su tamaño, se transmitan de manera confiable y conforme a las normas del protocolo. Para una transmisión exitosa desde el datalogger hacia el servidor, se declara la variable de Código 1.8, esta variable controla el estado de la transmisión en la máquina de estados de la región **TransmisionAlServer** de la figura 1.52.

```

1 // Transmision por Ethernet
2 var TransmisionExitosa: uint8_t=FALSE
  
```

Código 1.8: Declaración de variable para controlar el estado de una transmisión al servidor.

La máquina de estados de la figura 1.52 inicia en el estado **IDLE**. Cuando se detecta la señal **siDatosListosParaBdtos** y la condición **isConnectWebServer** está en **True**, se realiza una

transición al estado **TX_MESSAGE**. En este estado, se invoca la función `opTransmitWebSocketEthernet(ptrCantDatosProcesado)`, la cual tiene como responsabilidad codificar y enviar el mensaje a través del socket conectado al servidor web.

La codificación del mensaje depende de su longitud, para ello se implementaron dos funciones internas:

- Si la longitud es de hasta 125 bytes, se utiliza la función `encodeMessage125()`. El encabezado del mensaje codificado consta de 2 bytes.
- Si la longitud supera los 125 bytes, se emplea la función `encodeMessage126()`, que añade un campo adicional de 2 bytes para indicar la longitud extendida del mensaje.

Tras la codificación, se procede con el envío del mensaje. El servidor, al recibir el mensaje, responde con la longitud del mensaje recibido. Si la longitud coincide con la del mensaje enviado, se considera que la transmisión ha sido exitosa. De lo contrario, se asume que la transmisión ha fallado.

En caso de fallo en la transmisión, la máquina de estados transiciona al estado **BACK_UP**. Este estado se encarga de almacenar los datos hasta que se restablezca la comunicación con el servidor. Si la transmisión es exitosa, se incrementa un contador y se retorna al estado **IDLE**, quedando listo para una nueva transmisión.

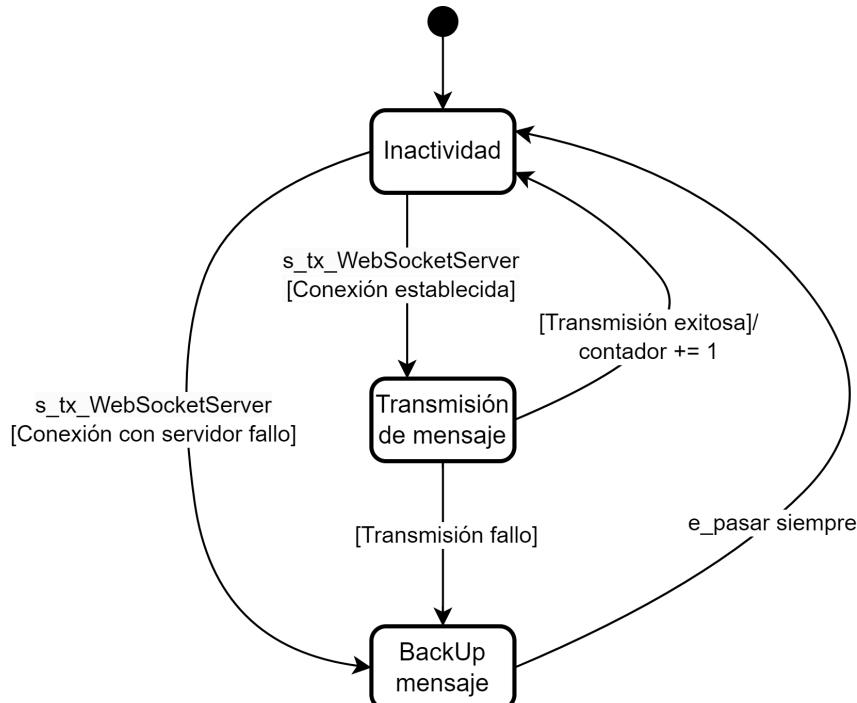


Figura 1.52: Máquina de estados para gestionar la transmisión de datos al servidor WebSocket.

1.3.4. Sistema de control PID

En la Sección 1.1.4, se detalla el desarrollo del circuito **PWM**. Este circuito, al recibir un valor digital entre 0 y 255, produce un voltaje de salida que oscila entre 53 mV y 3,5 V. Este voltaje se utiliza para controlar el variador de velocidad del motor que trae el túnel de viento, permitiendo variar la velocidad del aire entre 1 m s^{-1} y 25 m s^{-1} .

El circuito PWM se utiliza para la implementación de un controlador **PID** de lazo cerrado. La referencia de la velocidad del viento, en metros por segundo, se obtiene del servidor. Paralelamente, se toma una muestra de la velocidad del viento a través de un sensor patrón, también en metros por segundo. La diferencia entre la referencia y la muestra se define como el error, sobre el cual se aplica el control PID.

Las variables necesarias para el control PID se declaran como se muestra en el Código 1.9. Las variables `viIntesidadPatron` y `viIntesidadReferencia` almacenan las muestras del viento patrón y la referencia respectivamente. La variable `viPwmValue` guarda el valor de PWM calculado por el controlador, y en `viPid` se almacenan los parámetros del controlador PID.

```

1 // Control del tunel
2 // Valor del patron de viento
3 var viIntesidadPatron: real32_t = 0
4 var ptriIntesidadPatron: ptrReal32_t
5 // Valor de referencia de viento
6 var viIntesidadReferencia: real32_t = 0
7 var ptriIntesidadReferencia: ptrReal32_t
8 // Valor de PWM
9 var viPwmValue: int32_t = 0
10 var ptrPwmValue: ptrInt32_t = &viPwmValue
11 // Controlador PID
12 var viPid: pidController_t
13 var prtPid: ptrPidController_t = &viPid

```

Código 1.9: Declaración de variables para el controlador PID del motor del túnel de viento.

En el Código 1.10, se presenta la declaración de una estructura que define los parámetros del controlador PID. Estos parámetros incluyen las ganancias proporcional (`kp`), integral (`ki`) y derivativa (`kd`). Además, se registran variables para el manejo del error actual (`error`), el error anterior (`lastError`), la suma acumulada del error (`integral`) y la derivada del error (`derivative`) y se incluye el tiempo de muestreo (`Ts`), puesto que es un controlador en tiempo discreto.

```

1 // Estructura del controlador PID
2 typedef struct {
3     real32_t kp;          // Ganancia proporcional
4     real32_t ki;          // Ganancia integral
5     real32_t kd;          // Ganancia derivativa
6     real32_t error;       // Error actual
7     real32_t lastError;   // Error anterior
8     real32_t integral;    // Integral del error

```

```

9  real32_t derivative; // Derivada del error
10 real32_t Ts;          // Tiempo de muestreo
11 } pidController_t;

```

Código 1.10: Estructura para los parámetros y variables del controlador PID.

Finalmente, se aplica el algoritmo de control **PID** sobre el error calculado para regular la salida del circuito PWM y, consecuentemente, la velocidad del viento en el túnel. La máquina de estados comienza en el estado **INICIO**, donde se ejecuta la función `configPID(prtPid)`. Esta función configura los parámetros del controlador PID, obtenidos heurísticamente, con $kp = 3$; $kd = 0,5$; $ki = 0$; y el tiempo de muestreo $Ts = 1\text{s}$. Además, prepara el pin digital de la EDU-CIAA para habilitar la salida PWM con un valor inicial de cero.

Si el indicador `starMesuare` se encuentra en `True`, se transiciona al estado **MESUARE_PAT**. En este estado, se invoca la función `opMesuareWindPAT(ptrAnemoPat, ptriIntesidadPatron, ptriIntesidadReferencia)`, que obtiene la intensidad instantánea del viento del anemómetro patrón y la intensidad de referencia enviada por el servidor, almacenando estos valores en variables correspondientes.

Posteriormente, se avanza al estado **CONTROL_TUNNEL**. Aquí, se llama a la función `pidControlTunnel(prtPid, ptriIntesidadPatron, ptriIntesidadReferencia, ptrPwmValue)`, que ejecuta las siguientes acciones:

- Calcula el error proporcional $e_i = vel_{ref_i} - vel_{pat_i}$.
- Determina el término integral $e_{int_i} = e_i \cdot Ts$.
- Estima el término derivativo $e_{dev_i} = \frac{e_i - e_{i-1}}{Ts}$.
- Actualiza el error y calcula la acción de control como $u_c = kp \cdot e_i + ki \cdot e_{int_i} + kd \cdot e_{dev_i}$.

El valor de u_c se somete a un saturador entre 0 y 255, y luego se aplica al pin de salida PWM. Este ciclo se repite cada 1s , correspondiente al tiempo de muestreo Ts .

Si el operador detiene las mediciones a través del software, se envía un comando que establece el indicador `starMesuare` en `False`. Esto hace que el controlador se ponga en cero y permanezca en espera para iniciar nuevas mediciones.

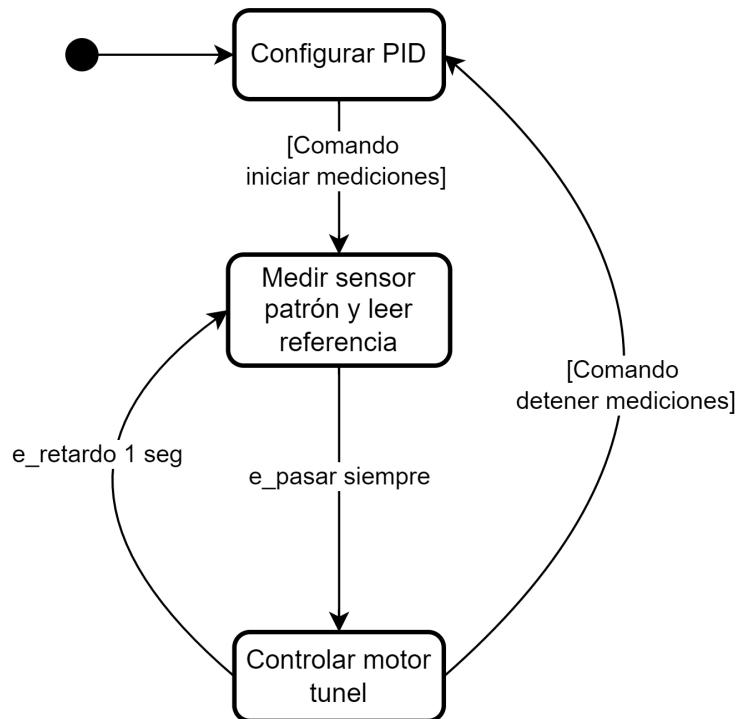


Figura 1.53: Máquina de estados que ejecuta el control PID con valores de referencia enviados por el servidor.

Apéndice A

Datalogger y sensores

A.1. Principio de medición para sensores de ultrasonido

La velocidad y dirección del viento se determinan midiendo el tiempo que tardan los pulsos ultrasónicos en recorrer la distancia desde el transductor que genera el pulso hasta el transductor receptor. El instrumento utiliza dos pares de transductores orientados a lo largo de dos ejes ortogonales. Detectar la velocidad del viento a lo largo de dos ejes permite determinar no solo la intensidad sino también la dirección del viento.

El instrumento mide el tiempo de viaje del pulso ultrasónico entre los dos transductores de la misma pareja en ambas direcciones, como se muestra en la figura A.1. Los tiempos de viaje en las dos direcciones opuestas se definen como t_A (tiempo en dirección hacia adelante) y t_R (tiempo en dirección inversa). Si la velocidad del viento es cero, los valores de t_A y t_R son iguales. En presencia de viento, uno de los dos valores de tiempo es mayor que el otro y la comparación entre los dos valores de tiempo permite determinar la dirección y la intensidad del viento.

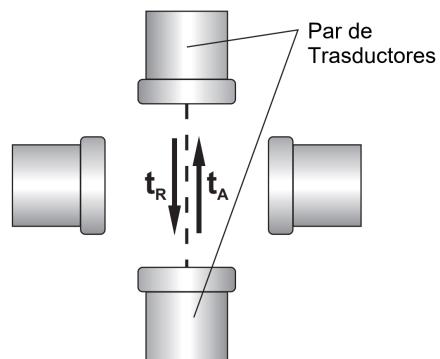


Figura A.1: Esquema de los transductores y el tiempo de viaje de un pulso ultrasónico. [2]

El tiempo de viaje del pulso de ultrasonido esta dado por las ecuaciones A.1 y A.2

$$t_A = \frac{D}{C + Vw} \quad (\text{A.1})$$

$$t_R = \frac{D}{C - Vw} \quad (\text{A.2})$$

donde se define:

- D: Distancia entre los dos transductores del mismo par.
- C: Velocidad del sonido.
- V_w : Componente de la velocidad del viento a lo largo del eje de medición.

Realizando la medición de los dos tiempos de viaje en los ejes cartesianos **U** y **V** podemos determinar las componentes de velocidad de viento como

$$V_{w_i} = \frac{D}{2} \left(\frac{1}{t_{a_i}} - \frac{1}{t_{R_i}} \right), \quad i = u, v \quad (\text{A.3})$$

El eje **U** es un eje que va de oeste a este, mientras que el eje **V** es un eje que va de sur a norte. Medir el tiempo de viaje en ambas direcciones también permite cancelar la dependencia de la velocidad de transmisión de los ultrasonidos en el aire de las condiciones ambientales de temperatura, humedad y presión barométrica.

A.2. Especificaciones técnicas y hojas de datos

Velocidad de viento	
Sensor	Ultrasonido
Rango de Medición	0 m s ⁻¹ a 75 m s ⁻¹
Resolución	0,1 m s ⁻¹
Precisión	±0,2 m s ⁻¹ o 2 % de la medición hasta 60 m s ⁻¹ 3 % de la medición para mayor que 60 m s ⁻¹
Dirección del viento	
Sensor	Ultrasonido
Rango de Medición	0° a 359,9°
Resolución	0,1°
Precisión	±2° RMSE (velocidad del viento 2 m s ⁻¹)
Fuente de alimentación	
Fuente de alimentación del calentador	24 V Vdc ±10 %
Consumo de energía del calentador	30 W
Fuente de alimentación del instrumento (sin calentador)	12 V a 30 V Vdc
Consumo de energía del instrumento (sin calentador)	60 mA @ 24 V Vdc
Otras especificaciones	
Salida Serial	RS232 aislado y RS485
Protocolo de comunicación	NMEA, MODBUS-RTU, ACII propietario
Intervalo de medición	Configurable de 250 ms a 1 s
Intervalo de velocidad promedio	Configurable de 1 s a 10 min
Conexión eléctrica	19-pole M23 conector Macho
Temperatura de operación	-40 °C a +70 °C
Grado de protección	IP66

Tabla A.1: Especificaciones de sensor de viento, marca Delta Ohm, modelo HD51.3.

Apéndice B

Software

B.1. Algoritmo del generador de trayectoria

```
1 def trajectoryGenerator(numCalib):
2     # Por ahora asumimos que el sistema empieza con una velocidad inicial del viento de
3     # 1 m/s que es lo que sucede generalmente al encender el tunel
4     vel_init = 1
5     vel_end = [1,1,1]
6     calibracion_instance = get_object_or_404(Calibracion, numeroDeCalibracion=numCalib)
7     datos_config_tunnel = ConfigTunel.objects.filter(calibracion=calibracion_instance)
8     puntosCalibrar = list(datos_config_tunnel)[0].puntosCalibrar
9     tipoDeMedicion = list(datos_config_tunnel)[0].tipoDeMedicion
10    if tipoDeMedicion == "Solo Ascendente":
11        velRef = puntosCalibrar.split(",")
12    if tipoDeMedicion == "Solo Descendente":
13        velRef = puntosCalibrar.split(",")
14        velRef = velRef[::-1]
15    if tipoDeMedicion == "Asc - Des":
16        velRefA = puntosCalibrar.split(",")
17        velRefB = velRefA[::-1]
18        velRef = velRefA + velRefB
19        pass
20
21    velRef = [int(x) for x in velRef]
22    aux = []
23    for i in range(len(velRef)):
24        aux.append([velRef[i]] * 3) # se multiplica por 3 ya que quiero que llegue,
25                                # estabilice y mida
26
27    velRef = sum(aux, []) # convertimos a un vector la matriz auxiliar
28    velRef = [vel_init] + velRef + vel_end
29    # Tiempos en minutos, ingresados por el usuario, en principio es el mismo para todos
30    # los puntos
31    tiempoTransitorio = list(datos_config_tunnel)[0].tiempoTransitorio*60
32    tiempoEstabilidad = list(datos_config_tunnel)[0].tiempoEstabilidad*60
33    tiempoMedicion = list(datos_config_tunnel)[0].tiempoMedicion*60
34    cantidadPuntos = list(datos_config_tunnel)[0].cantidadPuntos
35    if tipoDeMedicion == "Solo Ascendente" or tipoDeMedicion == "Solo Descendente" :
36        t_des = [tiempoTransitorio,tiempoEstabilidad, tiempoMedicion] * (cantidadPuntos +
37                           1)
38    else:
```

```

36     aux = [tiempoTransitorio , tiempoEstabilidad , tiempoMedicion]
37     t_des = (aux * (cantidadPuntos)*2 ) + aux
38
39
40 # Armamos la trayectoria / perfil de viento de referencia para el controlador
41 # trajectoryVector = [0]*1000
42 trajectoryVector = []
43 idx = 0
44 t_s = 1 # tiempo de muestreo en seg
45 t_acc = 10 # tiempo de aceleracion en seg
46 rapidez_max = 20 # pendiente maxima puede soportar el tunel, quiza tenga que ver con
    el saturador,255
47 qA = velRef[0] # punto de partida
48 for i in range(0,len(velRef)-1):
49     qB = velRef[i] # donde iba
50     qC = velRef[i+1] # donde voy
51     dA = qA-qB # de la zona 1
52     dC = qC-qB # de la zona 2
53     # Tj = max([dC/rapidez_max ,t_des[i]])
54     Tj = t_des[i]
55
56     for t_segm in range(-t_acc+t_s,t_acc,t_s): # zona 1
57         trajectoryVector.append((dC/Tj)*(((t_segm+t_acc)**2) /( 4*t_acc) ) + (dA/
            t_acc)*(((t_segm-t_acc)**2) / (4*t_acc)) + qB)
58
59     for t_segm in range(t_acc+t_s,Tj-t_acc+2,t_s): # zona 2
60         trajectoryVector.append( (dC/Tj)*t_segm + qB)
61     qA = trajectoryVector[idx-1]
62 # Redondeamos a 3 decimales los elementos de referencia
63 trajectoryVector = [round(numero , 3) for numero in trajectoryVector]
64 # Crear un arreglo de 1 a idx
65 t = np.arange(1, idx+1)
66 # Multiplicar cada elemento por t_sampling
67 t = t * t_s
68 # Graficamos la trayectoria armada
69 plt.plot(t/60,trajectoryVector )
70 plt.xlabel('t')
71 plt.ylabel('Vel Viento')
72 plt.title('Grafico de t vs Vel Viento')
73 plt.grid(True)
74 plt.show()
75 return trajectoryVector

```

Código B.1: Algoritmo generador de referencias para el controlador PID.

B.2. Comandos entre el servidor WebSocket y el datalogger

Comando	Descripción
<code>start</code>	El comando <code>start</code> inicia las mediciones del datalogger.
<code>stop</code>	El comando <code>stop</code> detiene las mediciones y la transmisión de datos del datalogger.
<code>setIBC;Uart_n;LED_n;BaudRate;sensorModel</code>	El comando <code>setIBC</code> configura: <ul style="list-style-type: none"> ■ El número de puerto COM. ■ El Led de recepción. ■ El baud Rate del sensor. ■ El modelo de sensor para llamar a la función que parsea los datos.
<code>setPAT;Uart_n;LED_n;BaudRate;sensorModel</code>	El comando <code>setPAT</code> configura: idem al comando anterior.
<code>setTimes;timeSampling;timeTable</code>	El comando <code>setTimes</code> configura: <ul style="list-style-type: none"> ■ El tiempo de muestreo. ■ El tiempo de tabla.
<code>refWindVel;valueWindRef</code>	El comando <code>refWindVel</code> envía <ul style="list-style-type: none"> ■ Un valor de velocidad de viento de referencia para el controlador.

Tabla B.1: Listado de comandos transmitidos por el servidor WebSocket al datalogger para configurar los parámetros seleccionados por el usuario.

Apéndice C

Cálculo de incertidumbre

En la tabla de la figura C.1 se muestra el cálculo, a través de una hoja de cálculo de Excel o una calculadora, para obtener la incertidumbre combinada en las mediciones de un anemómetro bajo calibración marca DeltaOhm, modelo HD51.3 en 10 m s^{-1} con un patrón Vaisala, modelo WMT700. Se cuantifican todas las incertidumbres, se normalizan por su factor de distribución, se multiplica por su coeficiente de sensibilidad que es la unidad en todos los casos y luego se realiza la suma de los cuadrados. En particular, se obtuvo una incertidumbre combinada de $u_{com} = \sqrt{0,07} = 0,26\text{ m s}^{-1}$ y, aplicando el factor de cobertura de 2, se obtiene una incertidumbre expandida de $U_{exp} = 2 \cdot 0,26 = 0,52\text{ m s}^{-1}$.

Por otro lado, se puede observar, en la última columna, el peso en porcentaje de la incertidumbre estándar al cuadrado para cada fuente. Se observa que las incertidumbre con mayor peso son el valor de calibración del patrón y el ajuste de la corrección de los datos del certificado del patrón. Por último, dado que el túnel no está caracterizado en su volumen, el aporte de las fuentes de incertidumbre se consideró nulo.

El software desarrollado realiza de forma iterativa estos cálculos para todos los puntos y devuelve un gráfico y una tabla con los resultados finales. De esta forma, el software permite optimizar en tiempo el procesamiento de los datos.

C.1. Ejemplo de Presupuesto de incertidumbre para un punto

Presupuesto de calibración para 10 m/s										
Fuentes de incertidumbre		Tipo de Incertidumbre	Valores $m s^{-1}$	Distribución	Factor de normalización	c_i	σ_i	$u_i = c_i \cdot \sigma_i$	u_i^2	Peso de u_i^2 (%)
Patrón	Calibración	B	0,279	N	2,00	1,00	0,139	0,14	0,02	29,17
	Ajuste de calibración	A	0,049	N	1,00	1,00	0,049	0,05	0,00	3,64
	Resolución	B	0,010	R	1,73	1,00	0,006	0,01	0,00	0,05
	Repetibilidad	A	0,010	N	9,80	1,00	0,010	0,01	0,00	0,15
	Histéresis	B	-0,020	R	1,73	1,00	- 0,012	-0,01	0,00	0,20
	Ajuste de la corrección del patrón	A	0,210	N	1,00	1,00	0,210	0,21	0,04	66,13
	Factor de bloqueo	B	0,002	RMS	1,00	1,00	0,002	0,00	0,00	0,00
Túnel de viento	Homogeneidad	B	0,000	R	1,00	1,00	0,000	0,00	0,00	0,00
	Ajuste de la homogeneidad	A	0,000	N	1,00	1,00	0,000	0,00	0,00	0,00
	Estabilidad	B	0,000	R	1,73	1,00	0,000	0,00	0,00	0,00
	Ajuste de la estabilidad	A	0,000	N	1,00	1,00	0,000	0,00	0,00	0,00
	Factor de calibración	B	0,000	RMS	1,00	1,00	0,000	0,00	0,00	0,00
IBC	Resolución	B	0,010	R	1,73	1,00	0,006	0,01	0,00	0,05
	Repetibilidad	A	0,010	N	9,80	1,00	0,010	0,01	0,00	0,15
	Histéresis	B	-0,030	R	1,73	1,00	- 0,017	-0,02	0,00	0,45
	Factor de bloqueo	B	0,002	RMS	1,00	1,00	0,002	0,00	0,00	0,00
TOTAL								0,40	0,07	100,00

Tabla C.1: En la tabla se indica el calculo de incertidumbre para un punto en particular $10 m s^{-1}$.

Bibliografía

- [1] NXP Semiconductors. *LPC435x/3x/2x/1x 32-bit ARM Cortex-M4/M0 MCU; up to 1 MB flash and 136 kB SRAM; Ethernet, two High-speed USB, LCD, EMC*. Ver. Rev. 5.3. Mar. de 2016. URL: https://www.nxp.com/docs/en/data-sheet/LPC435X_3X_2X_1X.pdf.
- [2] Delta OHM. *DeltaOHM HD51.3D Series Operating Manual*. 2023. URL: https://www.deltaohm.com/wp-content/uploads/document/DeltaOHM_HD51.3D_manual_ENG.pdf.
- [3] Embedded There. *RS485 Communication Protocol: Basics, Working Principle and Applications*. Accessed: July 23, 2024. 2023. URL: <https://embeddedthere.com/rs485-communication-protocol>.
- [4] Proyecto CIAA. *Ponchos de la CIAA - Desarrollo*. Sitio web. Accedido: 10 de julio de 2024, git clone <https://github.com/ciaa/Ponchos>. 2024. URL: <https://www.proyecto-ciaa.com.ar/devwiki/doku.php%3Fid=desarrollo:ciaa:ponchos.html>.
- [5] EEVblog. *EEVblog #225 - Lab Power Supply Design Part 4 - PWM Control*. YouTube. Accedido: 4 de julio de 2024. 2024. URL: <https://www.youtube.com/watch?v=YaRDbw38x7Q>.
- [6] Diodes Incorporated. *1N5819 Schottky Barrier Rectifier*. Ver. Rev. 1.0. Ago. de 2024. URL: <https://www.diodes.com/assets/Datasheets/ds23001.pdf>.
- [7] CIAA Project. *firmware_v3: Framework para desarrollo de Firmware de Sistemas Embedidos en C/C++*. GitHub. Accedido: 10 de julio de 2024. 2024. URL: https://github.com/ciaa/firmware_v3.
- [8] Proyecto CIAA. *Proyecto CIAA: Computadora Industrial Abierta Argentina*. <https://www.proyecto-ciaa.com.ar/>. Accedido: 1 de julio de 2024. 2024.
- [9] WIZnet. *W5100 Datasheet*. Consultado el: 2 de Julio de 2024. 2024. URL: https://ar.mouser.com/datasheet/2/443/W5100_Datasheet_v1.2.5-586411.pdf.
- [10] itemis CREATE: *Herramienta para desarrollar, simular y generar máquinas de estados finitos*. Sitio web de itemis. Accedido el 23 de julio de 2024. URL: <https://www.itemis.com>.

- [com/en/products/itemis-create/documentation/user-guide/overview_what_are_itemis_create_statechart_tools.](https://www.itemis.com/en/products/itemis-create/documentation/user-guide/overview_what_are_itemis_create_statechart_tools)
- [11] *UML state machine*. https://en.wikipedia.org/wiki/UML_state_machine. Accessed: 2024-08-26.