



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)

Director: Ing. Leonardo Carducci

Co-Directores: Dr. Ing. Lucas Sambuco

Ing. Claudio Arencibia

Jurado: Nombre y apellido

Nombre y apellido

Nombre y apellido

Buenos Aires, 24 de julio de 2024

Resumen

Abstract

Agradecimientos

Índice general

1. Introducción	1
1.1. Estado del arte	1
1.2. Motivación	1
1.3. Objetivos	2
2. Viento	3
2.1. Viento en superficie	4
2.2. Instrumentos de medición del viento	7
2.3. Túnel de viento	7
3. Cálculo de incertidumbre	9
3.1. Incertidumbre de la medición	9
3.2. Trazabilidad de las mediciones	9
3.3. Fuentes de incertidumbre	9
3.4. Modelo teórico de medición	9
4. Implementación del Datalogger	11
4.1. Desarrollo del Hardware	12
4.1.1. Módulo de alimentación eléctrica	14
4.1.2. Módulo de adquisición de datos de viento	15
4.1.3. Módulo de comunicación Ethernet/SPI	17
4.1.4. Circuito PWM	19
4.1.4.1. Simulación del circuito	20
4.1.4.2. Implementación del circuito	23
4.1.5. Circuito de adquisición de tensión del variador	29
4.1.5.1. Simulación del circuito	29
4.1.5.2. Implementación del circuito	31
4.1.6. Circuito de LEDs indicadores	32

4.2. Diseño y construcción del PCB	34
4.3. Desarrollo del Firmware	41
4.3.1. Configuración y conexión con los servidores	44
4.3.1.1. Servidor NTP	46
4.3.1.2. Servidor WebSocket	48
4.3.2. Muestreo, adquisición y procesamiento	52
4.3.3. Transmisión al servidor	57
4.3.4. Sistema de control PID	58
5. Implementación de la Aplicación WEB	62
5.1. Desarrollo aplicación WEB	63
5.1.1. Arquitectura del software	63
5.1.2. Back End	65
5.1.2.1. Base de datos	66
5.1.2.2. Servidor WebSocket	68
5.1.2.3. Generador de trayectoria y control del túnel	70
5.1.2.4. Cálculo de incertidumbre	72
5.1.3. Frond End	73
5.1.3.1. Carga de metadatos	73
5.1.3.2. Configuración del sistema	73
5.1.3.3. Adquisición de datos	73
5.1.3.4. Procesamiento de datos	74
5.1.3.5. Resultados	74
5.2. Integración con el hardware	74
6. Mediciones y Resultados experimentales	75
6.1. Caracterización de la zona de medición	75
6.1.1. Banco de medición	75
6.1.2. Procesamiento de datos	75
6.1.3. Resultados	75
6.2. Calibración Delta OHM HD500	76
6.2.1. Banco de medición	76
6.2.2. Procesamiento de datos	76
6.2.3. Resultados	76
6.3. Calibración Vaisala WMT700	76

6.3.1. Banco de medición	76
6.3.2. Procesamiento de datos	76
6.3.3. Resultados	76
7. Conclusiones	77
7.1. Comportamiento del sistema	77
7.2. Conclusiones finales	77
7.3. Trabajo futuro	77
A. Datalogger	78
A.1. Especificaciones técnicas y hojas de datos	78
A.2. Sección 2	79
B. Software	80
B.1. Algoritmo del generador de trayectoria	80
C. Cálculo de incertidumbre combinada	82

Índice de figuras

1.1. Here is a picture of a cat.	1
2.1. Tres componentes de la velocidad de viento v_x, v_y, v_z [$\frac{m}{s}$]. El vector de viento horizontal v_{xy} forma un ángulo de entrada θ con el vector de viento resultante v_{xyz}	3
2.2. Campo de viento representado con flechas en el sur de Sudamérica.	5
2.3. Representación gráfica del viento utilizando barbas.	5
2.4. Representación del viento en 850 hPa para el día 14/4/2016 a las 12 UTC en la región central de Argentina.	6
2.5. Viento (nudos) y líneas de corriente en 500hPa.	7
2.6. Túnel de viento.	8
4.1. Diagrama en bloques del sistema electrónico.	11
4.2. Placa de desarrollo EDU CIAA	12
4.3. Diagrama en bloques de la EDU-CIAA	12
4.4. Sensor Vaisala modelo WMT700	13
4.5. Sensor Delta OHM modelo HD51.3D	13
4.6. Regulador de voltaje DC-DC basado en el chip LM2596.	14
4.7. Esquemático del sistema de alimentación con módulo LM2596, diodo Zener, divisor resistivo y filtros de bypass para el LM358.	15
4.8. Esquema de conexión de datos y alimentación del sensor HD51.3D con un Data-logger o un PLC.	16
4.9. Puerto RS-485 integrado en la EDU-CIAA	16
4.10. Módulo convertidor de RS485 a UART (TTL)	17
4.11. Esquemático del módulo MAX485 para la recepción de datos del sensor bajo calibración.	17
4.12. Módulo W5100 utilizado para realizar la conectividad a internet a través de la interfaz SPI del microcontrolador.	18

4.13. Esquemático del módulo W5100 para la comunicación bidireccional con distintos servidores.	18
4.14. Esquemático circuital del tablero de control del tunel de viento.	19
4.15. Circuito diseñado para obtener distintos niveles de tensión mediante la variación del ciclo de trabajo de una señal cuadrada (PWM).	21
4.16. Circuito diseñado para obtener distintos niveles de tensión mediante la variación del ciclo de trabajo de una señal cuadrada (PWM).	21
4.17. Señales simuladas para un ciclo de trabajo al 100 % de la señal de entrada.	22
4.18. Señales simuladas para un ciclo de trabajo al 75 % de la señal de entrada.	23
4.19. Señales simuladas para un ciclo de trabajo al 55 % de la señal de entrada.	23
4.20. Esquemático del circuito PWM que controla la velocidad del túnel de viento por programación.	24
4.21. Banco de medición para caracterizar el circuito PWM.	24
4.22. 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.	25
4.23. 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.	25
4.24. 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.	26
4.25. 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.	26
4.26. 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.	27
4.27. 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.	29
4.28. 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 %.	30
4.29. 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 %.	30
4.30. Esquemático de la implementación del circuito que toma muestras de la señal continua que ingresa al variador de velocidad.	31
4.31. 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.	33

4.32. 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.	34
4.33. Template disponible para la realización de ponchos para la EDU-CIAA.	35
4.34. Diseño del esquemático que integra los módulos desarrollados.	36
4.35. Diseño PCB del Shield (poncho) que contiene todos los módulos descritos en las secciones anteriores.	37
4.36. Vista en 3D de la capa BOTTOM.	38
4.37. Vista en 3D de la capa TOP.	39
4.38. Shield conectado a la placa EDU-CIAA	40
4.39. Otra vista del Shield diseñado para esta aplicación.	40
4.40. Vista superior de la placa Shield conectada a la EDU-CIAA	41
4.41. Lanzador de aplicaciones integradas para el desarrollo de firmware para la EDU-CIAA.	42
4.42. Estructura de carpetas, del proyecto Datalogger	42
4.43. Lista de regiones con su respectiva prioridad.	43
4.44. Modos de configuración para el módulo W5100.	45
4.45. Diagrama de flujo para establecer una comunicación, en modo cliente, con un servidor utilizando sockets.	46
4.46. Máquina de estados para configurar el RTC del datalogger a través de un servidor NTP.	48
4.47. Máquina de estados para establecer una conexión bidireccional con un servidor WebSocket.	50
4.48. Máquina de estados para mantener la conexión con el servidor y elevar un raise en caso de que se reciba información del servidor.	51
4.49. Máquina de estados para procesar los mensajes recibidos del servidor WebSocket.	52
4.50.	54
4.51.	55
4.52. Máquina de estados para acumular muestras en una tabla y realizar un preprocesamiento de esos datos.	57
4.53. Máquina de estados para gestionar la transmisión de datos al servidor WebSocket.	58
4.54. Máquina de estados que ejecuta el control PID con valores de referencia enviados por el servidor.	61
5.1. Arquitectura del software implementada.	64

5.2. Patrón de diseño de software MVT, que divide la lógica del programa en tres elementos interconectados.	65
5.3. Diagrama entidad relación diseñada para la aplicación Web.	67
5.4. Segmento para pasar de un punto de velocidad v^B a otro v^C viiniendo desde v^A . .	71
5.5. Trayectoria generada utilizando el método de Paul para un ciclo ascendente y descendente de velocidades.	72
5.6. Diagrama de flujo del calculo de incertidumbre.	73
5.7.	74

Índice de tablas

1.1.	Parámetros configurados en el termohigrómetro patrón.	1
4.1.	Mediciones de tensión y corriente de la fuente del variador para distintas resistencias de carga.	20
4.2.	Mediciones para ciclos de trabajo en modo Ascendente, VadcTunnel y la velocidad del viento.	28
4.3.	Mediciones para ciclos de trabajo en modo Descendente, VadcTunnel y la velocidad del viento.	28
4.4.	Mediciones a la entrada pin <i>VADC</i> del variador y en el pin ADC_CH2 de la EDU-CIAA, para un barrido ascendente del ciclo de trabajo.	32
4.5.	Mediciones a la entrada pin <i>VADC</i> del variador y en el pin ADC_CH2 de la EDU-CIAA, para un barrido descendente del ciclo de trabajo.	32
4.6.	Diferencias entre los tipos de máquinas de estados.	43
A.1.	Especificaciones de sensor de viento, marca Delta Ohm, modelo HD51.3.	78

Capítulo 1

Introducción

1.1. Estado del arte

Figura 1.1: Here is a picture of a cat.

1.2. Motivación

Puerto Serie	Velocidad [Baud Rate]	9600
	Bit de Datos	8
	Bits de Parada	1
	Paridad	none
	Control de Flujo	none
Datos	Modo de Comunicación	Run
	Intervalo Run [segundos]	10
	Dirección de Dispositivo	0
	Formato de los Datos	Estándar Vaisala

Tabla 1.1: Parámetros configurados en el termohigrómetro patrón.

vemos la tabla 1.1

1.3. Objetivos

Capítulo 2

Viento

La velocidad del viento es una magnitud vectorial tridimensional que experimenta fluctuaciones aleatorias de pequeña escala en el espacio y en el tiempo, que se superponen a un flujo organizado de mayor escala. Para los fines de la observación meteorológica, la meteorología sinóptica y el pronóstico del tiempo, es crucial analizar el movimiento horizontal del aire que se define como viento. No obstante, es importante considerar que, para algunas aplicaciones, también es necesario conocer el movimiento vertical del viento. Ejemplos de esto son el aterrizaje de aeronaves y el estudio de la dispersión de contaminantes en la atmósfera. En la Figura 2.1 se muestran las tres componentes de un vector de viento \mathbf{v}_{xyz} , y sobre el plano XY se muestra la proyección \mathbf{v}_{xy} que no interesa analizar.

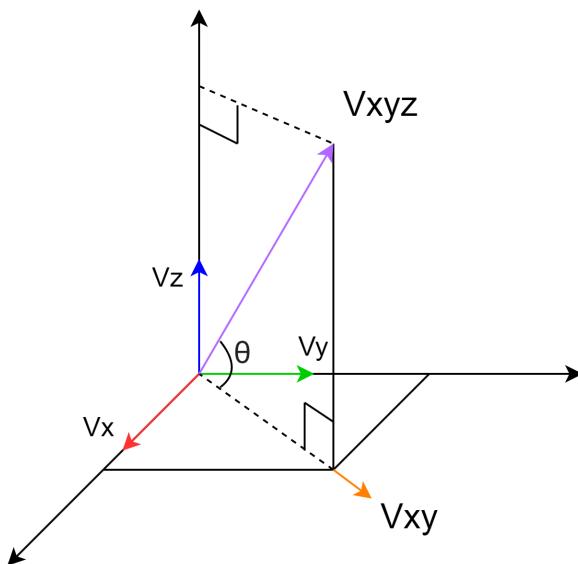


Figura 2.1: Tres componentes de la velocidad de viento v_x, v_y, v_z [$\frac{m}{s}$]. El vector de viento horizontal v_{xy} forma un ángulo de entrada θ con el vector de viento resultante v_{xyz} .

2.1. Viento en superficie

Se considerará que el viento de superficie es fundamentalmente una magnitud vectorial bidimensional definida por dos componentes

- **Intensidad:** Indica el valor de la velocidad del viento. En general se utilizan unidades como nudos (kn), metros por segundo (m s^{-1}) o kilómetros por hora (km h^{-1}).

- **Dirección y sentido:** Indica desde donde proviene el aire que se encuentra en movimiento.

Se mide en grados sexagesimales y considera el apartamiento con respecto al norte geográfico. Por ejemplo, un viento de dirección noreste (aire que se desplaza del noreste hacia el suroeste) tiene una dirección de 45° , ya que esta es la separación (en grados) entre el norte y el noreste.

El grado de fluctuación experimentado por el viento se denomina ráfagosidad, y las diferentes fluctuaciones son conocidas como ráfagas o rachas. La mayoría de los usuarios de datos sobre el viento necesitan conocer el viento horizontal en promedio, eliminando las fluctuaciones. Por ejemplo, en meteorología se utilizan promedios de entre 10 y 60 minutos con fines de predicción, mientras que en aeronáutica se utilizan promedios de 2 minutos. No obstante, cada vez son más las aplicaciones que precisan información acerca de la variabilidad o ráfagosidad del viento. Para este propósito, se utilizan tres magnitudes: la ráfaga o racha máxima, y las desviaciones típicas de la velocidad y de la dirección del viento [1].

La representación gráfica del viento se puede realizar de distintas formas, entre ellas se encuentran las flechas, las barbas y las líneas de corriente [2].

Las **flechas** son la forma más común de representar un vector. La intensidad del viento se representa con la longitud de la flecha, mientras que la dirección se indica por el ángulo respecto al norte. El sentido de la flecha siempre apunta "hacia dónde va el viento". En la figura¹ 2.2 se muestra un ejemplo en el que se observa que el viento en la Patagonia proviene principalmente del sector oeste (es decir, el aire se desplaza de oeste a este), por lo tanto, las flechas apuntan hacia el este.

¹Imagen extraída del curso de observadores <https://crf.smn.gob.ar/>

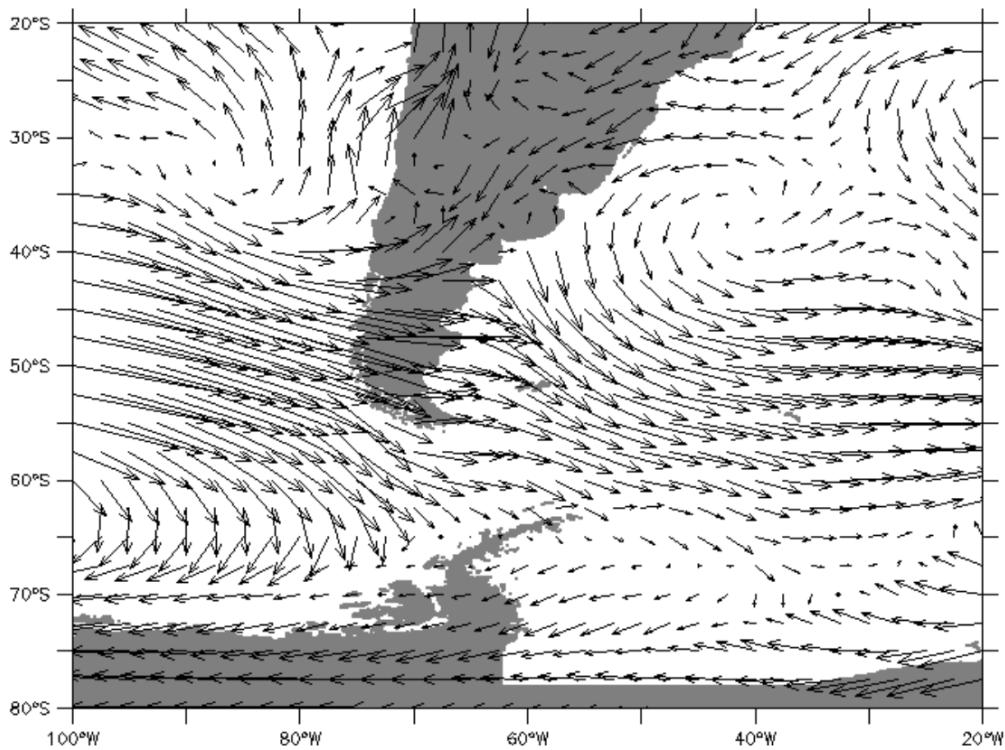


Figura 2.2: Campo de viento representado con flechas en el sur de Sudamérica.

Las **barbas** son ideogramas en los que la línea más larga indica la dirección del viento, mientras que las líneas más pequeñas, ubicadas en un extremo, indican la intensidad en múltiplos de 5 knot. En la figura 2.3, se puede observar una representación del viento del oeste con una intensidad de 15 nudos, donde la media flecha representa 5 knot y la flecha entera representa 10 knot. Más abajo, se observa una barba que representa el viento del sur con un banderín negro que indica una intensidad de 50 knot. La dirección del viento está dada por el ángulo respecto al norte verdadero y, como se mencionó anteriormente, indica desde dónde viene el aire.

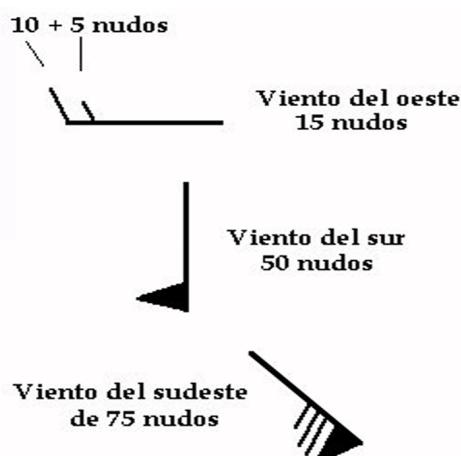


Figura 2.3: Representación gráfica del viento utilizando barbas.

Las barbas son muy utilizadas en meteorología sinóptica y aeronáutica. En la figura² 2.4 se muestra una representación gráfica del viento en 850 hPa utilizadas en la meteorología aeronáutica. Nótese en la elipse roja el viento de 40 knot del sector oeste en el norte de la Patagonia, mientras que en la zona central del país prevalecen vientos del sector norte en torno a los 10 y 15 knot.

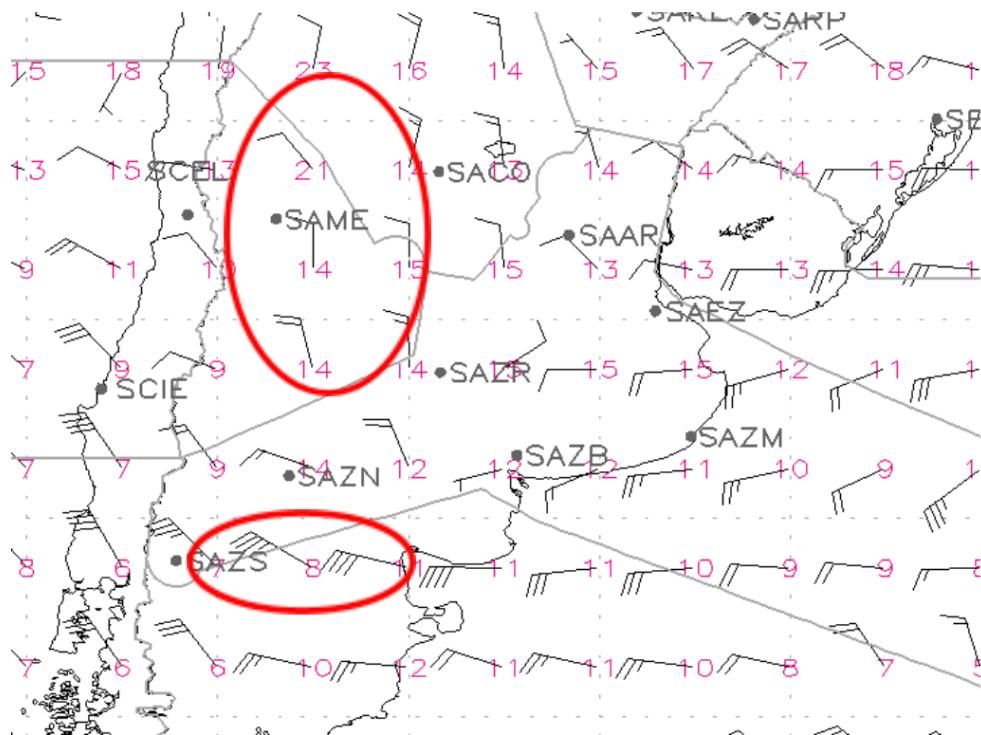


Figura 2.4: Representación del viento en 850 hPa para el día 14/4/2016 a las 12 UTC en la región central de Argentina.

Las **Líneas de corriente** son la representación gráfica más utilizada para analizar los factores de la circulación a gran escala. Una línea de corriente se define como la curva cuya tangente en cualquiera de sus puntos tiene la dirección de la velocidad del aire. Se utilizan en general en niveles altos, así como también en superficie en zonas tropicales. Un ejemplo de esta representación se puede apreciar en la figura³ 2.5. En esta figura, se indica que un sistema de alta presión se desplazó al sur de la Patagonia argentina, lo que produjo el ingreso de aire templado a la península antártica, junto con condiciones de subsidencia que reforzaron aún más el efecto de secamiento y calentamiento del aire al descender hacia la superficie.

²Imagen extraída del curso de observadores <https://crf.smn.gob.ar/>

³Imagen extraída de la página del SMN <https://www.smn.gob.ar>

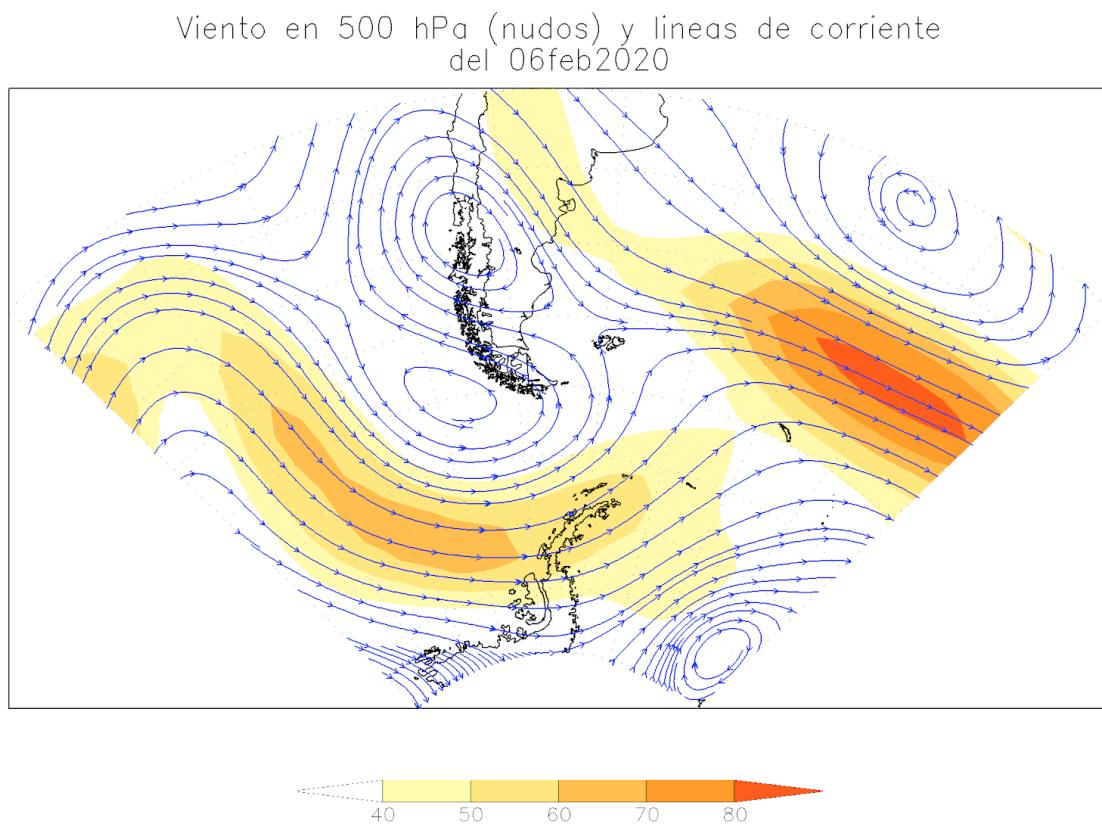


Figura 2.5: Viento (nudos) y líneas de corriente en 500hPa.

Si fuera necesario hablar de las fuerzas que actúan sobre el movimiento del aire

2.2. Instrumentos de medición del viento

Aca dar el detalle de los manuales de Vaisala y DeltaOHM, como se mide el viento, sus partes y como se debe alinear en la instalacion, sacar todo de los manuales, mencionar por arriba la coperola, el anemo de hilo caliente y el laserdopplerlidar.

2.3. Túnel de viento

aca poner el modelo del patron utilizado en el tunel de viento

poner como mide viento el anemotro de ultrasonido con un esquema del manual del deltaOHM

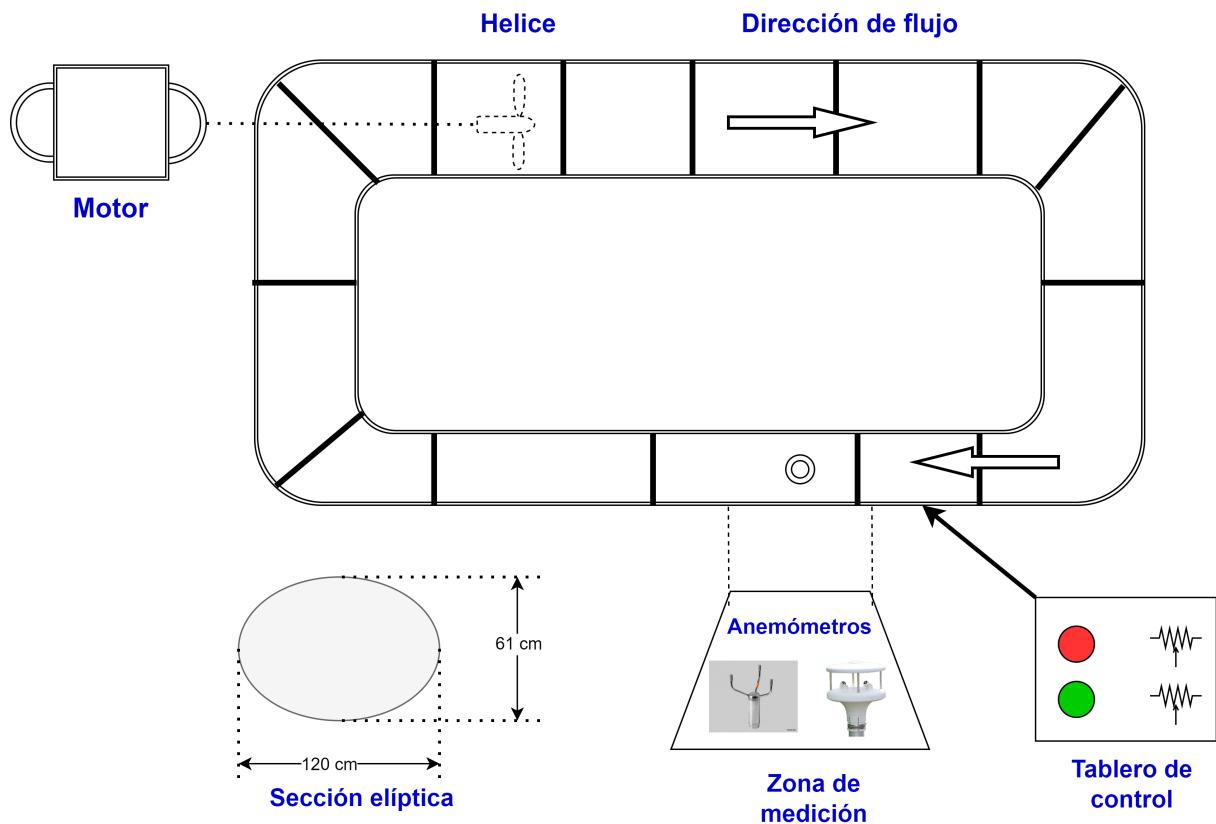


Figura 2.6: Túnel de viento.

poner foto real del tunel de viento, zona de medicion, vista superior y del motor.

Capítulo 3

Cálculo de incertidumbre

3.1. Incertidumbre de la medición

Definir algunas cosas como incertidumbre, y los tipos A y B

Para obtener una incertidumbre real del mensurando se debe realizar un presupuesto de incertidumbre (*uncertainty budget*). Un presupuesto de incertidumbre debe incluir el modelo de medición, las estimaciones y las incertidumbres de medición asociadas con las cantidades en el modelo de medición, las covarianzas, el tipo de funciones de densidad de probabilidad aplicadas, los grados de libertad, el tipo de evaluación de la incertidumbre de medición y cualquier factor de cobertura. Además debe incluir la incertidumbre de calibración del patrón, y las propiedades de los instrumentos patrón y bajo calibración que se obtienen de su hoja de datos o manual, a saber, deriva, resolución, tiempo constante vs. fenómeno observado, repetibilidad, estabilidad

3.2. Trazabilidad de las mediciones

Explicar la trazabilidad a partir de una diagrama.

3.3. Fuentes de incertidumbre

Aca pondremos todas las fuentes de incertidumbre que consideraremos para un anemómetro adentro de un tunel de viento en particular

3.4. Modelo teórico de medición

El modelo de medición utilizado para calcular la incertidumbre se muestra en la ecuación 3.1. Este modelo se basa en una comparación, donde se obtiene la corrección del mensurando bajo

calibración a partir de la diferencia de medición en un punto determinado y bajo condiciones ambientales controladas, entre un patrón calibrado y el instrumento bajo calibración (IBC).

$$CV_{IBC} = \overbrace{V_p + CV_p}^{V_{ref}} - V_{IBC} \quad (3.1)$$

donde se define:

- V_{IBC} representa el valor de viento medido con el anemómetro bajo calibración.
- CV_{IBC} representa la corrección del anemómetro bajo calibración.
- V_p representa el valor de viento medido con el anemómetro patrón.
- CV_p representa la corrección de viento del anemómetro patrón, obtenida a partir de su certificado de calibración.

A partir de la ecuación 3.1 y tomando en cuenta las fuentes de incertidumbre, definidas en la tabla ?? se realiza el calculo de la incertidumbre combinada de la corrección, como la suma de los cuadrados de las incertidumbres estándar de cada fuente, como se muestra en la ecuación 3.2

$$u^2(CV_{IBC}) = u^2(V_{ref}) + u^2(V_{IBC}) + u^2(\text{túnel de Viento}) \quad (3.2)$$

donde se define:

- $u^2(CV_{IBC})$ es la incertidumbre estándar de la corrección del anemómetro bajo calibración.
- $u^2(V_{ref})$ representa la suma de todas las fuentes de incertidumbre estándar debidas al patrón de referencia.
- $u^2(V_{IBC})$ representa la suma de todas las fuentes de incertidumbre estándar debidas al anemómetro bajo calibración.
- $u^2(\text{túnel de Viento})$ es la contribución de incertidumbre estándar referida al túnel de viento, quien se encarga de generar condiciones de velocidad y dirección de viento.

Capítulo 4

Implementación del Datalogger

Este capítulo se centra en el diseño y la programación del datalogger, el cual está basado en la placa de desarrollo EDU-CIAA, con el agregado de un shield (poncho) específico para la aplicación. El sistema se describe en la Figura 4.1 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 las tensiones necesarias para el funcionamiento y control del túnel de viento. Además, se toman muestras de esta última tensión y de la fuente para monitorear por software el comportamiento del sistema.

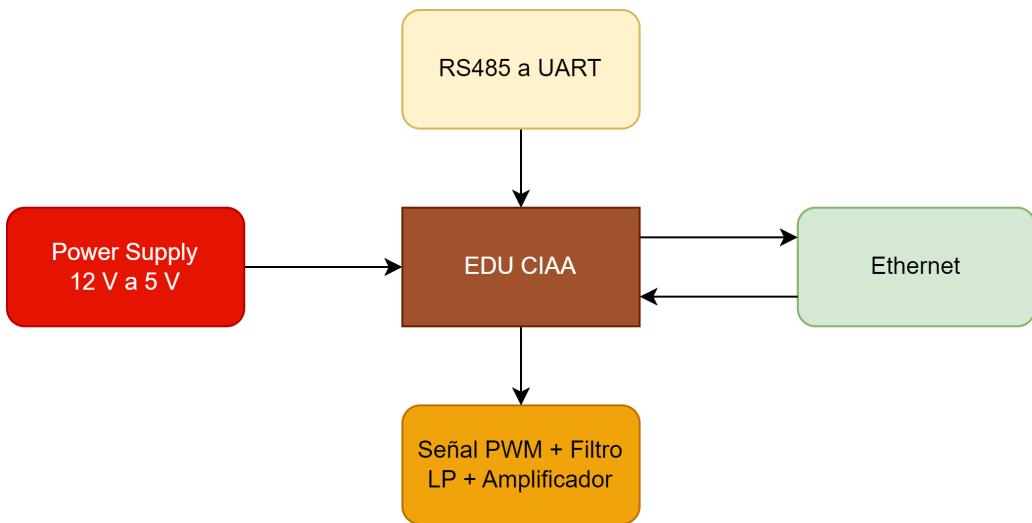


Figura 4.1: Diagrama en bloques del sistema electrónico.

4.1. 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 4.2, es una versión de bajo costo de la CIAA-NXP [8] pensada para la enseñanza universitaria, terciaria y secundaria.

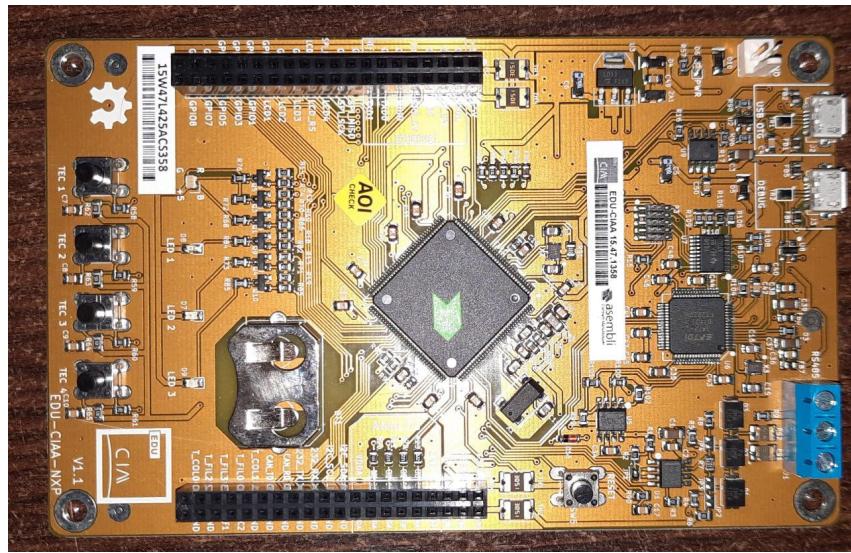


Figura 4.2: Placa de desarrollo EDU CIAA

En la Figura 4.3 se describe un diagrama en bloques de los periféricos de la placa de desarrollo.

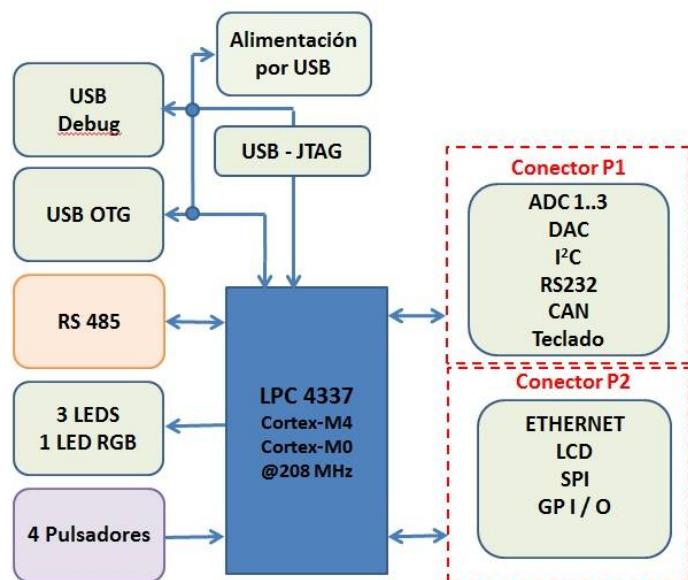


Figura 4.3: Diagrama en bloques de la EDU-CIAA

El SMN proporcionó los sensores de viento de ultrasonido. Se trabajó con los siguientes anemómetros: el modelo WMT700 de la marca VAISALA, Figura 4.4 , utilizado como anemómetro patrón, y el modelo HD51.3D de la marca Delta OHM, Figura 4.5, 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 4.4: Sensor Vaisala modelo WMT700



Figura 4.5: 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.

4.1.1. 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 4.6 . 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 4.6: Regulador de voltaje DC-DC basado en el chip LM2596.

En la Figura 4.7 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 preciso del sistema de alimentación. Además, se han agregado filtros de bypass a la alimentación del amplificador operacional LM358 para asegurar una operación

estable y sin ruido.

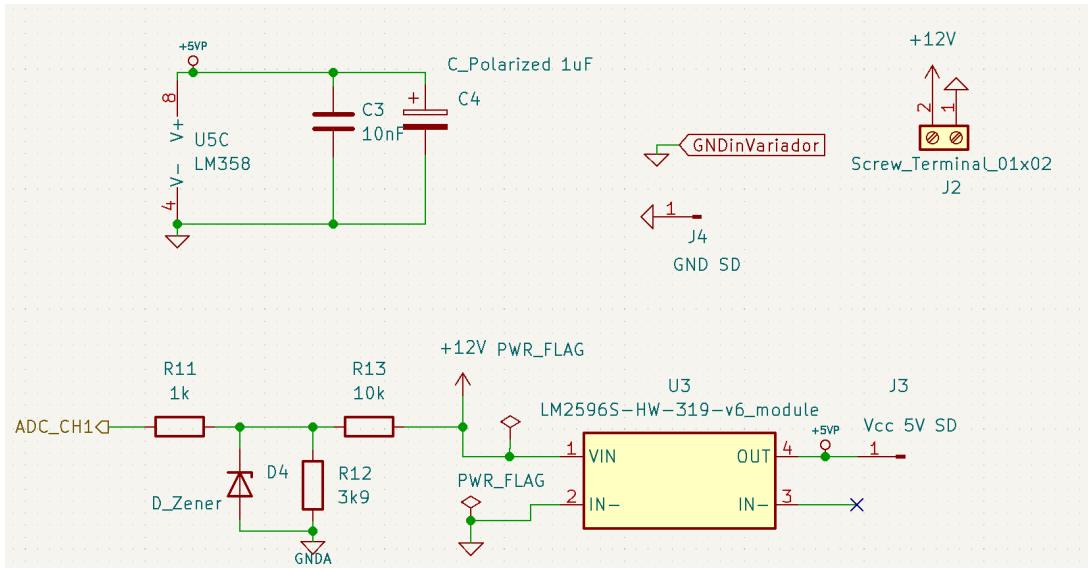


Figura 4.7: Esquemático del sistema de alimentación con módulo LM2596, diodo Zener, divisor resistivo y filtros de bypass para el LM358.

4.1.2. Módulo de adquisición de datos de viento

En la sección 2.2 del capítulo 2 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 [4], 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 voltaje entre A y B, ofreciendo alta inmunidad al ruido.

En la Figura 4.8 se muestra un esquema de conexión para el sensor DELTA OHM HD51.3D el cual fue utilizado para conectar dicho sensor al 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

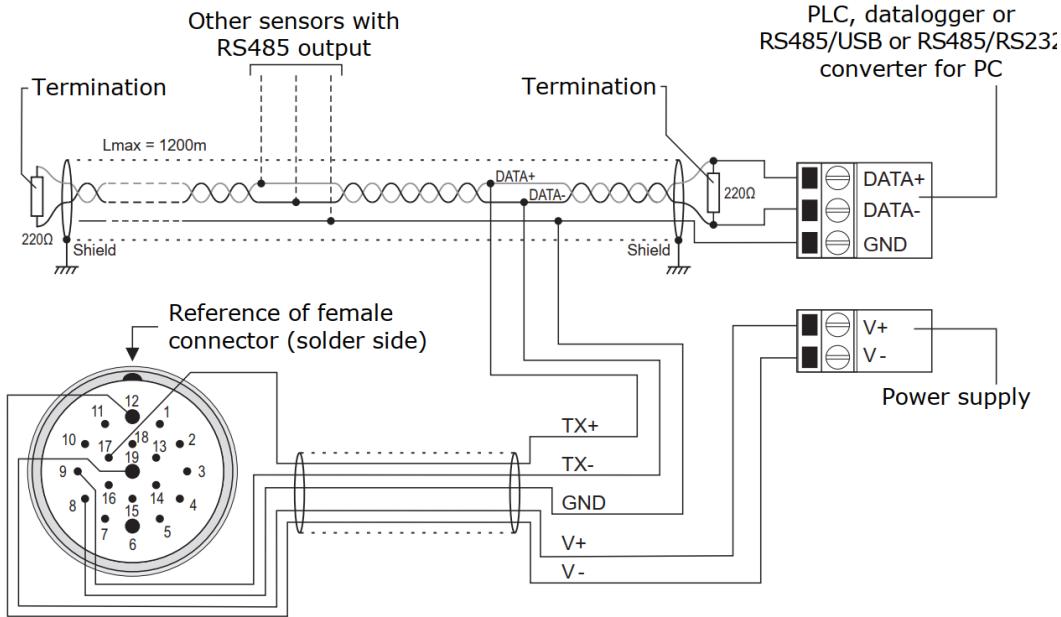


Figura 4.8: Esquema de conexión de datos y alimentación del sensor HD51.3D con un Datalogger o un PLC.

La placa EDU-CIAA cuenta con un módulo RS-485 integrado, como se muestra en la Figura 4.9. El anemómetro patrón WMT700 se conectó directamente a esta bornera. El sensor envía datos en modo ráfaga cada segundo, lo que permite una comunicación directa y eficiente sin necesidad de adaptadores adicionales.

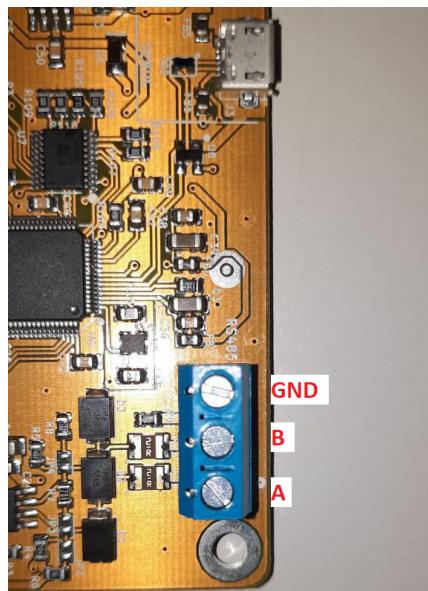


Figura 4.9: Puerto RS-485 integrado en la EDU-CIAA

Para conectar el anemómetro bajo calibración se utilizó el módulo MAX485, Figura 4.10, que convierte las señales RS-485 a niveles de la UART (TTL).

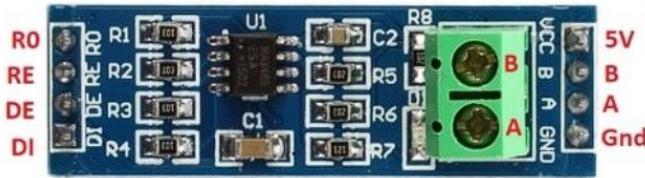


Figura 4.10: Módulo convertidor de RS485 a UART (TTL)

Como se muestra en la Figura 4.11, el módulo MAX485 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.

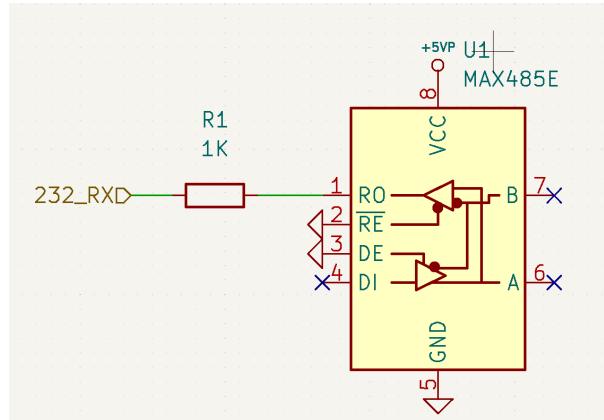


Figura 4.11: Esquemático del módulo MAX485 para la recepción de datos del sensor bajo calibración.

4.1.3. Módulo de comunicación Ethernet/SPI

Para comunicar el sistema embebido con distintos servidores, como el servidor WebSocket desarrollado en la sección 5.1.2, así como servidores NTP para sincronizar la fecha y hora, se utilizó el módulo W5100, Figura 4.12. 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.

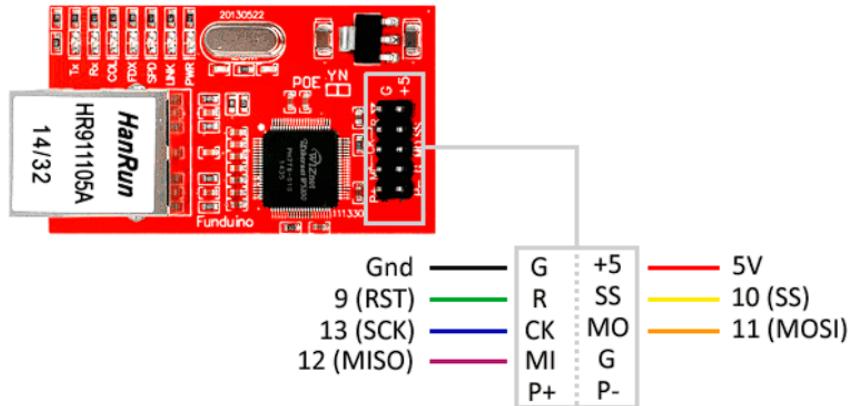


Figura 4.12: Módulo W5100 utilizado para realizar la conectividad a internet a través de la interfaz SPI del microcontrolador.

Como se muestra en la Figura 4.13, 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.

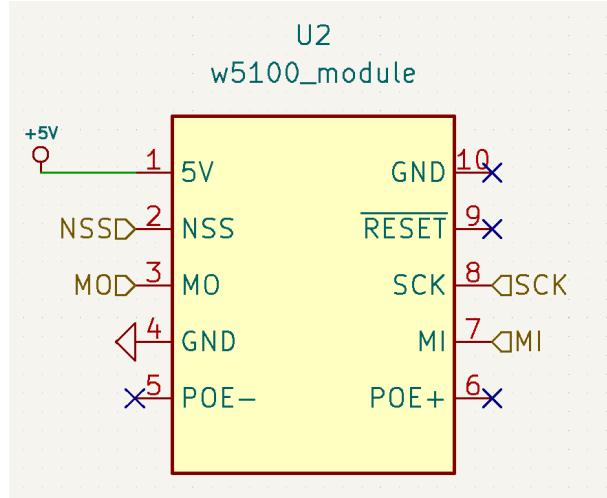


Figura 4.13: Esquemático del módulo W5100 para la comunicación bidireccional con distintos servidores.

4.1.4. 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 4.14 se muestra un esquema del tablero de control del 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. Además, cuenta con dos voltímetros de corriente continua, V1 y V2, que muestran el valor de tensión que se está aplicando al motor.

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 la fuente $VCC = 10\text{ V}$ del variador. Estos componentes, junto con los potenciómetros, forman un divisor de tensión variable que permite ajustar la tensión entre 0 V y 10 V, la cual se aplica al pin $VADC$.

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 $140\text{ }\mu\text{A}$. Cuando se ajusta el potenciómetro para obtener una $VADC = 5\text{ V}$, la corriente es de $200\text{ }\mu\text{A}$. 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 $500\text{ }\mu\text{A}$.

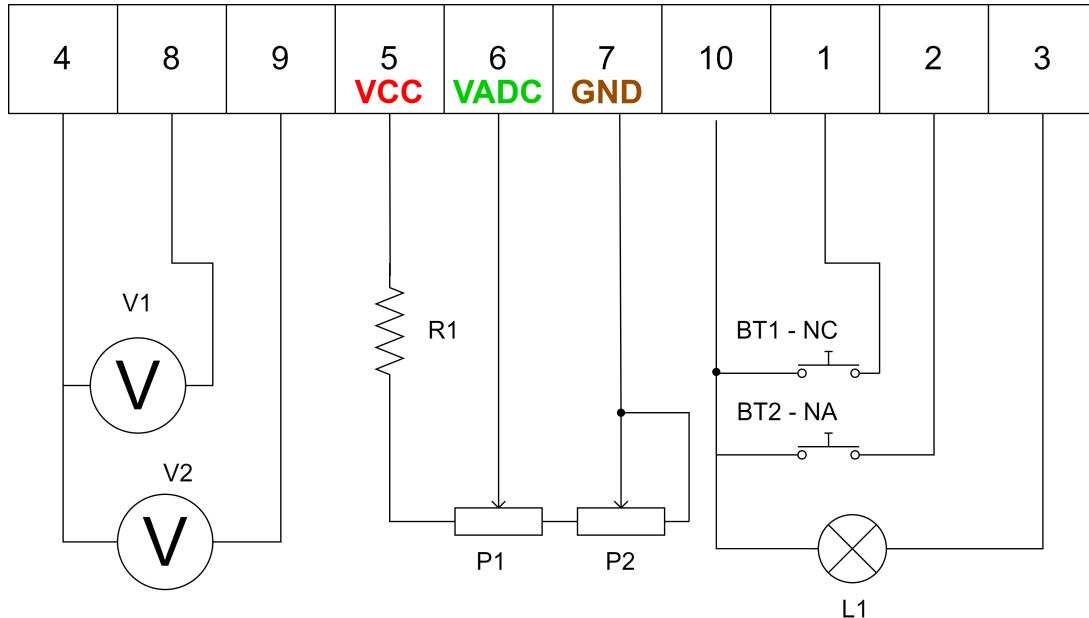


Figura 4.14: Esquemático circuital del tablero de control del tunel de viento.

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 4.1. Como se aprecia a medida que disminuimos la carga R , tanto la tensión, como la corriente que puede entregar la fuente, disminuye y la corriente máxima que puede entregar esta fuente es de 100 mA, con una tensión de 2,29 V. Tomando como referencia el consumo de corriente y la tensión que precisa el datalogger, esta fuente VCC del variador 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 4.1.1

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	859	100

Tabla 4.1: Mediciones de tensión y corriente de la fuente del variador para distintas resistencias de carga.

4.1.4.1. Simulación del circuito

En la Figura 4.15 se muestra el diseño del circuito del PWM [6], que reemplaza al divisor de tensión formado por los potenciómetros $P1$, $P2$ y la resistencia $R1$, mostrado en la Figura 4.14. 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. Con esta doble etapa de filtro pasa bajo, se obtiene una señal continua con menor rizado, como se muestra en la Figura 4.16 (señal verde y azul).

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. 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.

No obstante, para cuando se mejore la mecánica y transmisión del motor del túnel de viento, se dejó un resistor variable (Preset) $R3 = 10\text{k}\Omega$. Con este preset se cambia la ganancia del amplificador. Se lo ajustó para alcanzar 3,5 V a la salida del operacional, y podrá ser recalibrado para alcanzar mayores niveles de tensión, lo que genera, mayores velocidades de viento.

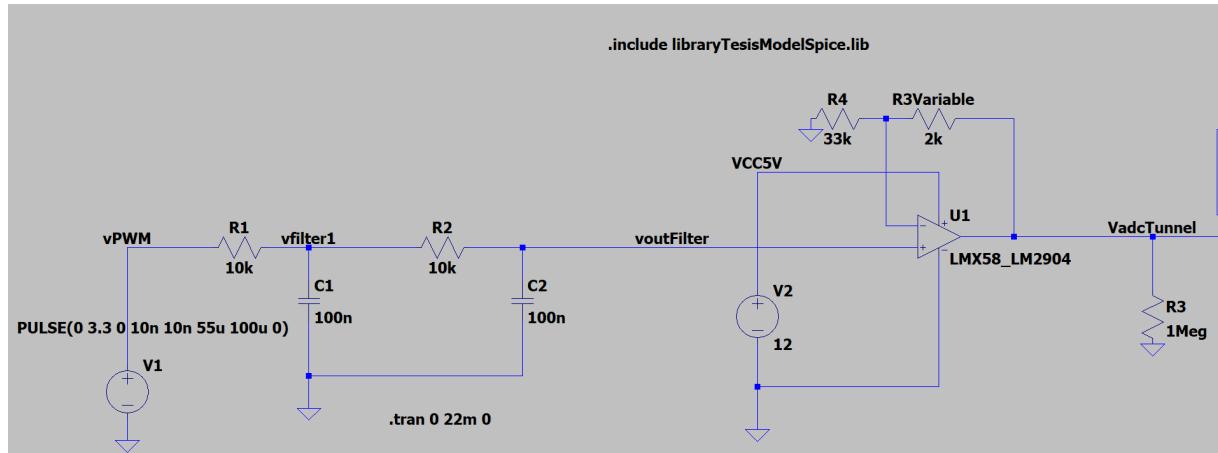


Figura 4.15: Circuito diseñado para obtener distintos niveles de tensión mediante la variación del ciclo de trabajo de una señal cuadrada (PWM).

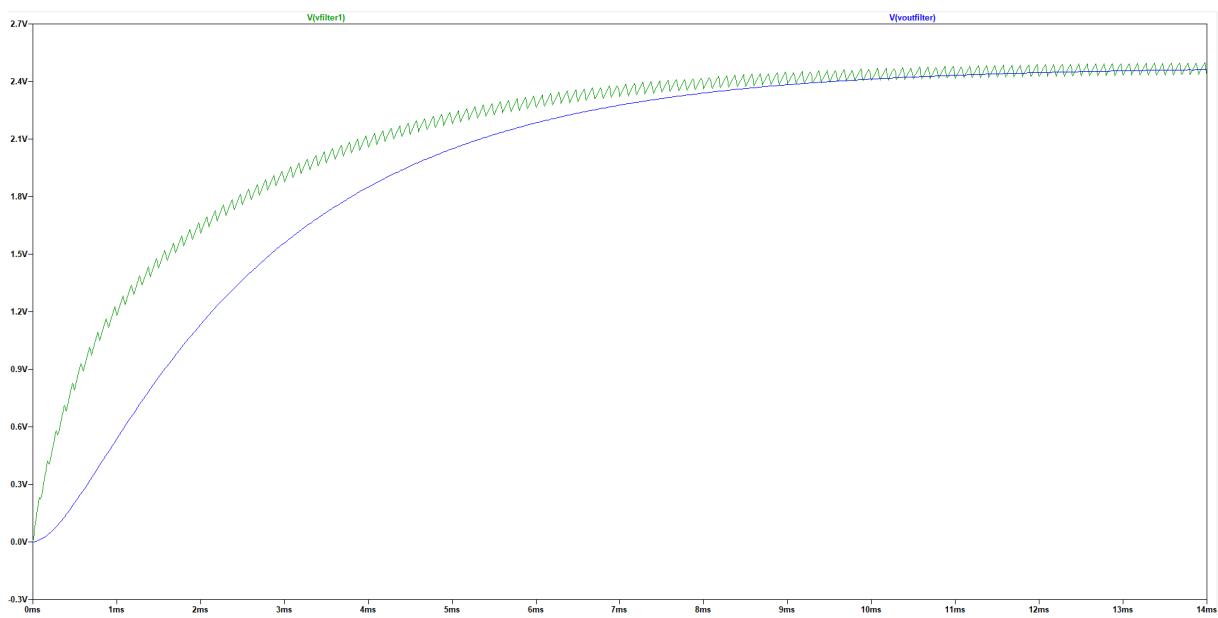


Figura 4.16: Circuito diseñado para obtener distintos niveles de tensión mediante la variación del ciclo de trabajo de una señal cuadrada (PWM).

En la Figura 4.17 se muestra el resultado de la simulación para un pulso vPWM (señal azul) de 100 μs de periodo (10 kHz) y una amplitud de 3,3 V con un ciclo de trabajo del 99 %.

La tensión vFilter1 (señal verde) en la primera etapa del filtro pasabajo tiene un tiempo de crecimiento corto, del orden de 10 ms, pero presenta mayor rizado. La segunda etapa del filtro genera la tensión voutFilter (señal roja), que tiene un mayor tiempo de crecimiento, del orden de 16 ms, pero con un rizado mucho menor. Esto demuestra que la adición de una segunda etapa de filtrado permite obtener un nivel de continua con un rizado significativamente reducido.

Finalmente, podemos observar la señal vadcTunel (señal celeste). Esta señal es estable y se conecta al pin *VADC* del variador de velocidad.

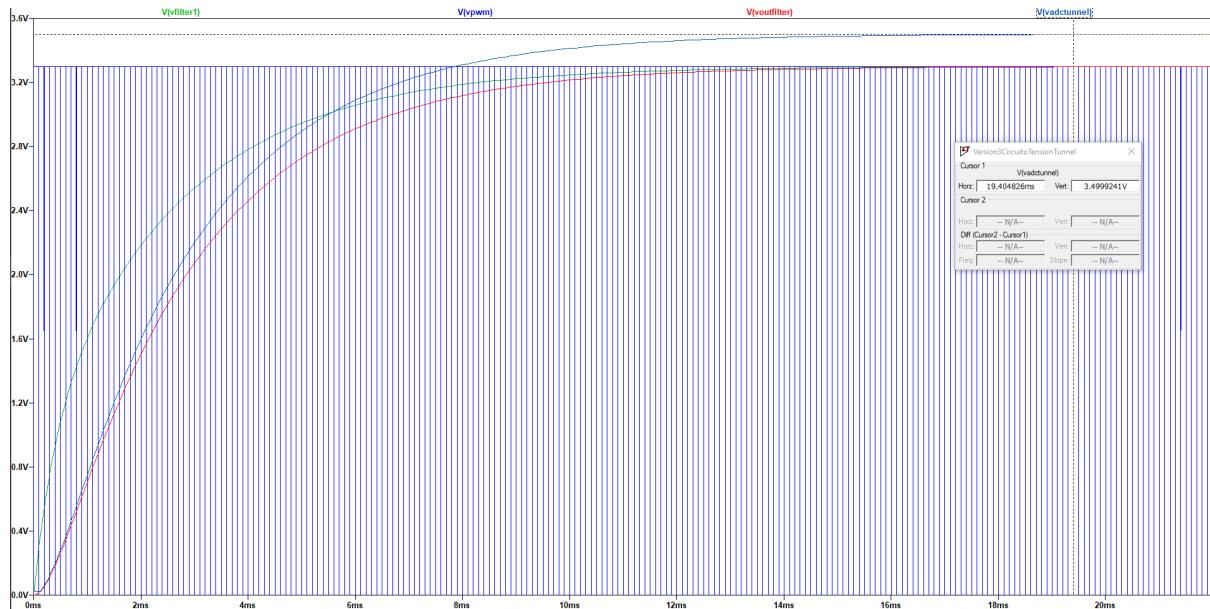


Figura 4.17: Señales simuladas para un ciclo de trabajo al 100 % de la señal de entrada.

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 4.18 y 4.19, que muestran los resultados para ciclos de trabajo del 75 % y 55 %, respectivamente.

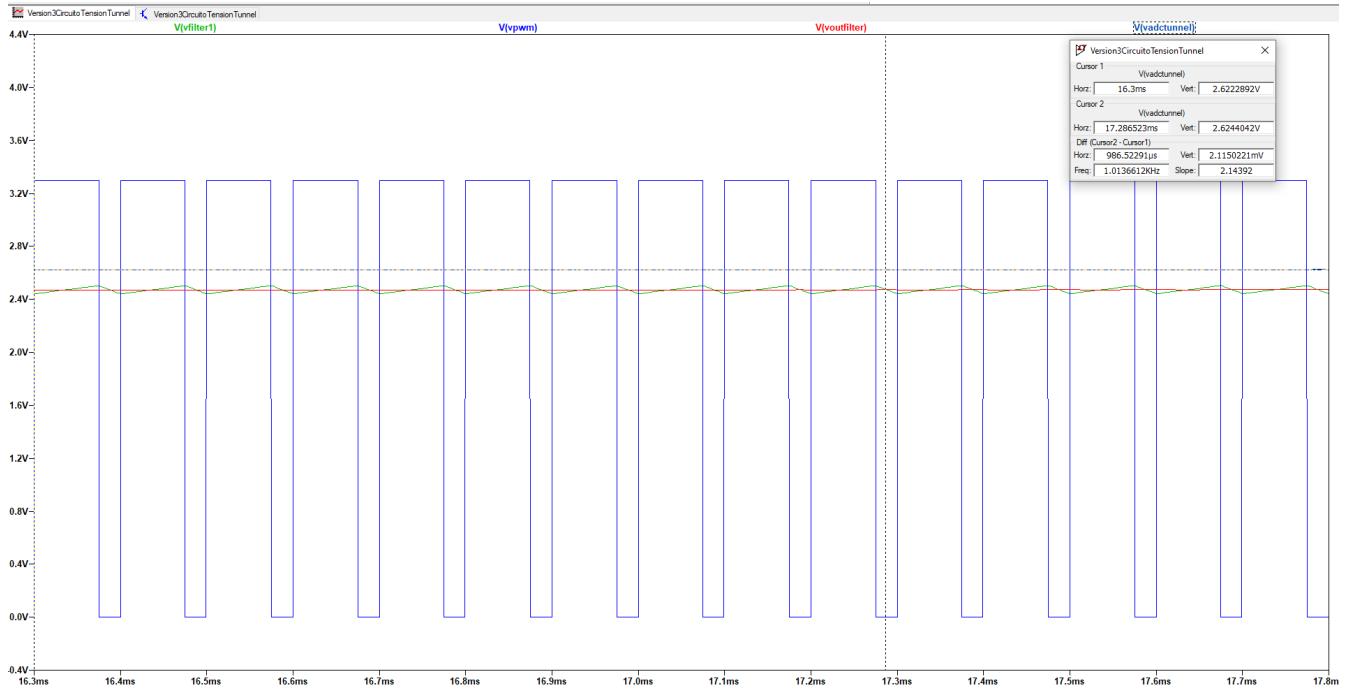


Figura 4.18: Señales simuladas para un ciclo de trabajo al 75 % de la señal de entrada.

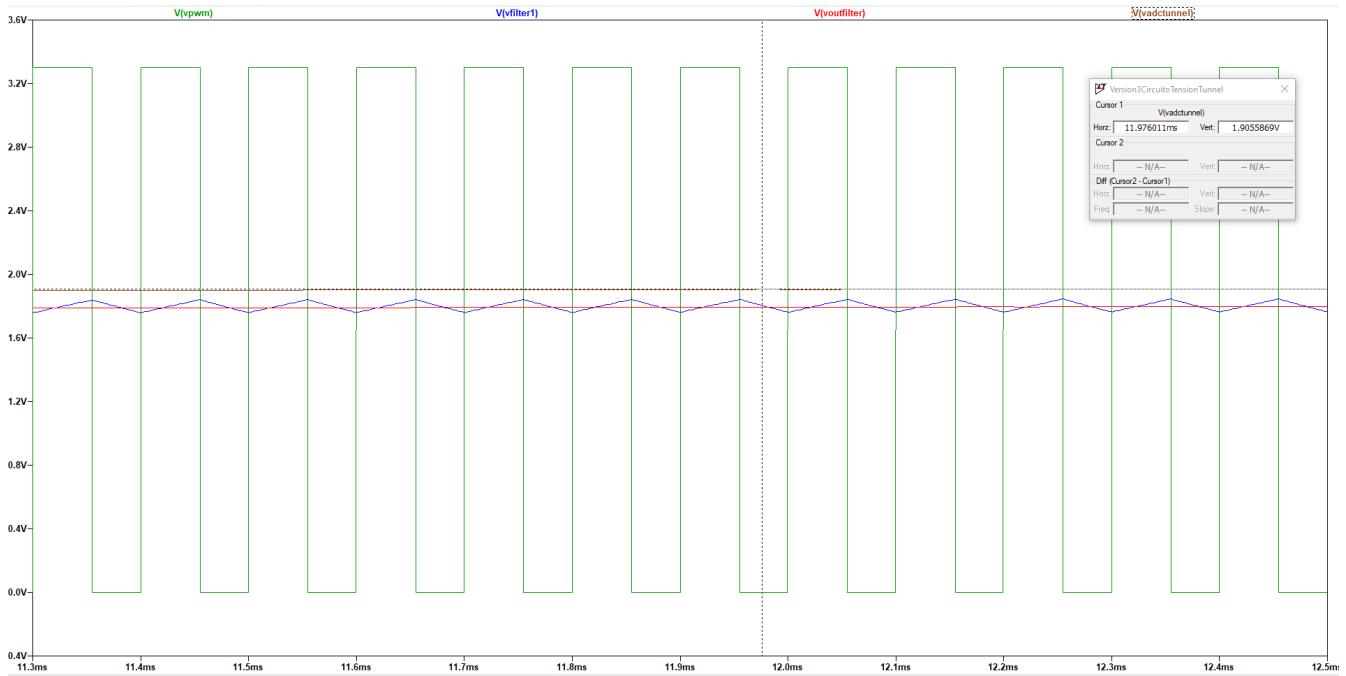


Figura 4.19: Señales simuladas para un ciclo de trabajo al 55 % de la señal de entrada.

4.1.4.2. Implementación del circuito

Se implementó el circuito de la Figura 4.20 en un protoboard de prueba con componentes Through-Hole Technology (THT), conectado a la EDU-CIAA, como se muestra en la Figura 4.21. Para comprobar y caracterizar el comportamiento del PWM, se utilizaron distintos valores

de ciclo de trabajo configurados manualmente en 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 de 10 kHz con amplitud de 3,3 V que sale de un pin digital de la placa de desarrollo y se conecta a la entrada al circuito PWM diseñado. En el canal 2 se midió la señal continua a la salida del amplificador operacional (ADCinVariador), ajustando el potenciómetro *RV1* del esquema de la Figura 4.20, para obtener 3,5 V cuando el ciclo de trabajo es 100 %.

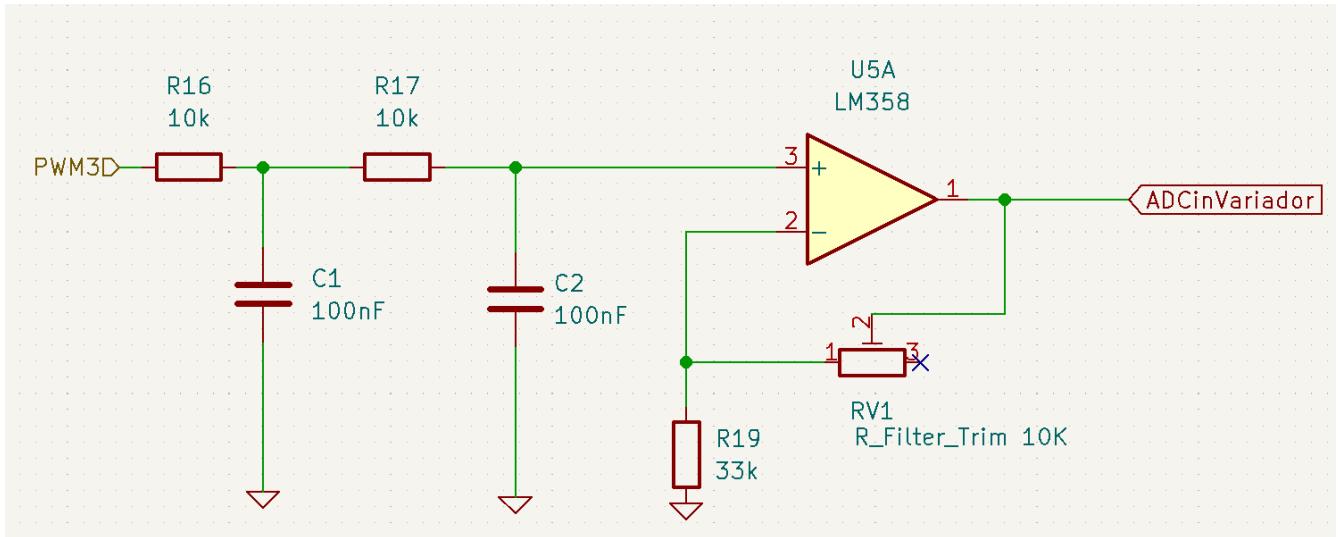


Figura 4.20: Esquemático del circuito PWM que controla la velocidad del túnel de viento por programación.

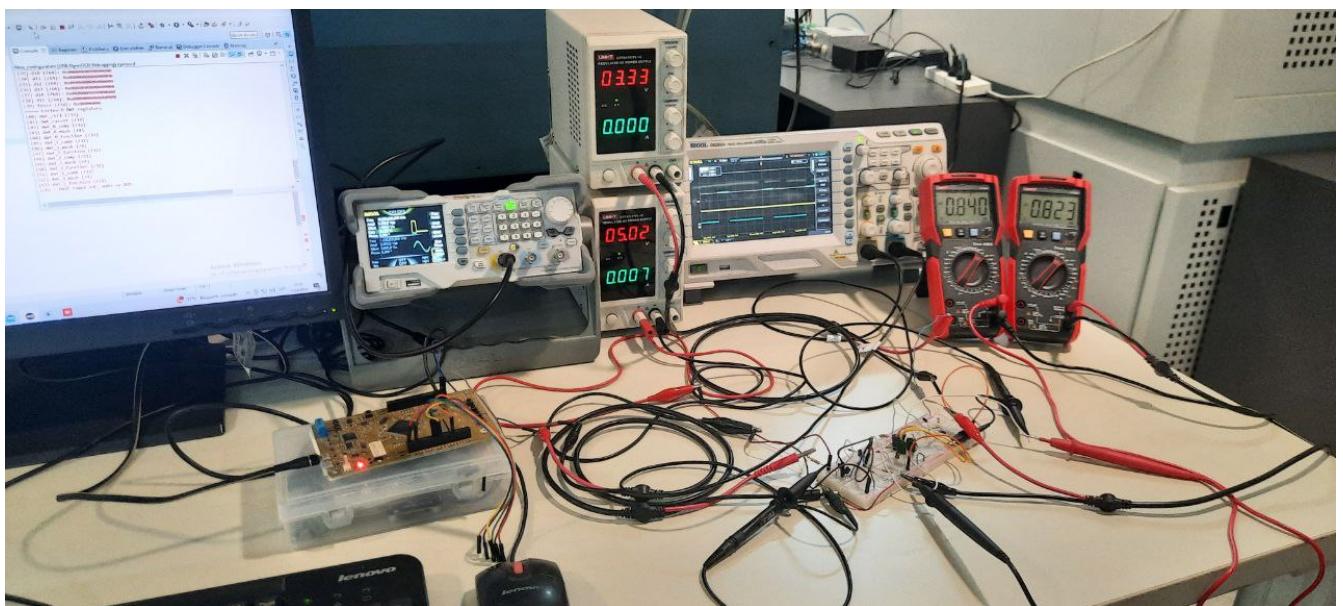


Figura 4.21: Banco de medición para caracterizar el circuito PWM.

Los resultados de las mediciones realizadas con el osciloscopio se muestran en las Figuras 4.22, 4.23, 4.24, 4.25 y 4.26, equivalente a ciclo de trabajo de 1%, 25%, 50%, 75% y 100% respectivamente.



Figura 4.22: 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.

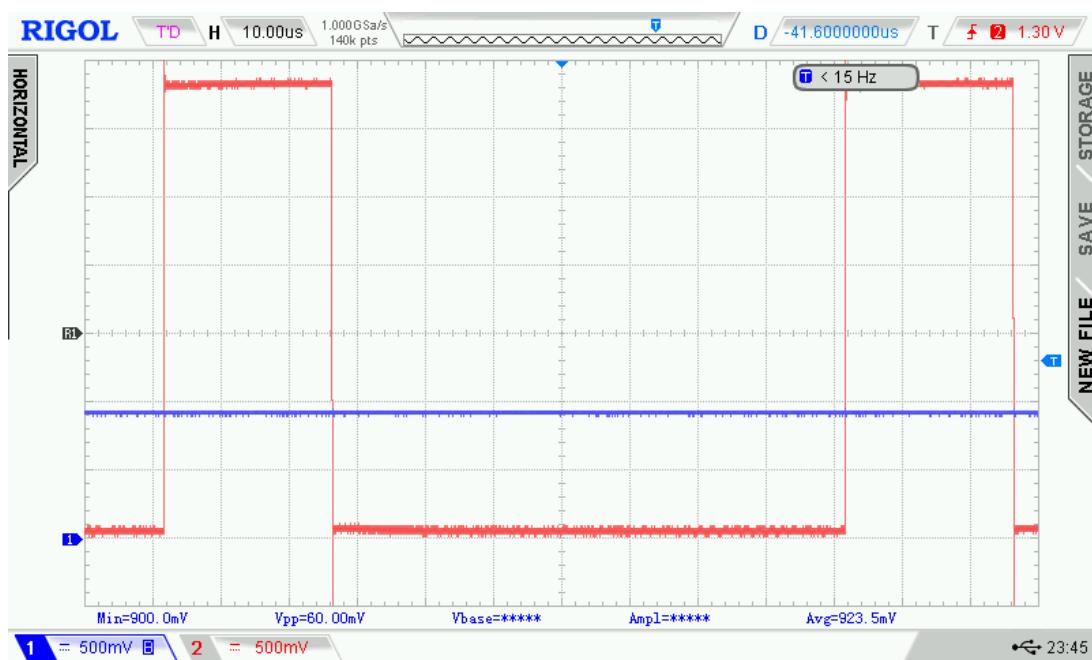


Figura 4.23: Medición de la señal PWM (rojo) con un ciclo de trabajo de (25 %) y la señal continua a la salida del amplificador operacional (azul) iguala 900 mV.

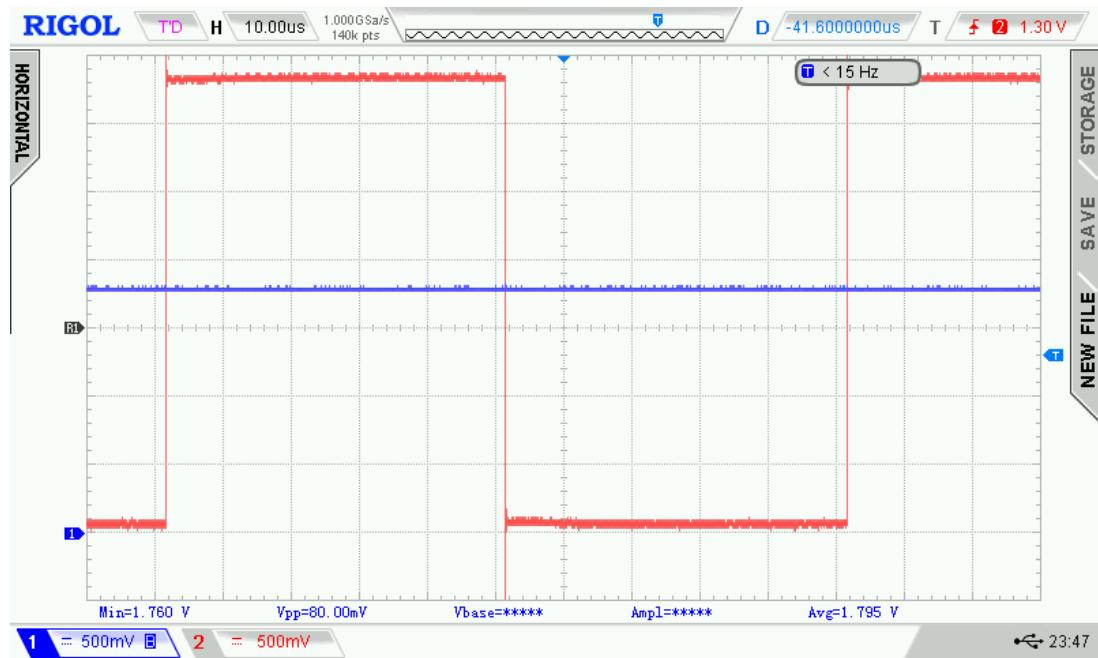


Figura 4.24: Medición de la señal PWM (rojo) con un ciclo de trabajo de (50 %) y la señal continua a la salida del amplificador operacional (azul) iguala 1,8 V.

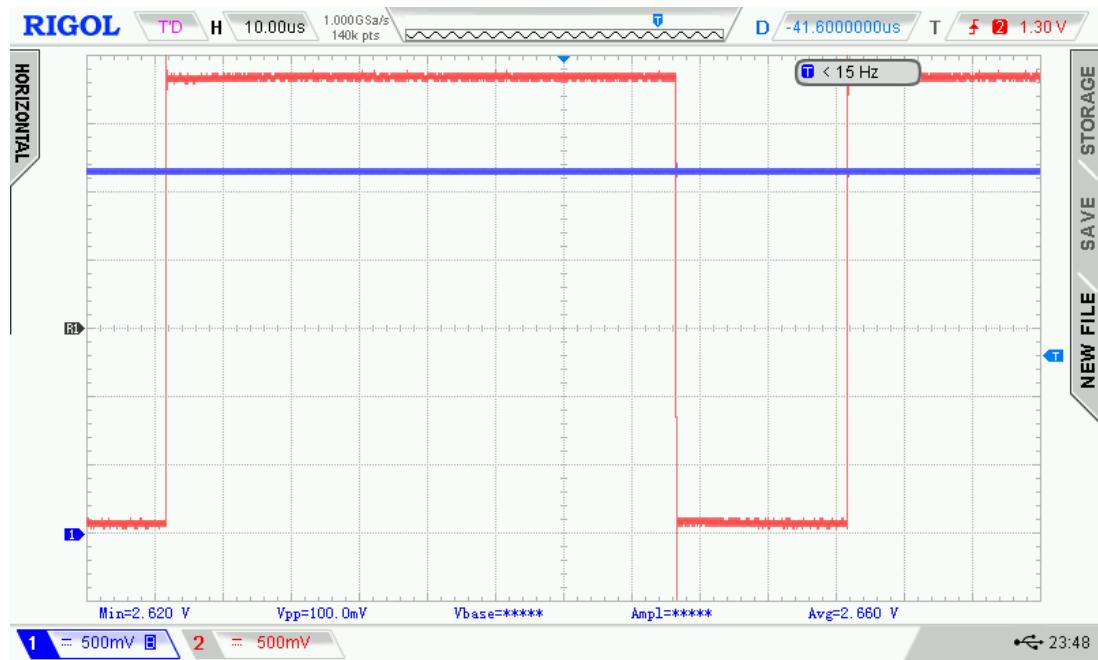


Figura 4.25: Medición de la señal PWM (rojo) con un ciclo de trabajo de (75 %) y la señal continua a la salida del amplificador operacional (azul) iguala 2,65 V.

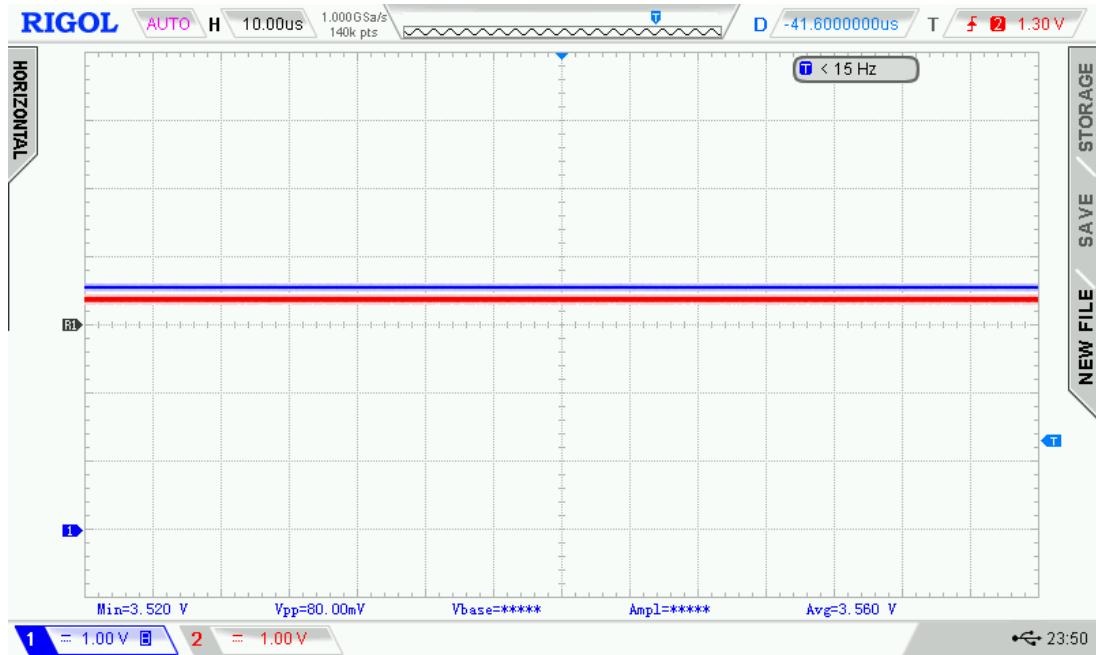


Figura 4.26: Medición de la señal PWM (rojo) con un ciclo de trabajo de (100 %) y la señal continua a la salida del amplificador operacional (azul) iguala 3,56 V.

Luego de caracterizar el circuito PWM, se lo llevó al túnel de viento y 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 tabla 4.2 se muestran los resultados de un barrido ascendente, y en la tabla 4.3, los resultados de un barrido descendente.

Ciclo de trabajo [0 - 255]	VadcTunnel [mV]	Viento [m s^{-1}]
0	4,2	1
1	16,6	1,1
2	30,3	1,2
31	427	4,5
63	864	7,8
94	1291	11,1
127	1745	14,7
158	2170	18
191	2621	21,5
222	3048	24,5
255	3500	26,3

Tabla 4.2: Mediciones para ciclos de trabajo en modo Ascendente, VadcTunnel y la velocidad del viento.

Ciclo de trabajo [0 - 255]	VadcTunnel [mV]	Viento [m s^{-1}]
255	3501	25,9
222	3048	24,5
191	2622	21,3
158	2170	18
127	1744	14,7
94	1291	11,1
63	865	7,8
31	426	4,5
2	30	1,3
1	16	1,1
0	4	1,0

Tabla 4.3: Mediciones para ciclos de trabajo en modo Descendente, VadcTunnel y la velocidad del 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, 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 4.3.4.

4.1.5. Circuito de adquisición de tensión del variador

En esta sección se diseñó 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.

4.1.5.1. Simulación del circuito

En la Figura 4.27 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.

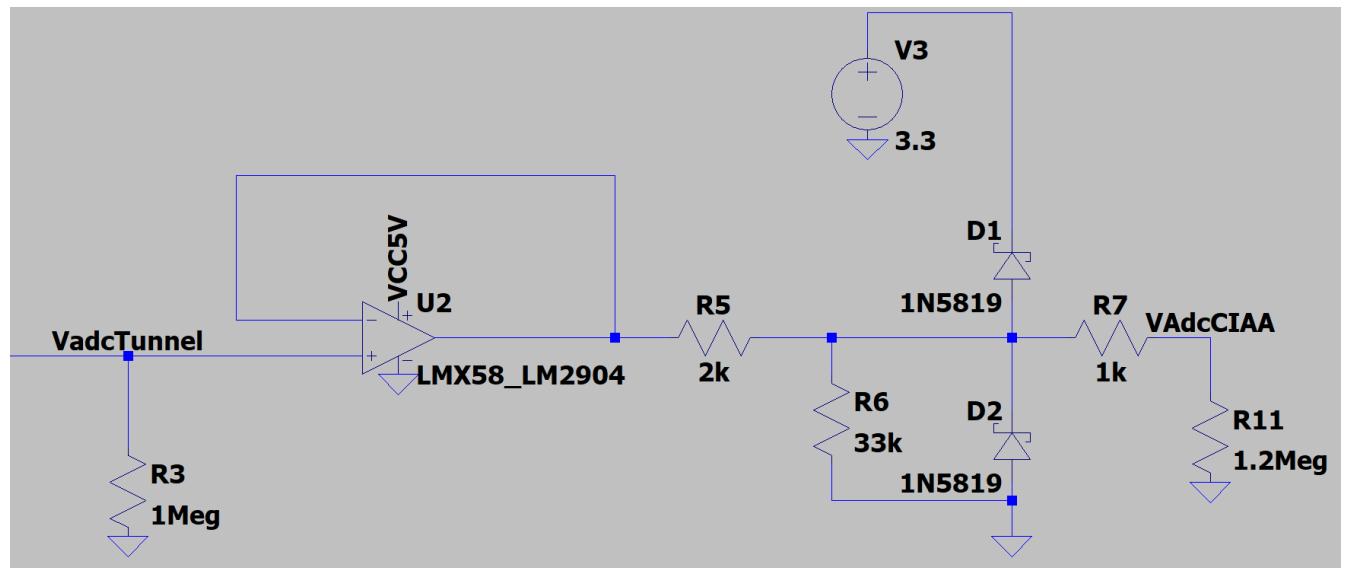


Figura 4.27: Circuito diseñado para tomar muestras de la señal que se suministra al variador de velocidad y se las envía a una canal analógico-digital de la EDU-CIAA.

Los resultados de la simulación del circuito descrito se muestran en la Figura 4.28 para un ciclo de trabajo del 100 % y en la Figura 4.29 para un ciclo de trabajo del 55 %. En ambos casos, se puede apreciar que la señal vadcCIAA (azul) está por debajo de la señal vadcTunel (verde), lo cual garantiza que no se superen los 3,3 V a la entrada del pin analógico-digital de la placa de

desarrollo.

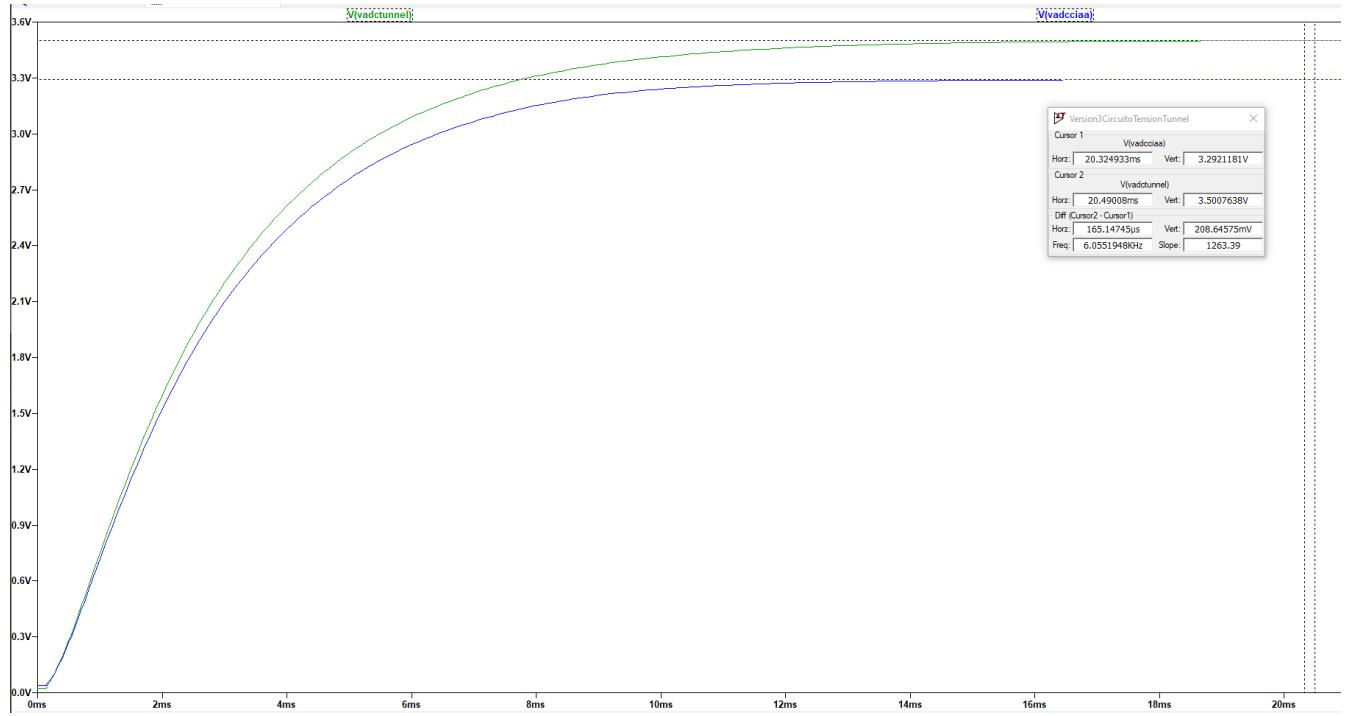


Figura 4.28: Señales simuladas a la salida del variador de velocidad y a la entrada del pin analógico-digital de la EDU-CIAA para un ciclo de trabajo del 100 %.

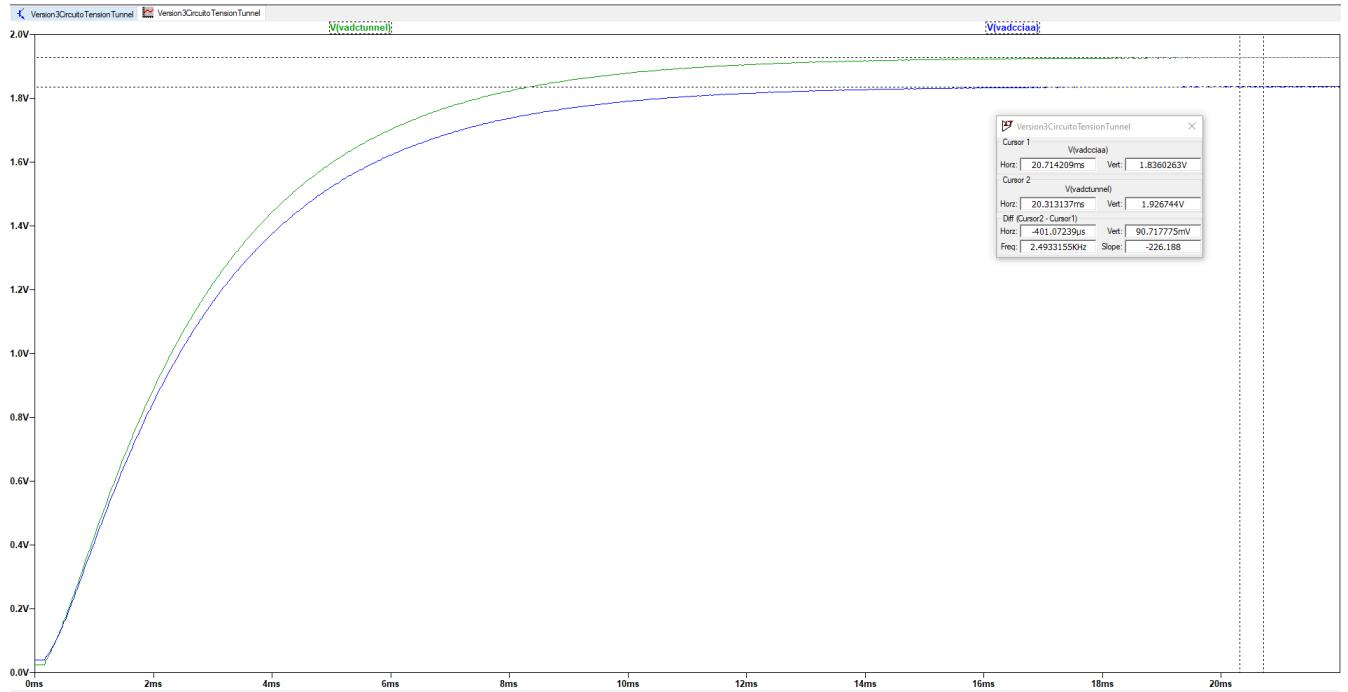


Figura 4.29: Señales simuladas a la salida del variador de velocidad y a la entrada del pin analógico-digital de la EDU-CIAA para un ciclo de trabajo del 55 %.

4.1.5.2. Implementación del circuito

Se implementó el circuito de la Figura 4.30. Para ajustar los niveles de tensión de 3,5 V a 3,3 V, se agregó en el divisor de tensión, un resistor variable (trimmer) de $RV2 = 10\text{ k}\Omega$. Luego se conectó los diodos a la fuente de 3,3 V de la EDU-CIAA y se agregó una resistencia limitadora de tensión de $R18 = 1\text{ k}\Omega$, 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.

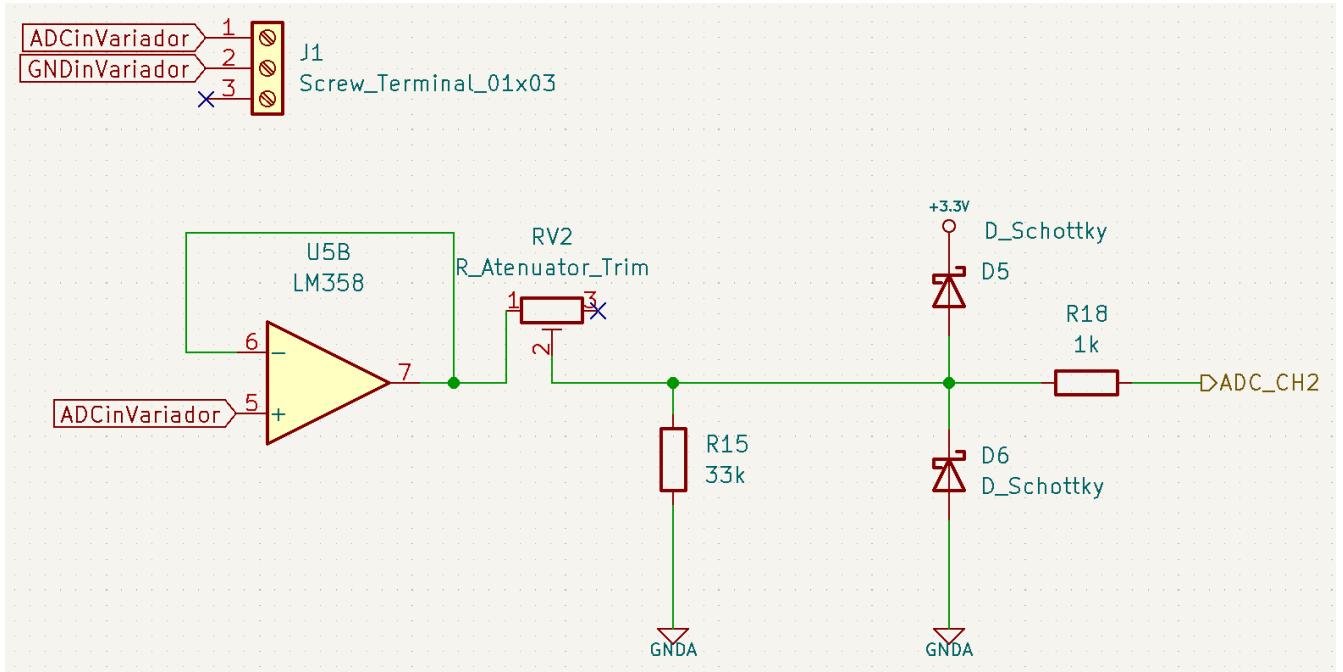


Figura 4.30: Esquemático de la implementación del circuito que toma muestras de la señal continua que ingresa al variador de velocidad.

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 tablas 4.4 para un barrido ascendente y 4.5 para un barrido descendente.

Ciclo de trabajo [0 - 255]	VadcTunnel [mV]	VadcCIAA [mV]
0	4,2	20
1	16,6	16,7
2	30,3	37,3
31	427	421,3
63	864	856
94	1291	1280
127	1745	1725
158	2170	2146
191	2621	2593
222	3048	3013
255	3500	3364

Tabla 4.4: 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.

Ciclo de trabajo [0 - 255]	VadcTunnel [mV]	VadcCIAA [mV]
255	3501	3364
222	3048	3012
191	2622	2593
158	2170	2146
127	1744	1725
94	1291	1277
63	865	855
31	426	422
2	30	38
1	16	27
0	4	21

Tabla 4.5: 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.

4.1.6. Circuito de LEDs indicadores

Se diseñó el circuito de la Figura 4.31 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.

El diseño de cada circuito incluye un transistor NPN y dos resistencias, conectados a un diodo LED. La señal de entrada llega al resistor $R_4 = 1\text{k}\Omega$, el cual está conectado a la base del transistor. El colector del transistor está conectado a una fuente de 5 V, mientras que el emisor está conectado a una resistencia limitadora de corriente $R_5 = 39\Omega$ y luego al ánodo del LED, con su cátodo a masa. Ambos circuitos son prácticamente idénticos, diferenciándose únicamente en los colores de los LEDs y los canales RS-485 a los que están conectados.

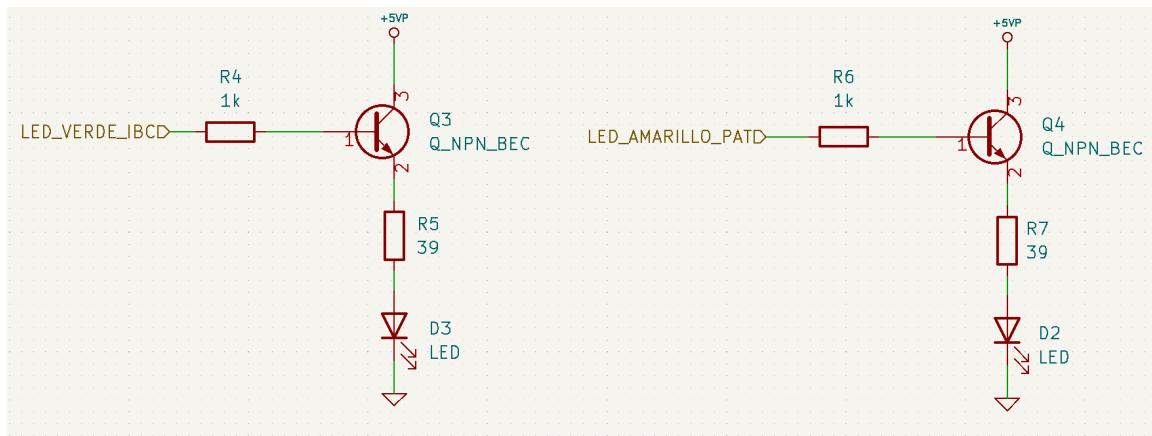


Figura 4.31: Esquemático de los circuitos que encienden y apagan dos LEDs, verde y amarillo, cada vez que se recibe un dato de los anemómetros, tanto del patrón como del que está bajo calibración.

En La Figura 4.32 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 5.1.2. 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.

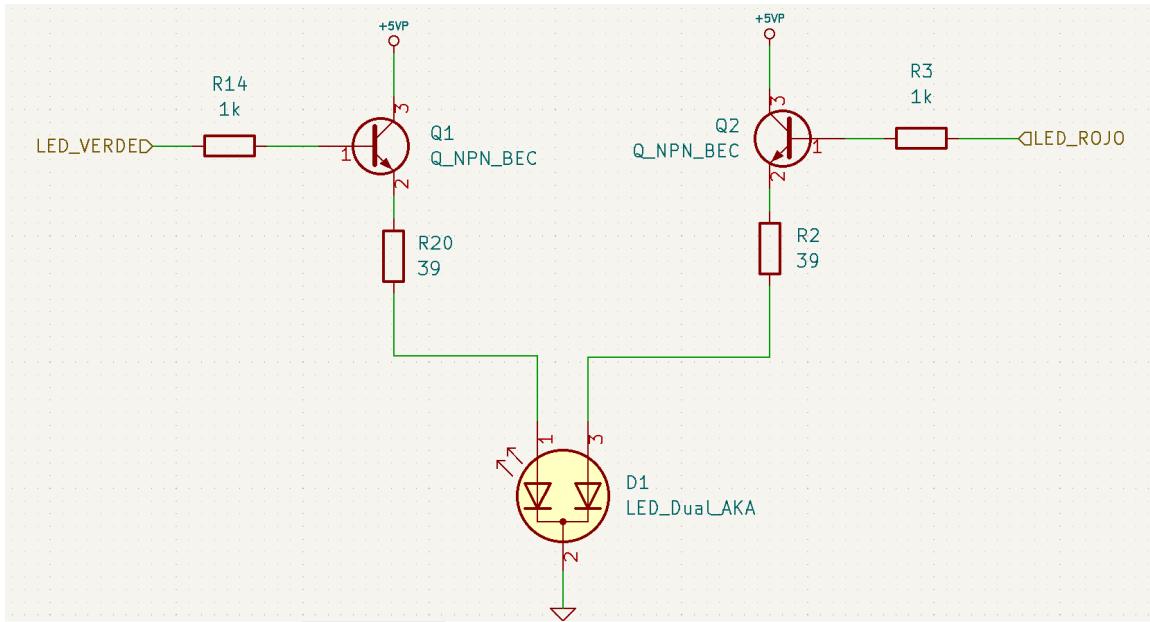


Figura 4.32: Esquemático del circuito de un LED dual que se enciende en rojo cuando el sistema no está conectado al servidor WebSocket y en verde cuando hay una conexión estable con el servidor.

4.2. 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 (poncho) que se conecta sobre la EDU-CIAA. Para ello, se trabajó con la lista de templates que ofrece el proyecto CIAA [5]. 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 4.33 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.

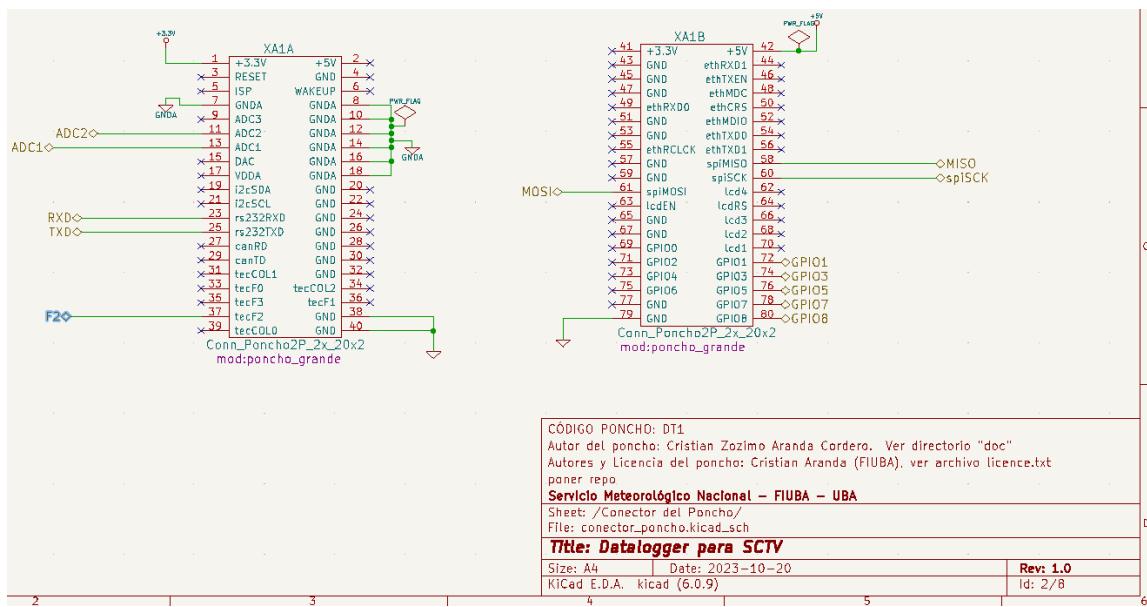


Figura 4.33: Template disponible para la realización de ponchos para la EDU-CIAA.

Por cada módulo, se agregó una hoja jerárquica que contiene el circuito esquemático como se indica en la Figura 4.34. 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 4.34: Diseño del esquemático que integra los módulos desarrollados.

En la Figura 4.35 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.

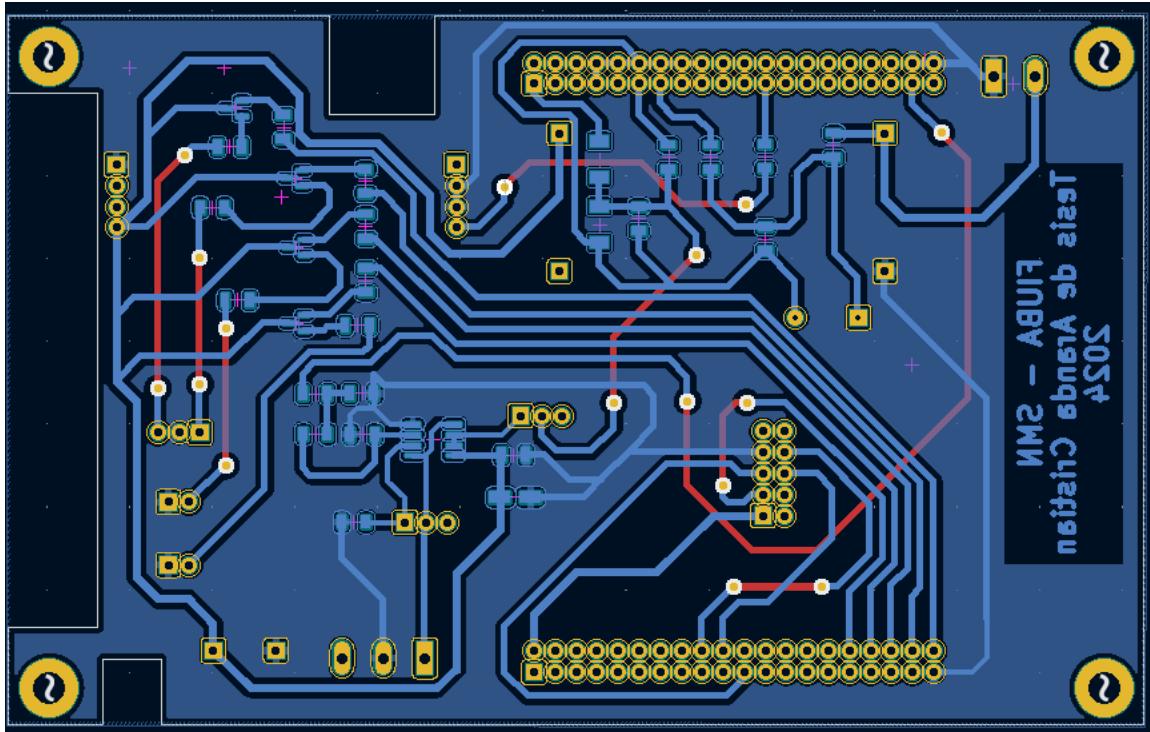


Figura 4.35: Diseño PCB del Shield (poncho) que contiene todos los módulos descritos en las secciones anteriores.

En la figura 4.36 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.

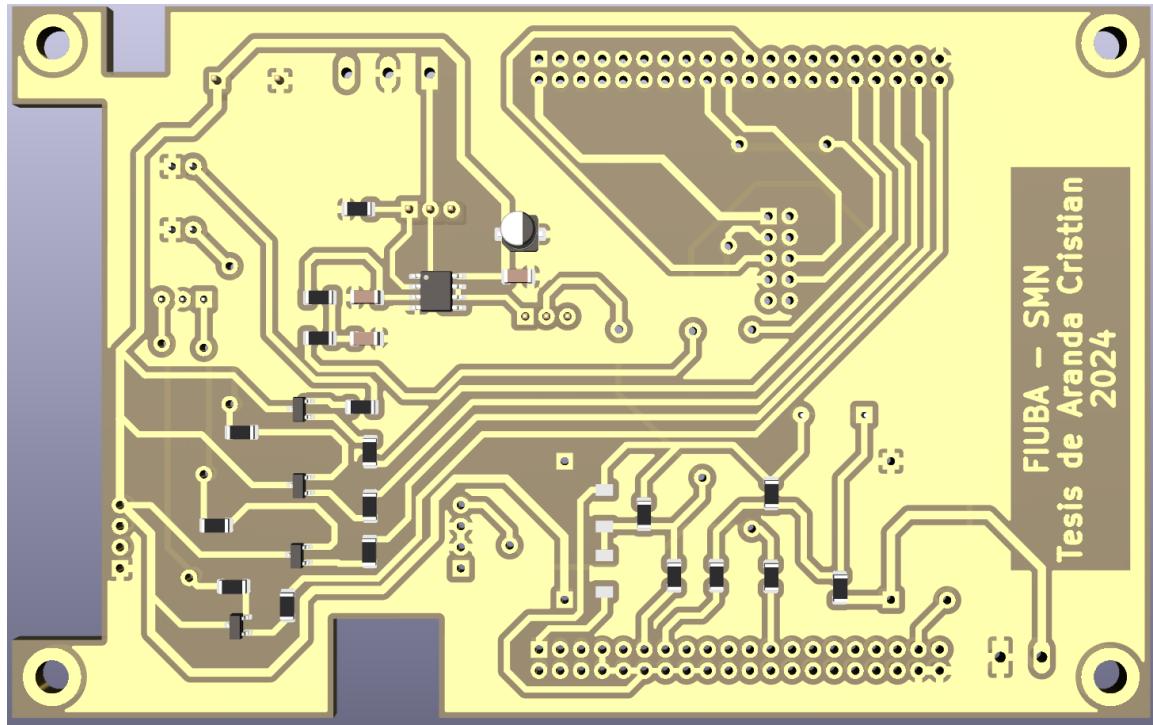


Figura 4.36: Vista en 3D de la capa BOTTOM.

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.

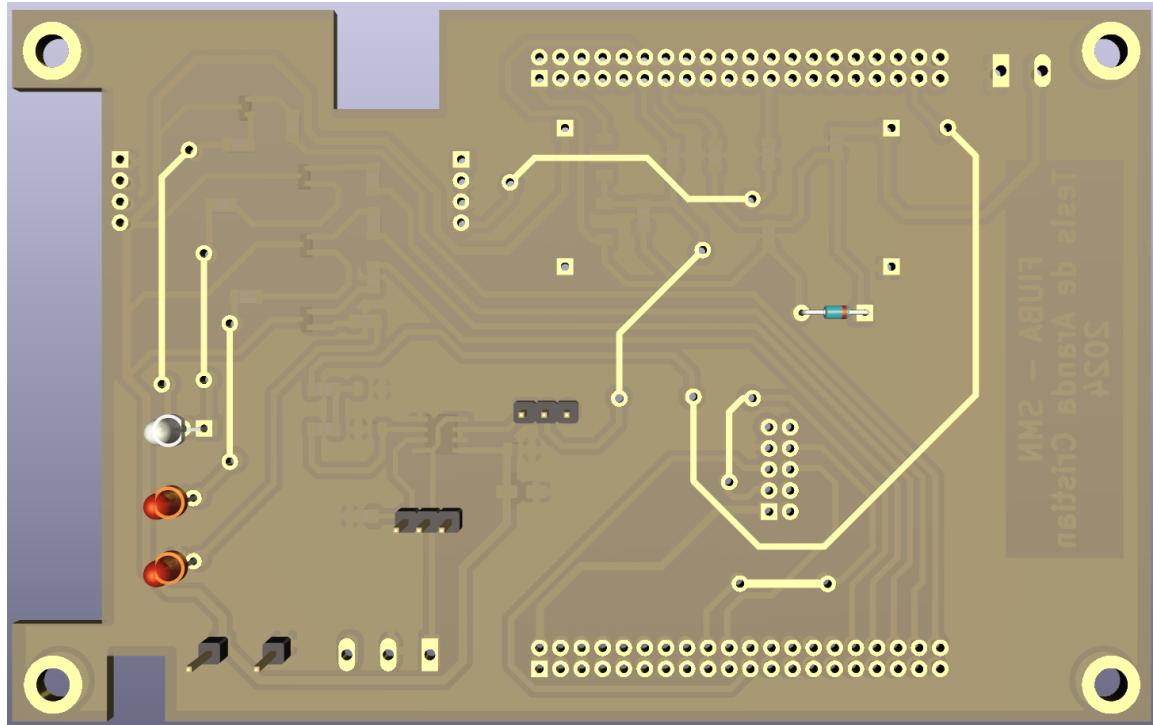


Figura 4.37: Vista en 3D de la capa TOP.

Para la construcción del *shield*, se seleccionó una placa simple FAS 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 sobrante. 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 FAS, 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. Las Figuras 4.38 y 4.39 ilustran el resultado final del montaje.

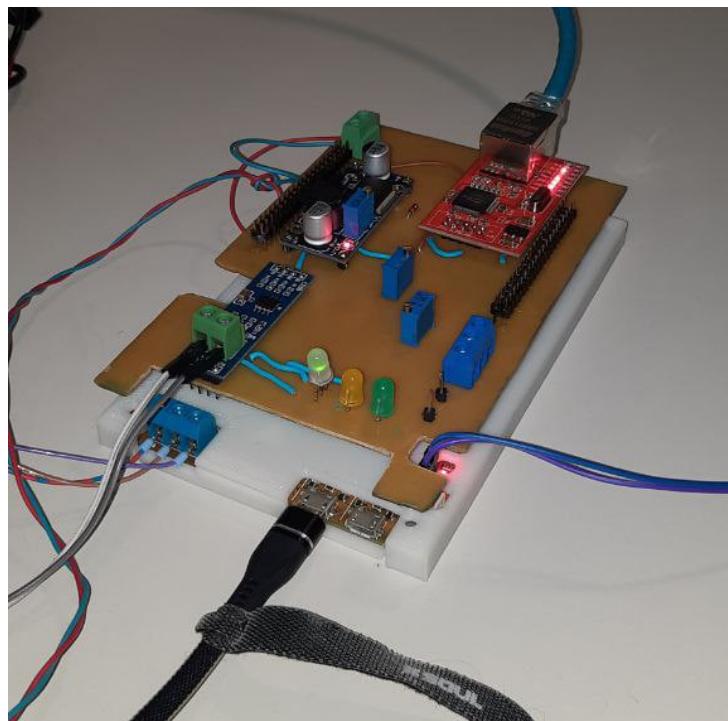


Figura 4.38: Shield conectado a la placa EDU-CIAA

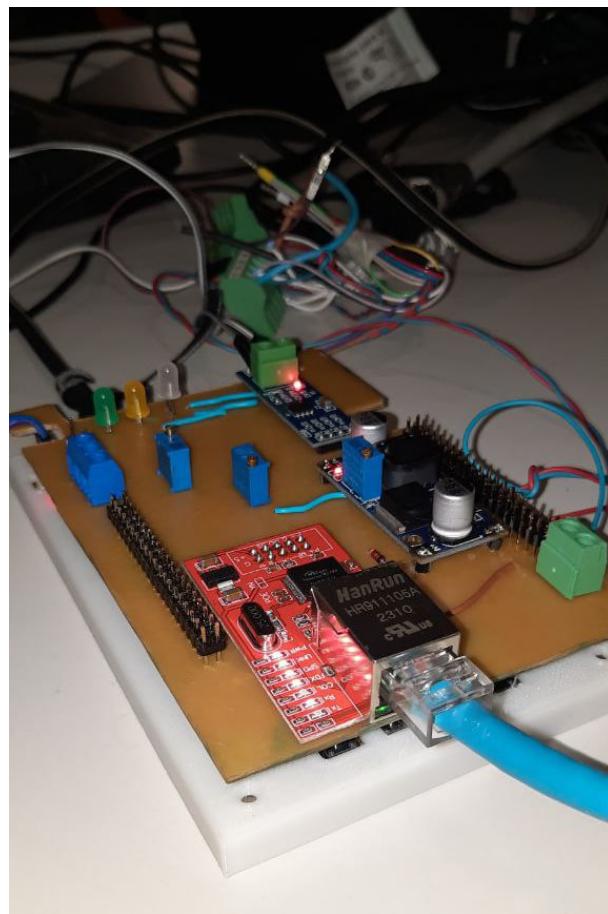


Figura 4.39: Otra vista del Shield diseñado para esta aplicación.

Como se indica en la Figura 4.40, se conectó el USB para programar y depurar la placa. Se conectó en la bornera azul de la EDU-CIAA el sensor patrón y en la bornera verde del módulo RS485 se conectó el anemómetro bajo calibración. En la parte superior de la figura está una bornera azul, ahí se conectaron los tres cables que vienen del variador de velocidad del túnel. En el lado izquierdo, se conectó un cable UTP que está conectado a la red local del SMN, y en la bornera verde que está abajo a la izquierda, se conectó la alimentación de 12 V.

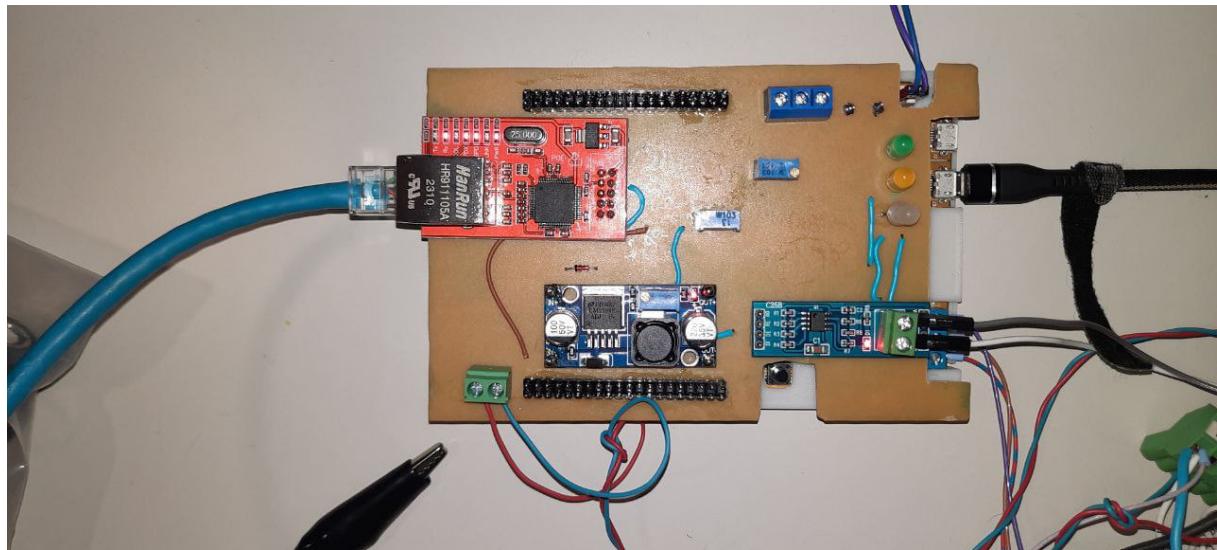


Figura 4.40: Vista superior de la placa Shield conectada a la EDU-CIAA

4.3. 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 4.41, 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.

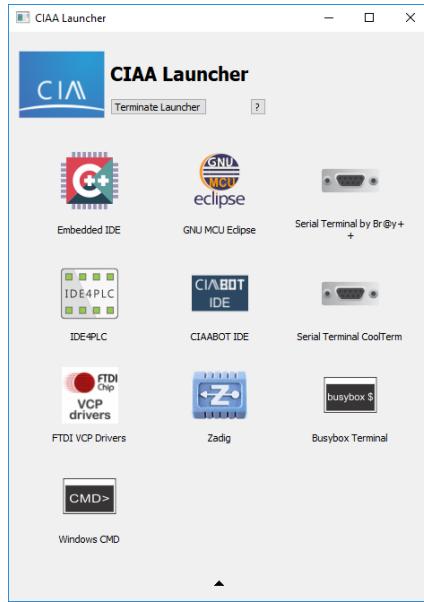


Figura 4.41: Lanzador de aplicaciones integradas para el desarrollo de firmware para la EDU-CIAA.

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 4.42.

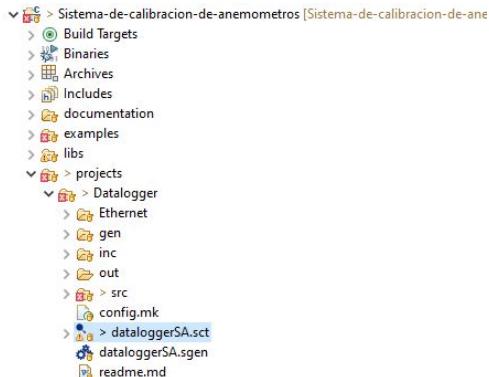


Figura 4.42: Estructura de carpetas, del proyecto Datalogger

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. 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 habilitan la comunicación por difusión, mientras que las transiciones condicionales, 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 4.6 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, condiciones	✗	✗	✓	✓

Tabla 4.6: Diferencias entre los tipos de máquinas de estados.

Se crearon doce regiones que trabajan de forma concurrente (de forma ortogonal), como se muestra en la Figura 4.43, y se comunican entre sí mediante eventos internos. 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.

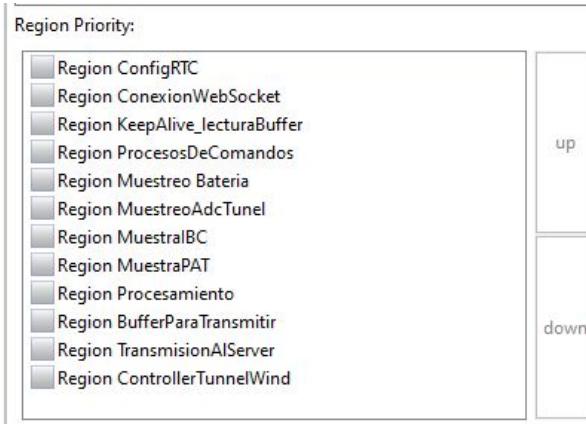


Figura 4.43: Lista de regiones con su respectiva prioridad.

Se utilizó una licencia 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, como se ejemplifica en el Código 4.1.

```

1 import: "inc/common.h"
2 import: "inc/da_acquisition.h"
3 import: "inc/da_processing.h"

```

```

4 import: "inc/da_transmision.h"
5 import: "inc/da_rtc.h"
6 import: "inc/da_configDatalogger.h"
7 import: "inc/controlTunnel.h"
8 /*Datalogger para Sistema anemometrico con EDU-CIAA-NXP */
9 interface:
10 //Variables
11 var startMesuare:uint8_t = FALSE
12 //Puntero
13 var ptrStartMesuare:ptrUnt8_t
14 var viTeclaOprim: uint16_t
15 //Constantes
16 const TRUE: uint8_t =1
17 const FALSE: uint8_t =0
18 const Tec1: uint16_t = 1<<0
19 const Tec2: uint16_t = 1<<1

```

Código 4.1: Configuración principal del modelo statechart dataloggerSA.

También se agregaron eventos internos que permiten transicionar del estado de una región a otro estado de otra región (*raiseEvent* en inglés), como se muestra en el Código 4.2.

```

1 internal:
2 //En forma interna tengo seniales para vincular las maquinas de estado
3 event siDatosListosParaBdtos
4 event siDatosProcesado
5 event siMuestrasNuevas
6 event siStartMeasure
7 event siKeepAlive
8 event siProcesMsjServer

```

Código 4.2: Declaración de eventos internos que permiten transicionar entre regiones del statechart.

4.3.1. Configuración y conexión con los servidores

Como se detalló en la sección 4.1.3, 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 y permite configurar un sistema embebido como servidor o cliente, como se muestra en la Figura 4.44.

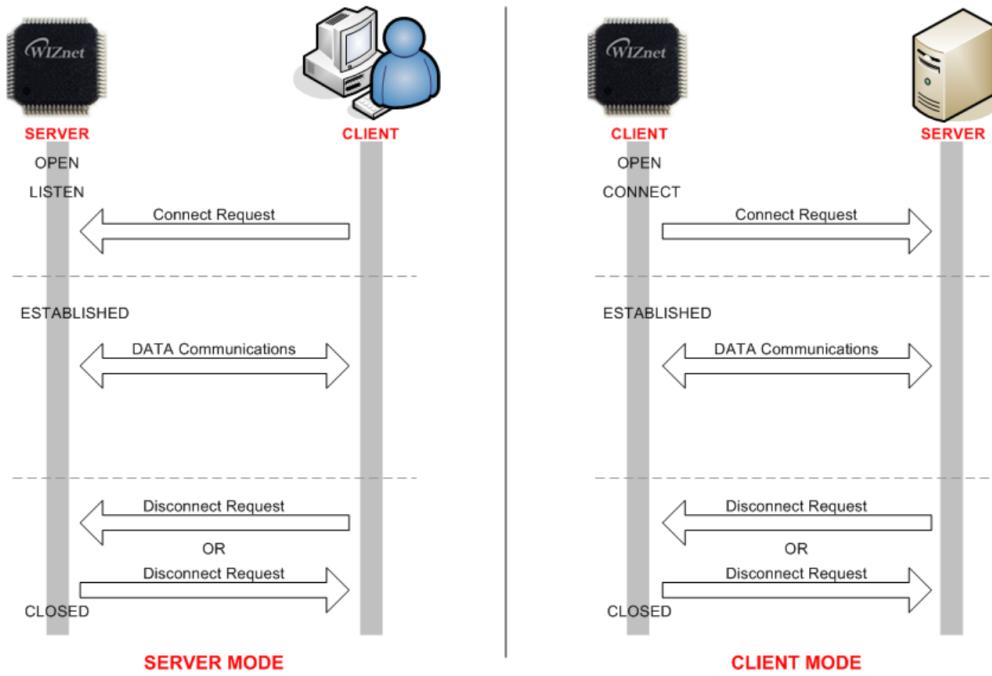


Figura 4.44: Modos de configuración para el módulo W5100.

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, como se indica en el diagrama de flujo de la figura 4.45.

Un socket es una interfaz que permite la comunicación entre dispositivos a través de redes. En términos simples, un socket es un punto final para enviar y recibir datos en una red, y está compuesto por una dirección IP y un número de puerto.

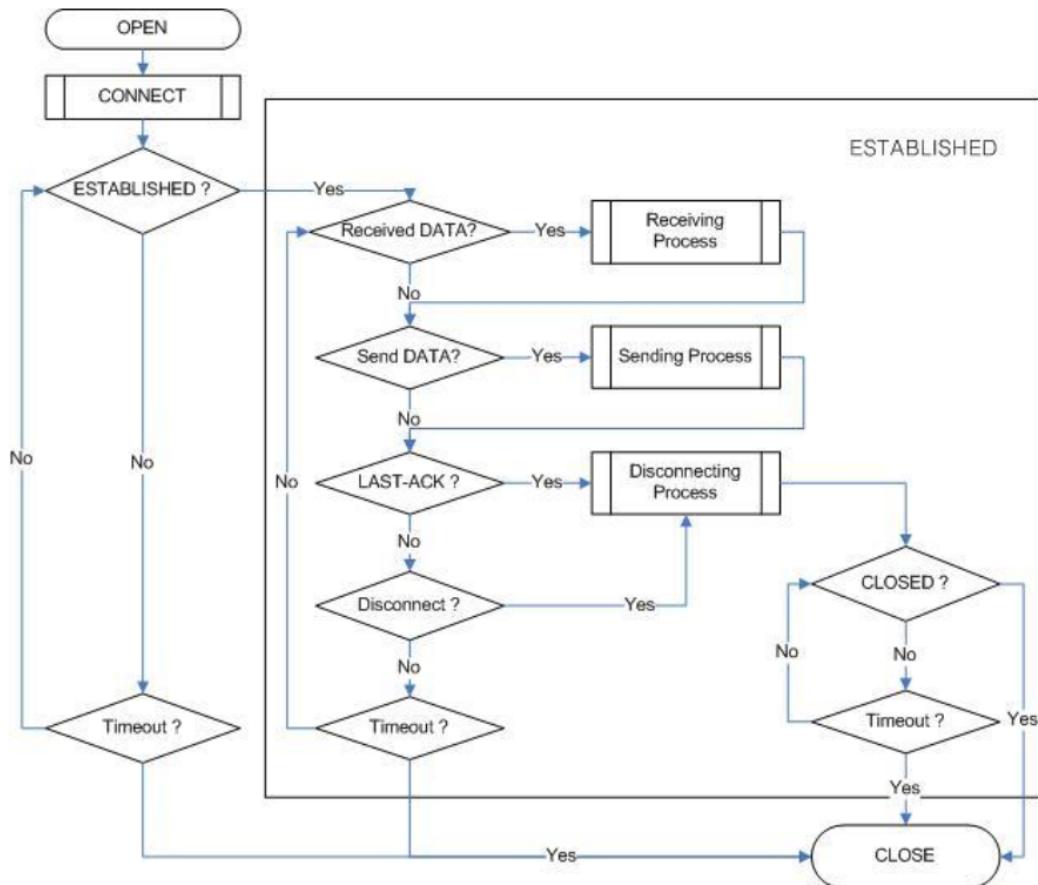


Figura 4.45: Diagrama de flujo para establecer una comunicación, en modo cliente, con un servidor utilizando sockets.

4.3.1.1. 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. Esta conexión retorna una cadena de caracteres, 59992 23-02-17 17:47:01 00 0 0 831.5 UTC(*NIST*)*, que se procesó e integró en la estructura del RTC del datalogger. Para gestionar y validar esta comunicación con el servidor NTP, se configuró una región específica dentro del sistema. Las variables declaradas en el Código 4.3 se utilizan para validar cada transición dentro de la máquina de estados ilustrada en la Figura 4.46.

```

1 // Variables para la conexión al servidor NTP
2 var ConfigNtpSocket: uint8_t = FALSE
3 var setRtc: uint8_t = FALSE
4 var ConfigNTP: uint8_t = FALSE

```

Código 4.3: Declaración de variables booleanas para validar la conexión con el servidor NTP.

La máquina comienza en el estado **INIT** y luego de un período de `DelatInit` segundos, pasa al estado **CONFIG_SOCKET**. En este estado, se llama a la función `opConfigNtpSocketViaTCP()`, que se encarga de configurar los parámetros de red del cliente: gateway, dirección MAC, máscara de subred e IP.

Luego, estos parámetros se asignan a uno de los cuatro sockets del módulo W5100, y se configura el protocolo de transporte (TCP en este caso), el puerto, y se abre la conexión para establecer una comunicación con el servidor. Esta función devuelve `True` si la apertura y configuración son correctas, almacenando este resultado en la variable `ConfigNtpSocket`. Si la configuración es exitosa, la máquina de estados transiciona al estado **CONFIG_NTP**.

En el estado **CONFIG_NTP**, se invoca la función `opConfigNtpViaTCP()`, encargada de configurar la conexión con el servidor NTP. Esta función establece la dirección IP y el puerto del servidor y realiza un intento de conexión. Si la conexión es exitosa, la función retorna `True` y el resultado se almacena en la variable `ConfigNTP`. Una vez configurado, el servidor responde con la hora y fecha actual en formato UTC, que se guarda en una variable interna. Posteriormente, la máquina de estados avanza al estado **SET_RTC_VIA_SOCKET**, 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 **IDLE**, cuando se realiza esta última transición se eleva un evento a través de la señal `siStartMeasure`, la cual será un disparador para iniciar otra máquina de estado en otra región. En caso de fallo en cualquiera de los estados previos, se transiciona al estado **SET_RTC_DEFAULT**, donde se establece una hora predeterminada.

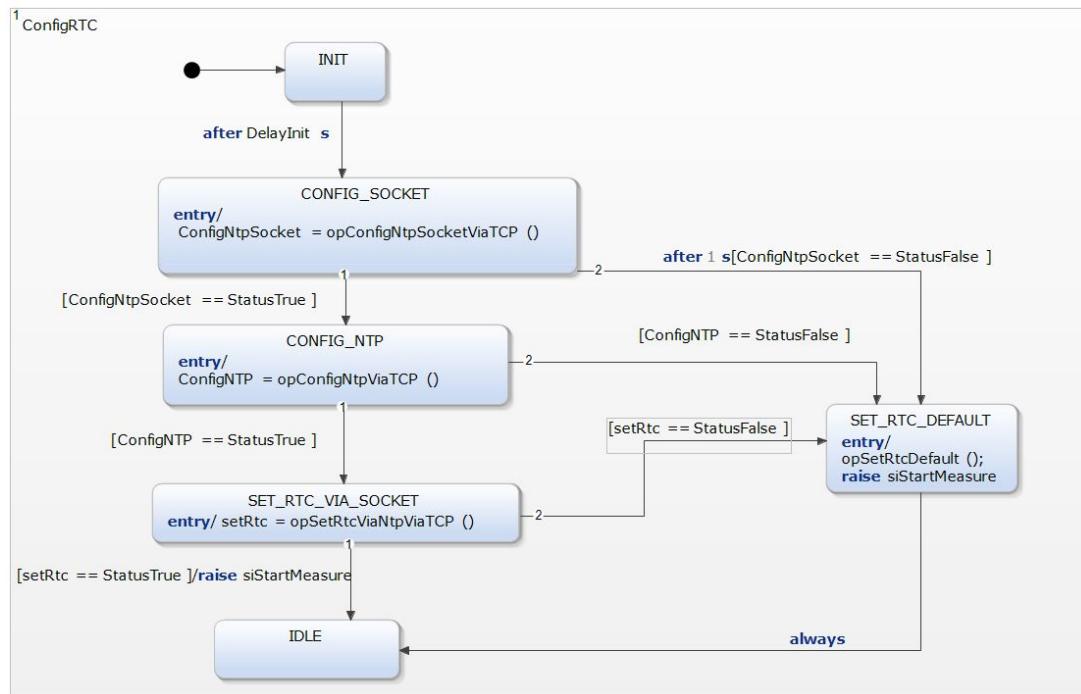


Figura 4.46: Máquina de estados para configurar el RTC del datalogger a través de un servidor NTP.

4.3.1.2. 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 como se implementó la comunicación. Para ello se declaran las variables de estado que se muestran en el Código 4.4 y permiten la transición de estados de la región `ConexionWebSocket` que se muestra en la figura 4.47

```
1 // Variables para la conexion 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 4.4: 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.

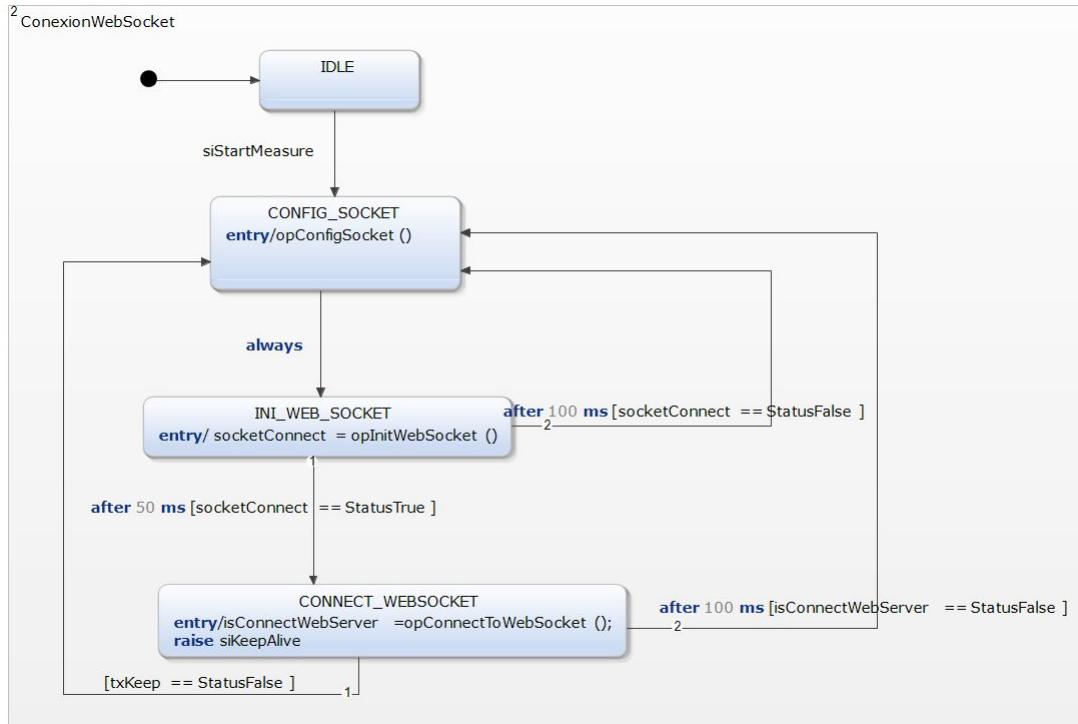


Figura 4.47: Máquina de estados para establecer una conexión bidireccional con un servidor WebSocket.

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 4.47, 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.

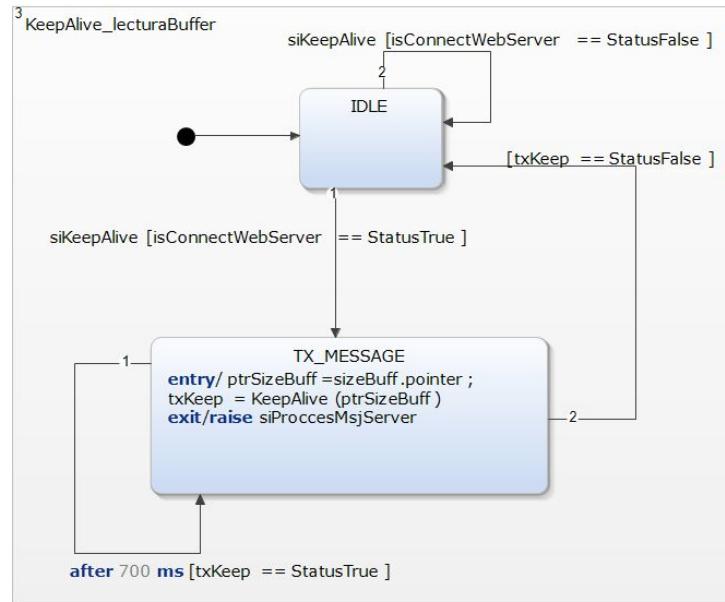


Figura 4.48: Máquina de estados para mantener la conexión con el servidor y elevar un raise en caso de que se reciba información del servidor.

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

4.5

```

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

```

Código 4.5: 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 4.3.4.

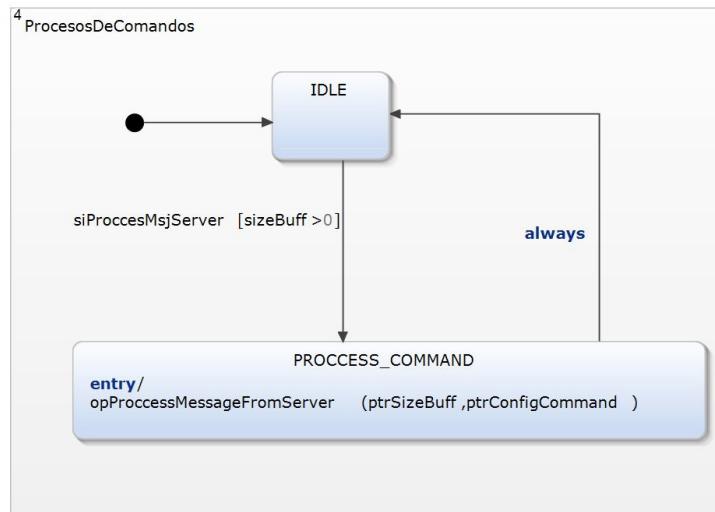


Figura 4.49: Máquina de estados para procesar los mensajes recibidos del servidor WebSocket.

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 4.6, 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 4.6: Funciones que permiten configurar o modificar el comportamiento del datalogger.

4.3.2. 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 4.7. 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 4.7: 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 4.8. Esta estructura se inicializa con datos ingresados por el operador a través de la aplicación web descrita en el capítulo 5. 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.

```

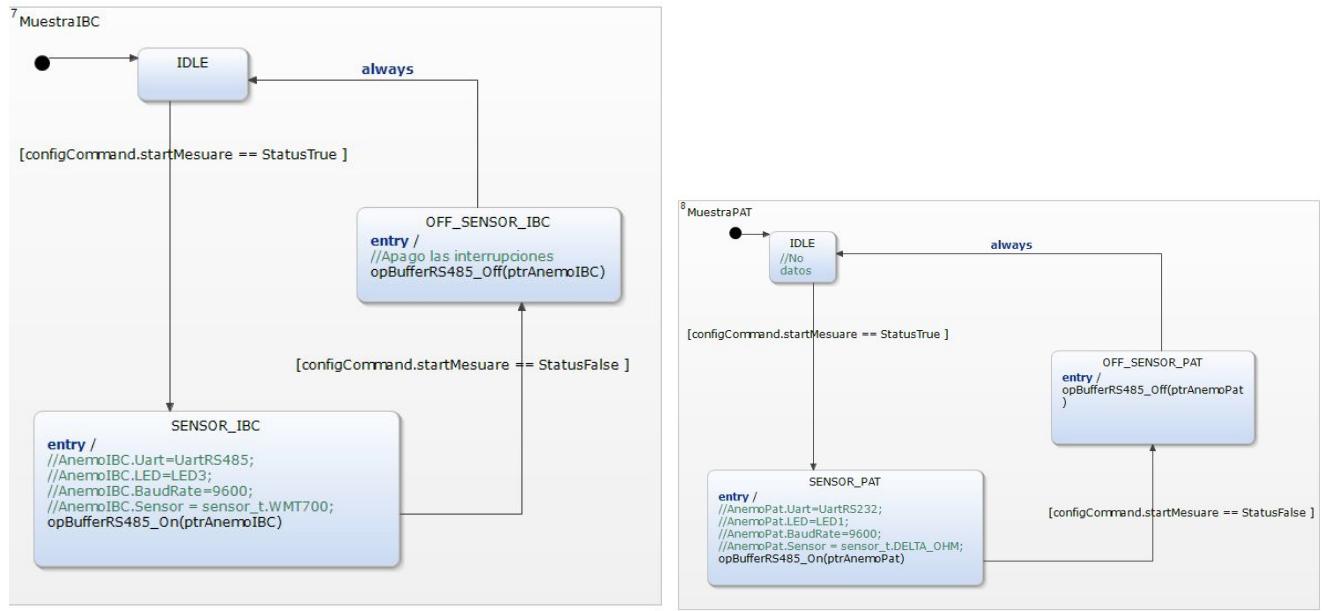
1 typedef struct{
2     uint16_t Uart;
3     uint16_t LED;
4     uint32_t BaudRate;
5     char Buffer[100];
6     char * ptrUartBuffer;
7     real32_t DataAnemometer[7];
8     real32_t * ptrDataAnemometer;
9     sensor_t Sensor;
10 }amenometerSerialParam_t;

```

Código 4.8: Estructura de datos para anemómetro.

En las Figuras 4.50a y 4.50b 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.



(a) Máquina de estados para tomar muestras de sensor bajo calibración.

(b) Máquina de estados para tomar muestras de sensor patrón.

Figura 4.50

El string del patrón VAISALA, modelo WMT700 tiene el formato `$-MWV,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 4.9. 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.

```

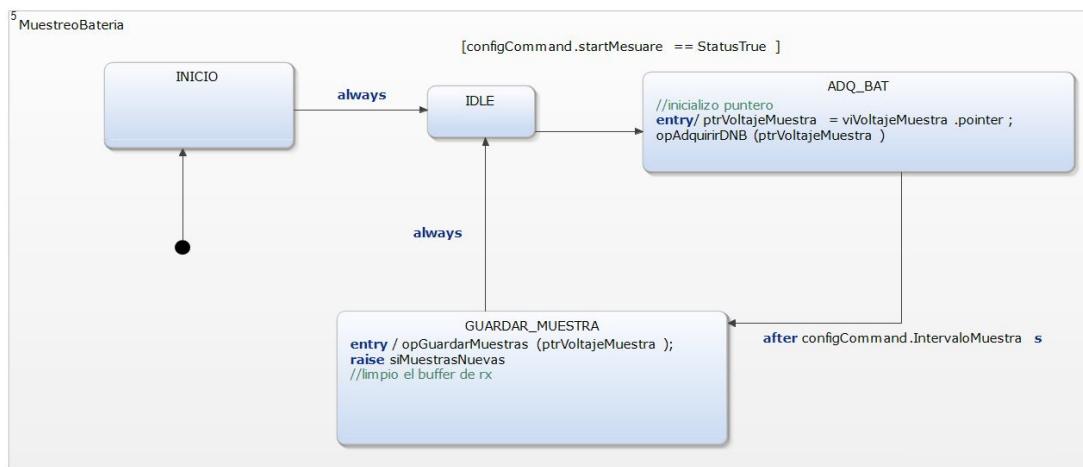
1 // Diccionario de punteros a función para procesar en función del sensor elegido
2 void (*opProcesoDatosViento [])(amenometerSerialParam_t * data) = {
3     opProcesoDeltaOHM,
4     opProcesoWMT700,
5     opProcesoVentus
6 };

```

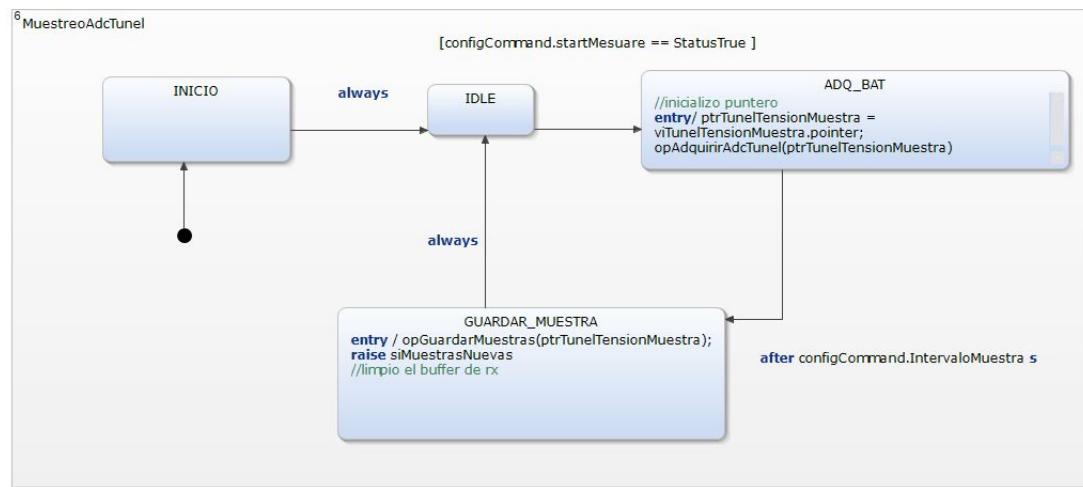
Código 4.9: Diccionario de punteros a función para procesar los datos recibidos de los anemómetros.

En las Figuras 4.51a y 4.51b 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 4.7. 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`.



(a) Máquina de estado para tomar muestras de la fuente de alimentación del datalogger.



(b) Máquina de estado para tomar muestras de la tensión que inyecta al variador de velocidad del túnel de viento.

Figura 4.51

En el Código 4.10 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 4.10: Variables para controlar la adquisición y procesamiento de los datos .

La región **Procesamiento** de la figura 4.52 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 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.

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

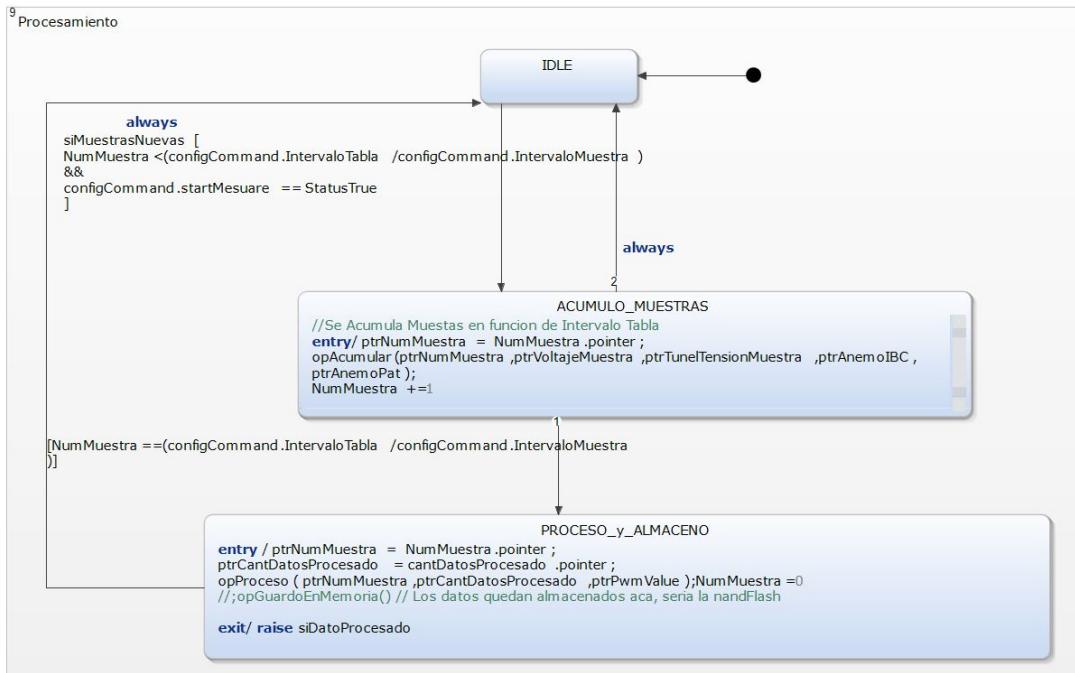


Figura 4.52: Máquina de estados para acumular muestras en una tabla y realizar un preprocesamiento de esos datos.

4.3.3. 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 4.11, esta variable controla el estado de la transmisión en la máquina de estados de la región **TransmisionAlServer** de la figura 4.53.

```

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

Código 4.11: Declaración de variable para controlar el estado de una transmisión al servidor.

La máquina de estados de la figura 4.53 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.

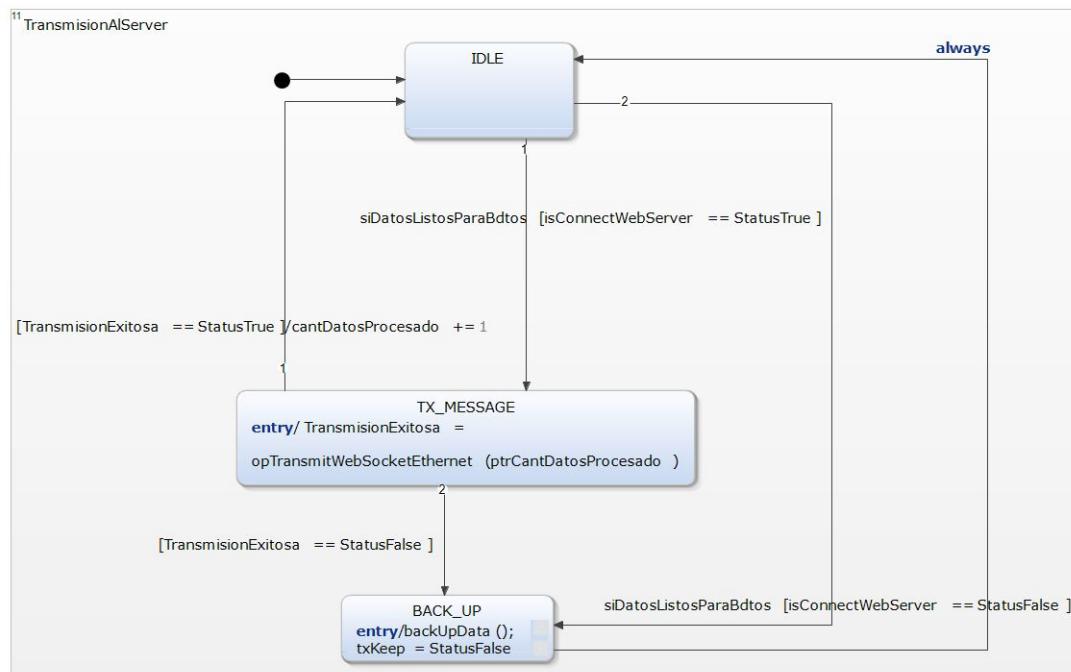


Figura 4.53: Máquina de estados para gestionar la transmisión de datos al servidor WebSocket.

4.3.4. Sistema de control PID

En la Sección 4.1.4, 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 4.12. 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 4.12: Declaración de variables para el controlador PID del motor del túnel de viento.

En el Código 4.13, 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 4.13: 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, ptrIntesidadPatron, ptrIntesidadReferencia)`, 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, ptrIntesidadPatron, ptrIntesidadReferencia, 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.

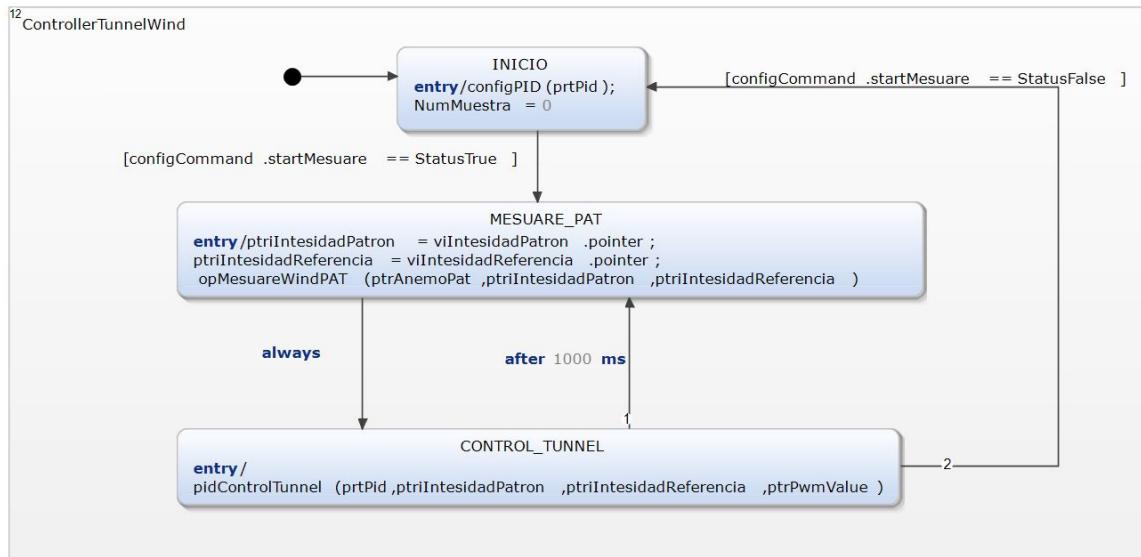


Figura 4.54: Máquina de estados que ejecuta el control PID con valores de referencia enviados por el servidor.

Capítulo 5

Implementación de la Aplicación WEB

En este capítulo, se introduce una aplicación web diseñada para interactuar con el datalogger descrito en el capítulo anterior, con el objetivo de optimizar la calibración de anemómetros. Esta herramienta de software simplifica la carga de metadatos esenciales, incluyendo la información de los sensores de referencia y los sensores bajo calibración, los certificados de calibración y las especificaciones de la zona de medición del túnel de viento.

La aplicación ofrece la posibilidad de configurar el datalogger, definiendo la interfaz eléctrica de los anemómetros, los intervalos de muestreo y procesamiento de datos, y los puntos de medición de la velocidad del viento. Además, permite establecer los tiempos de encendido y el periodo de estabilidad en cada punto de medición. Una vez configurado el sistema, el software inicia el proceso de medición, enviando referencias de viento al hardware. Este último interactúa con el túnel de viento a través de su controlador PID para ajustar la velocidad del viento al valor deseado. Las mediciones obtenidas son verificadas por el operador y, si son correctas, se calcula automáticamente la incertidumbre expandida. Los resultados se presentan en el front end, en gráficos y tablas, se almacenan en una base de datos y pueden descargarse para emitir el certificado de calibración correspondiente.

La integración de esta aplicación con el datalogger automatiza y estandariza el proceso de calibración, reduciendo de manera significativa el tiempo dedicado al procesamiento manual de datos y a la separación de estos en diferentes etapas. Esto contribuye a la reducción de tiempos operativos y minimiza errores sistemáticos, mejorando notablemente la calidad y precisión en la calibración de sensores de viento de ultrasonido.

5.1. Desarrollo aplicación WEB

Durante el desarrollo de la aplicación web, se emplearon diversas herramientas para llevar a cabo el proyecto de manera eficiente. Se utilizó Django en su versión 4.2 como framework principal, junto con Python 3.10, proporcionando una base sólida y flexible para el desarrollo backend. Para la gestión de la base de datos, se optó por PostgreSQL en su versión 15 y se utilizó PGAdmin 4 para gestionar dicha base de datos. La conexión entre Django y PostgreSQL se realizó utilizando PSYCOPG2 en su versión 2.9.5.

En el frontend, se emplearon JavaScript y jQuery para hacer la página web dinámica y permitir la visualización de datos actualizados en tiempo real. También se utilizaron HTML y CSS para estructurar y estilizar el contenido de la aplicación web. Además, se emplearon herramientas como NumPy 1.24.2, SciPy 1.11.2 para el procesamiento de datos y Plotly 5.15 para generar gráficos dinámicos, así como Pandas 2.2 para la manipulación de datos.

Adicionalmente, se trabajó en un servidor local, dentro de un entorno virtual de Python para mantener las dependencias aisladas y facilitar la portabilidad del proyecto a otros entornos. Se gestionó el control de versiones con Git y GitHub, en el repositorio https://github.com/InstrumentalSMN/AppWebSistemaDecalibracion/tree/develop_apps, subiendo el código y trabajando en distintas ramas para luego integrar los cambios en la rama principal, lo que permitió mantener un historial claro de los cambios realizados.

5.1.1. Arquitectura del software

Para desarrollar la aplicación, se utilizó una arquitectura de cinco capas, como se muestra en la Figura 5.1. La primera capa está conformada por el servidor web Nginx, que actúa como un servidor *proxy* inverso. Nginx se encarga de manejar las solicitudes HTTP entrantes, distribuyéndolas eficientemente y proporcionando un equilibrio de carga, además de servir contenido estático como archivos CSS, JavaScript e imágenes. Esta capa permite mejorar el rendimiento y la seguridad de la aplicación al filtrar y dirigir el tráfico de manera óptima.

La segunda capa está compuesta por Gunicorn, un servidor WSGI (*Web Server Gateway Interface*) que sirve como intermediario entre Nginx y la aplicación web desarrollada en Django. Gunicorn se encarga de gestionar los procesos de la aplicación, proporcionando un entorno escalable y capaz de manejar múltiples solicitudes concurrentes. Esta capa es crucial para asegurar que la aplicación pueda responder de manera rápida y eficiente a las demandas de los usuarios.

En la tercera capa se encuentra la aplicación web desarrollada en Django, un *framework* de alto nivel que facilita el desarrollo rápido y eficiente de aplicaciones web seguras y mantenibles. Esta capa es responsable de la lógica de negocio de la aplicación, la gestión de las bases de datos

y la interacción con los usuarios a través de interfaces web dinámicas.

La cuarta capa está constituida por el sistema de gestión de bases de datos PostgreSQL, encargado del almacenamiento y recuperación de datos de manera eficiente y segura. La integración con Django se realiza a través de su ORM (*Object-Relational Mapping*), que permite interactuar con la base de datos utilizando objetos en Python. Las tablas se definen creando una clase que hereda de `models.Model` de Django, donde los campos se establecen como atributos de la clase. Esto permite crear registros, guardar información y realizar consultas de manera sencilla, traduciendo automáticamente las operaciones en Python a consultas SQL en la base de datos. Esta abstracción de los detalles subyacentes de la implementación facilita la gestión y manipulación de datos, mejorando la eficiencia en el desarrollo de aplicaciones web.

Finalmente, la quinta capa consiste en un servidor WebSocket que se conecta con un sistema embebido. Este servidor permite la comunicación bidireccional en tiempo real entre la aplicación y dispositivos embebidos, facilitando el control y monitoreo remoto. Esta capa es esencial para aplicaciones que requieren actualizaciones en tiempo real y una interacción continua con dispositivos de hardware.

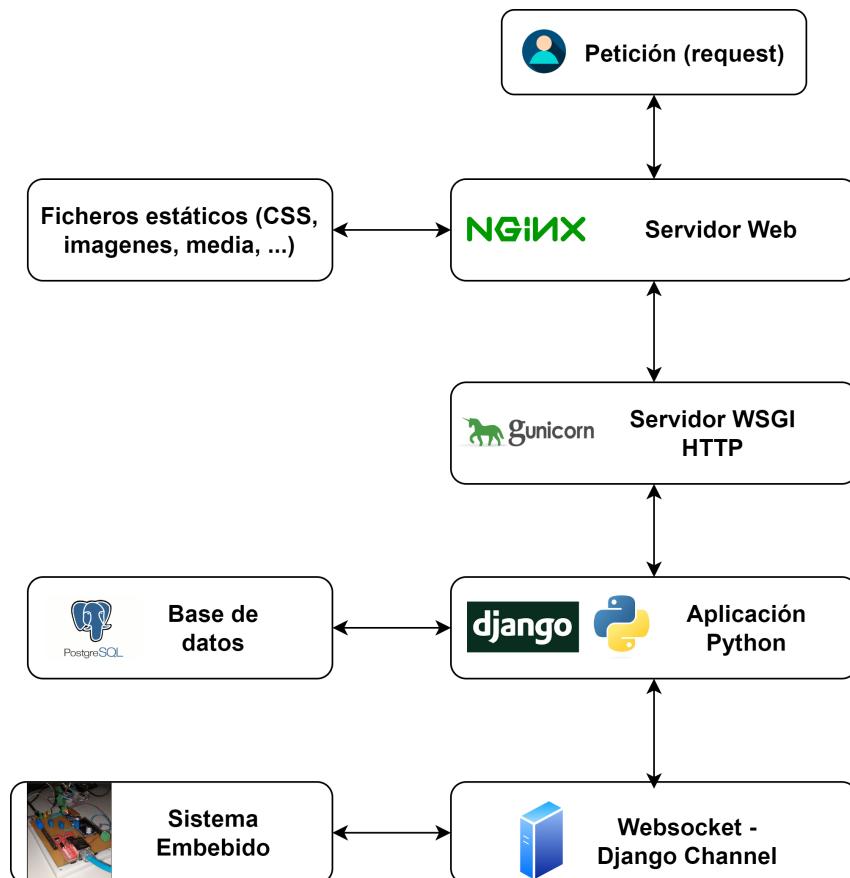


Figura 5.1: Arquitectura del software implementada.

Se empleó el patrón de diseño MVT (Modelo-Vista-Plantilla), ilustrado en la Figura 5.2. Este

patrón es ampliamente utilizado en el desarrollo web con el *framework* de trabajo Django. El patrón MVT permite una distinción explícita de las responsabilidades, lo que facilita tanto el desarrollo como el mantenimiento del código.

- **Modelo (M)**: Representa la capa de acceso a la base de datos (ORM). Contiene toda la información sobre los datos, cómo acceder a ellos, validarlos, su comportamiento y las relaciones entre ellos.
- **Vista (V)**: Corresponde a la capa de lógica de negocios. Se encarga de procesar las solicitudes del navegador y recuperar los datos necesarios del modelo. Luego, renderiza la plantilla (template) para mostrar el HTML resultante.
- **Template (T)**: (Plantilla), representa la capa de presentación, es la parte visual de la aplicación. El template integra los datos dinámicos recuperados del modelo y genera el HTML final que se envía al navegado

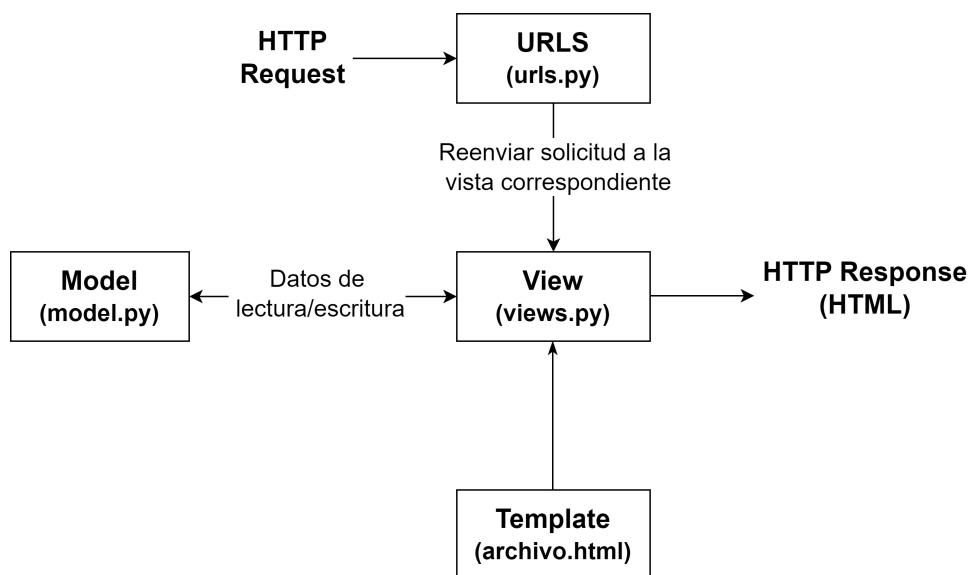


Figura 5.2: Patrón de diseño de software MVT, que divide la lógica del programa en tres elementos interconectados.

5.1.2. Back End

El Back end se encarga de:

- Administrar la base de datos, incluyendo la ejecución de consultas, la realización de actualizaciones y la aplicación de validaciones para garantizar la integridad y consistencia de la información.

- Recibir o gestionar solicitudes HTTP y conexiones al servidor WebSocket.
- Enviar comandos de configuración hacia el datalogger y gestionar la recepción de mediciones de los sensor.
- Generar un perfil de referencias de velocidad de viento, que se van a transmitir al datalogger para controlar el motor del túnel.
- A partir de las mediciones crudas de velocidades de viento, calcular el presupuesto de incertidumbre de la corrección.

5.1.2.1. Base de datos

Dado que la aplicación está diseñada para manejar una gran cantidad de datos que requieren almacenamiento y recuperación eficientes, se ha implementado una base de datos relacional. El diagrama entidad-relación, presentado en la figura 5.3, ilustra la estructura de la base de datos. Esta estructura incluye diversas tablas que almacenan información crucial para la aplicación. Entre estas tablas se encuentran las que registran los datos del sensor patrón y el sensor bajo calibración, así como los detalles relacionados con el certificado de calibración del patrón. Además, se documentan los registros del certificado de caracterización del túnel de viento. También se incluye información sobre la configuración del datalogger y del túnel de viento.

Cuando se inicia una calibración y se empiezan a recibir datos de los sensores, toda esta información se almacena en una tabla **DatosMedidos2023** de la base de datos. Asimismo, se guardan los resultados generados antes y después del procesamiento de los datos, como gráficos y tablas.

Todas estas tablas están relacionadas mediante claves foráneas (relación de uno a muchos) con la tabla de **Calibración**. Esto se debe a que una calibración está asociada a un único datalogger o a un único sensor IBC, y se puede recuperar la información de esa calibración a través clave primaria.

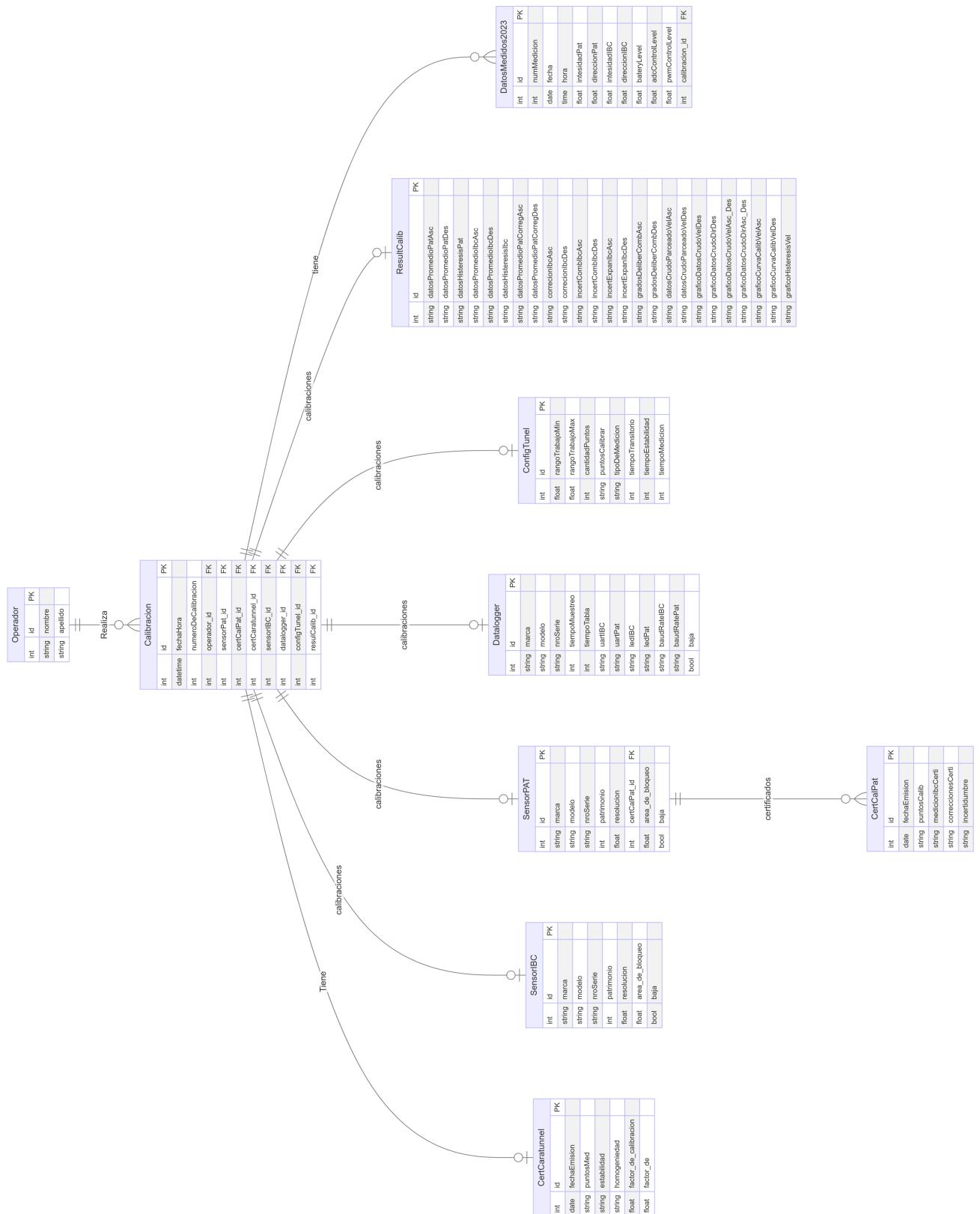


Figura 5.3: Diagrama entidad-relación diseñada para la aplicación Web.

5.1.2.2. Servidor WebSocket

Se utilizó la biblioteca Django-channels para implementar un servidor WebSocket que opera de forma concurrente con el servidor web HTTP. Este servidor permite la conexión simultánea de múltiples clientes, en este caso, un datalogger que envía datos y un usuario que recibe dicha información a través de WebSockets. Los datos recibidos por el servidor, son graficados y actualizados en tiempo real en la aplicación web.

Para configurar el servidor, se creó el archivo `routing.py`, el cual define las rutas de los canales, es decir, los puntos de acceso a los consumidores (consumers) que gestionan las conexiones WebSocket. La configuración del archivo `routing.py` se presenta en el Código 5.1. Cada ruta mapea una URL de WebSocket a un consumidor específico. En este caso, `ChatConsumer` es una clase que maneja las conexiones a la URL `ws/socket-server/`.

```

1 from django.urls import re_path
2 from . import consumers
3
4 websocket_urlpatterns = [
5     re_path(r'ws/socket-server/', consumers.ChatConsumer.as_asgi())
6 ]

```

Código 5.1: Configuración del routing para gestionar las conexiones de clientes al servidor WebSocket.

También se creó el archivo `consumers.py` que define las clases de consumidores que manejan las conexiones WebSocket. Un consumidor en Django-channels es una clase que gestiona la conexión con el cliente, incluyendo el manejo de eventos como la recepción y el envío de mensajes. En el código 5.2 se muestra el consumidor implementado

```

1 class ChatConsumer(WebSocketConsumer):
2     def connect(self):
3         self.room_group_name = 'test'
4         sync_to_sync(self.channel_layer.group_add)(
5             self.room_group_name,
6             self.channel_name
7         )
8         self.accept()
9
10    def disconnect(self, close_code):
11        pass
12
13    def receive(self, text_data):
14        text_data_json = json.loads(text_data)
15        message = text_data_json['message']
16        # message = text_data
17        sync_to_sync(self.channel_layer.group_send)(
18            self.room_group_name,
19            {
20                'type': 'chat_message',
21                'message': message,

```

```

22         'sender_channel_name': self.channel_name,
23     }
24   )
25   if(len(message)>0):
26     if(message[0] == "M"):
27       saveMesuareDataBase(message)
28     print('Message:', message)
29
30   def chat_message(self, event):
31     message = event['message']
32     sender_channel_name = event['sender_channel_name']
33     if sender_channel_name != self.channel_name:
34       self.send(text_data=json.dumps({
35         'type': 'chat',
36         'message': message
37       }))

```

Código 5.2: Declaracion de las funciones que utiliza cada consumidor para interactuar con el servidor WebSocket.

En particular, se implementa la clase `ChatConsumer`, que hereda de `WebsocketConsumer`, para gestionar las conexiones WebSocket. Esta clase define varios métodos, incluyendo `connect`, `disconnect`, `receive` y `chat_message`.

El método `connect` se ejecuta cuando un cliente se conecta al servidor WebSocket. Dentro de este método, se asigna el nombre del grupo de la sala a `self.room_group_name` y se añade el canal del cliente al grupo utilizando `async_to_sync(self.channel_layer.group_add)`. Finalmente, se acepta la conexión mediante `self.accept()`. Por otro lado, el método `disconnect` se ejecuta cuando un cliente se desconecta del servidor WebSocket.

El método `receive` recibe y decodifica los mensajes desde JSON para luego enviarlos a un grupo de la sala utilizando `async_to_sync(self.channel_layer.group_send)`. Los mensajes que envía el datalogger tiene la forma del arreglo declarado en código 5.3, que contiene toda la información que ha recolectado el datalogger en un ciclo de iteración.

```

1 real32_t Tabla_Mediciones[25] = {
2   velocidadInst, velocidadMin, velocidadMax, velocidadPromedio,
3   velocidadInstIBC, velocidadMinIBC, velocidadMaxIBC, velocidadPromedioIBC,
4   direccionInst, direccionMin, direccionMax, direccionPromedio,
5   direccionInstIBC, direccionMinIBC, direccionMaxIBC, direccionPromedioIBC,
6   nv_bateriaInst, nv_bateriaMin, nv_bateriaMax, nv_bateriaPromedio,
7   nv_adcTunelInst, nv_adcTunelMin, nv_adcTunelMax, nv_adcTunelPromedio,
8   nv_pwmControlTunelInst
9 };

```

Código 5.3: Estructura del mensaje enviado por el datalogger

El mensaje decodificado tiene la siguiente forma:

```

poner numeros de la base de datos
M;634;14-06-2024;16:42:41;

```

```

10.08;10.08;10.08;10.08;
9.53;9.53;9.53;9.53;
169;169;169;169;
175;175;175;175;
12.02;12.02;12.02;12.02;
1.25;1.25;1.25;1.25;
90;

```

De esta forma, si el mensaje recibido comienza con la letra "M", se guarda en la base de datos mediante la función `saveMeasureDataBase(message)`. Esta función conoce el orden en que se creó el string y separa los datos para guardarlos en su respectivo campo de la base de datos.

Finalmente, el método `chat_message` se utiliza para enviar mensajes a los clientes. Si el mensaje recibido no se origina del mismo canal que el receptor, se envía el mensaje de vuelta al cliente utilizando `self.send()`. Por ejemplo, cuando un cliente está conectado al WebSocket desde el navegador web, puede ver cómo se grafican los datos en tiempo real a medida que estos llegan desde el datalogger hacia servidor.

5.1.2.3. Generador de trayectoria y control del túnel

En la sección 4.3.4 se ha detallado el funcionamiento del controlador PID del túnel de viento. Para que este funcione, necesita referencias que se construyen a partir de un generador de trayectoria utilizando el método de Paul [3]. El generador toma los valores de velocidad de viento deseados y los tiempos deseados desde la configuración realizada por el operador en el frontend, y con estos valores se genera una curva de referencia de velocidades de viento en función del tiempo.

El método de Paul consiste en definir, para cada punto, dos zonas: una de aceleración constante y otra de velocidad constante. Cada segmento tiene una duración T_j y un tiempo de aceleración t_{acc} , como se muestra en la figura 5.4, de forma tal que no se realicen cambios abruptos en el controlador, permitiendo que los cambios de velocidad del viento sean suaves.

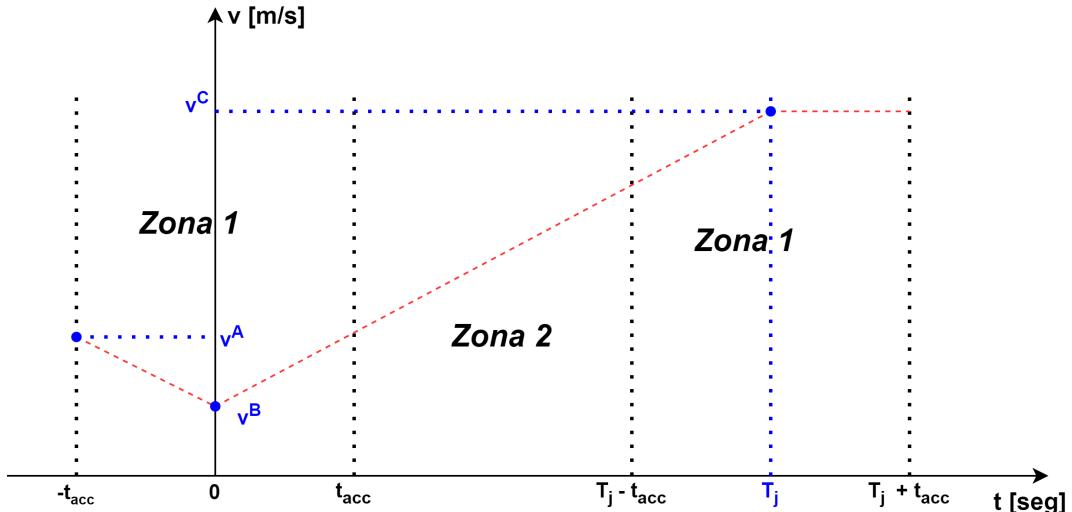


Figura 5.4: Segmento para pasar de un punto de velocidad v^B a otro v^C vieniendo desde v^A .

Estos cambios suaves se traducen, a través del controlador PID, en cambios suaves en los valores de PWM para el motor.

Para la zona 2 del segmento T_j , donde se define $\Delta C = v^C - v^B$, el método de Paul aplica la ecuación 5.1:

$$v(t_{seg}) = \frac{\Delta C}{T_j} + v^B \quad (5.1)$$

Para la zona 1 del mismo segmento, donde se define $\Delta A = v^A - v^B$, se aplica la ecuación 5.2:

$$v(t_{seg}) = \frac{\Delta C}{T_j} \frac{(t_{seg} + t_{acc})^2}{4t_{acc}} + \frac{\Delta A}{t_{acc}} \frac{(t_{seg} - t_{acc})^2}{4t_{acc}} + v^B \quad (5.2)$$

Esto garantiza que en la zona 1, donde se realiza el enganche de segmento, la aceleración del cambio sea constante, y que en la zona 2, donde se realiza la transición de un segmento a otro, la velocidad de cambio sea constante.

El algoritmo generador de trayectoria se muestra en el código B.1. Este se ejecuta en el backend y calcula la trayectoria en función de cada punto de viento (segmento) y los tiempos de crecimiento, estabilización y medición configurados por el operador. Luego, realiza el encadenamiento de todas las velocidades para obtener un perfil como el de la Figura 5.5, donde se muestra una trayectoria de ciclo ascendente y descendente para los puntos 5, 10, 15, 20 y 25 $m s^{-1}$, con un tiempo de crecimiento de 2 min, un tiempo de estabilización de 5 min y un tiempo de medición de 2 min.

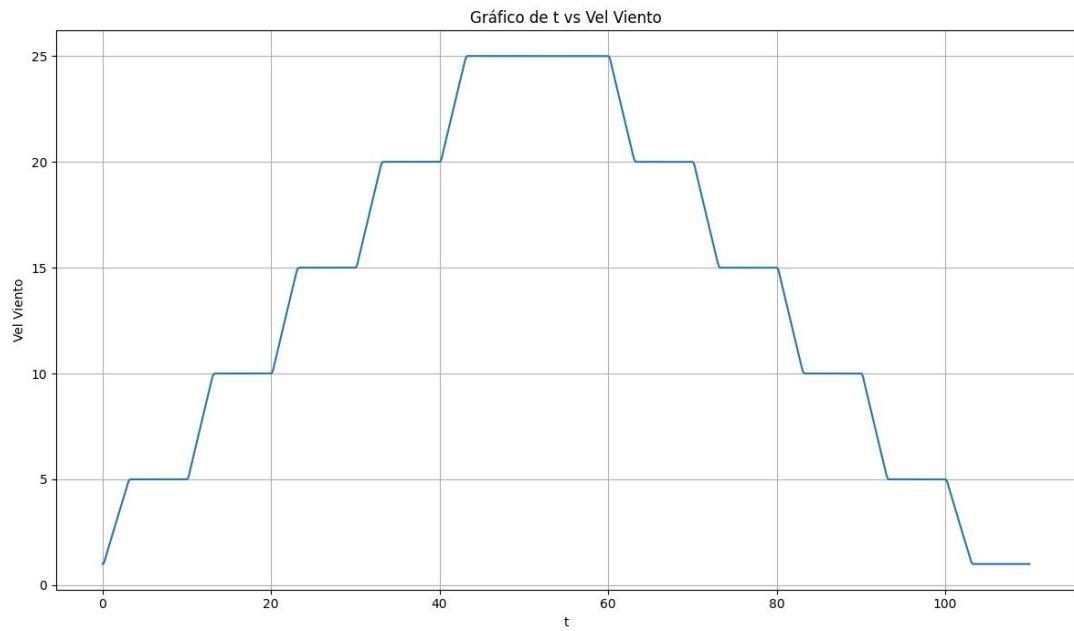


Figura 5.5: Trayectoria generada utilizando el método de Paul para un ciclo ascendente y descendente de velocidades.

5.1.2.4. Cálculo de incertidumbre

El algoritmo calcula la incertidumbre con los puntos parceados en la sección anterior, el En la tabla ?? se muestran las incertidumbres consideradas, el algoritmo realiza para cada punto de viento configurador por software, la siguiente cuenta

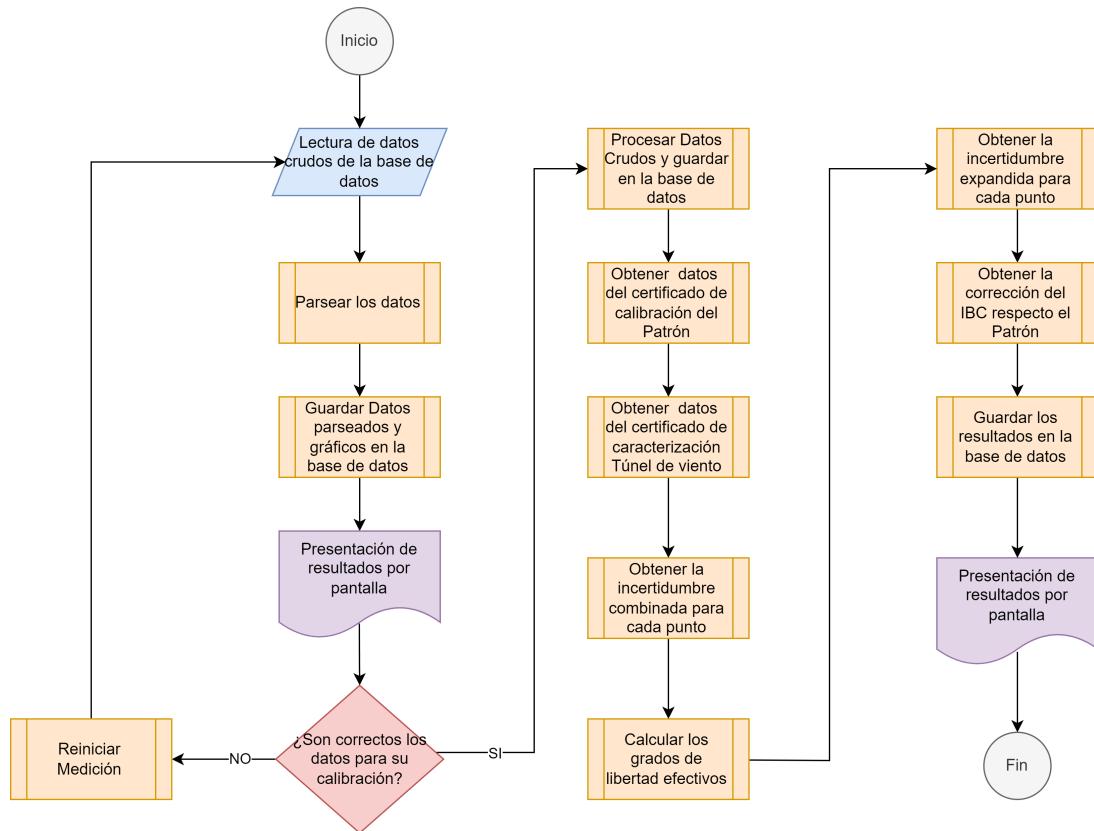


Figura 5.6: Diagrama de flujo del cálculo de incertidumbre.

5.1.3. Front End

-Presentación de datos al usuario (UI/UX). -Interacción del usuario (formularios, botones, eventos). -Comunicación con el backend a través de API REST o WebSocket. -Validaciones iniciales de datos del usuario (pueden hacerse también en el backend por seguridad).

mas que todo contar como se usa el soft, que aca se entienda como se usa el front

5.1.3.1. Carga de metadatos

que se carga en esta sección, capturas de pantalla de la vista

5.1.3.2. Configuración del sistema

que se carga en esta sección, capturas de pantalla de la vista

5.1.3.3. Adquisición de datos

que hace esta sección como se inicia y para, buscar alguna captura de pantalla, contar que comando se enviar internamente para y recibe la ciaa en formato JSON, se relaciona con el puntero a función de la sección ??..

5.1.3.4. Procesamiento de datos

captura de pantalla de los datos crudos, mas su grafico y el boton para calcular la incertidumbre

5.1.3.5. Resultados

mostrar los resultados que muestra el soft como ejemplo usar los del segundo DeltaOHM

5.2. Integración con el hardware

Cuando tenga armado el PCB, hago una prueba de configuración del datalogger, configuración del túnel, metadatos para la calibración

diagrama de como queda todo el sistema,

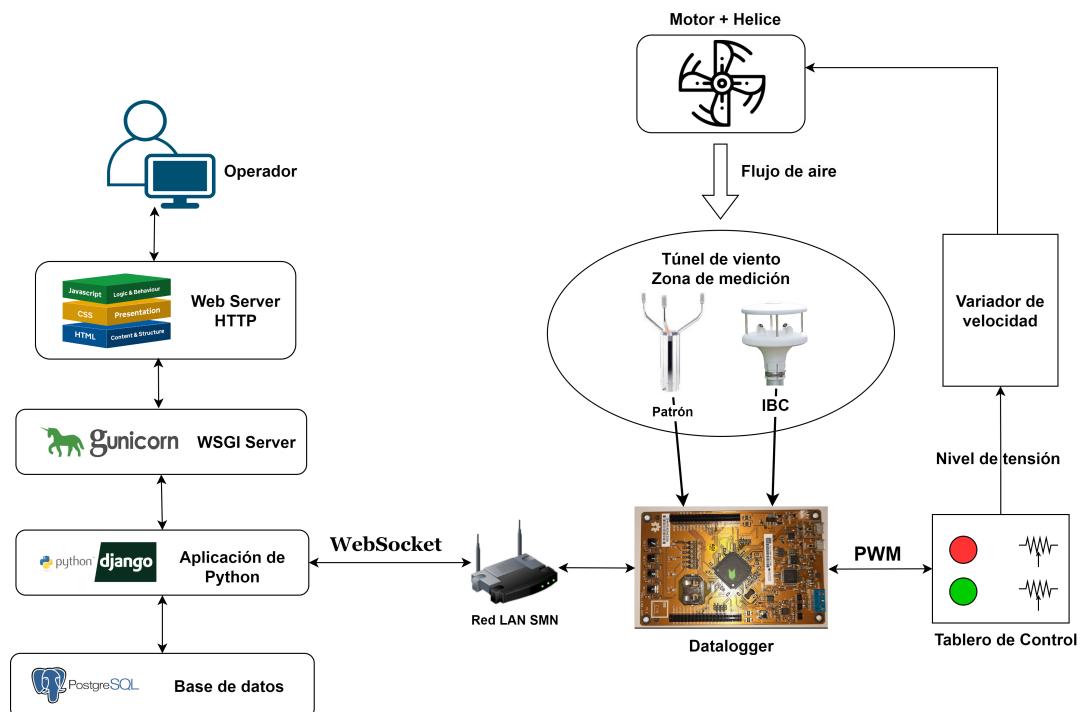


Figura 5.7

mostrar un diagrama de flujo desde el inicio con el botón calcular incertidumbre hasta la salida de tablas con los resultados, pasos del proceso de calibración

Capítulo 6

Mediciones y Resultados experimentales

6.1. Caracterización de la zona de medición

6.1.1. Banco de medición

6.1.2. Procesamiento de datos

6.1.3. Resultados

Se realizaron mediciones de 5 m/s a 25 m/s variando las distancias sobre el eje x (eje de la dirección del viento), a una altura constante en el eje z y centrado en el eje y constante,

- Se encontró que dos pares de puntos donde para generar una cantidad de viento se precisa aproximadamente el mismo ciclo de trabajo. Esos puntos son en -20cm y 60 cm y entre 10cm y 40 cm,
- Entre -20 y 60 cm quizá mejore el disminuya el factor de bloqueo pero en ambos casos el controlador sufre cambios mas abruptos de tensión debido a diferencia de 10 y 40 cm
- Las velocidades en estos 4 puntos obtenidas a partir del controlador son relativamente estables, calcular varianza y media, de todas formas en -20 y 60 si bien estabiliza, sufre mas el controlador, con picos de tensión para mantener estable, osea le cuesta mas estabilizar en esos puntos
- Entre -20 y 60 tengo una diferencia de dirección del viento aprox de 4 grados, en cambio entre 10 y 40 tengo una diferencia en grados de 2 grados,(Para esto hacerlo bien calcular

media y varianza). Ojo que esto tambien puede tener que ver en como me quedo el brazo para medir en 60, pudo haber quedado un poco inclinado y girado (repetir Medicion).

6.2. Calibración Delta OHM HD500

Dar un diagrama de flujo donde se muestra la secuencia del proceso de calibración

6.2.1. Banco de medición

6.2.2. Procesamiento de datos

6.2.3. Resultados

6.3. Calibración Vaisala WMT700

6.3.1. Banco de medición

6.3.2. Procesamiento de datos

6.3.3. Resultados

Cálculo del factor de bloqueo medición 1 Sensor Delta OHM

Primera área (Soporte): $30 \text{ cm} \times 7 \text{ cm} = 210 \text{ cm}^2$ Segunda área (Sensor): $10 \text{ cm} \times 15 \text{ cm} = 150 \text{ cm}^2$

Área total en cm^2 : $210 \text{ cm}^2 + 150 \text{ cm}^2 = 360 \text{ cm}^2$

convertir a metros cuadrados: $360 \text{ cm}^2 = 0.036 \text{ m}^2$

Area del patrón Primera área (Soporte): $27.5 \text{ cm} \times 8 \text{ cm} = 220 \text{ cm}^2$ Segunda área (Sensor-Cuernos): $19 \text{ cm} \times 1.5 \text{ cm} = 28.5 \text{ cm}^2$

Área total en cm^2 : $28.5 \text{ cm}^2 \times 3 + 220 \text{ cm}^2 = 305 \text{ cm}^2$

convertir a metros cuadrados: $305 \text{ cm}^2 = 0.0305 \text{ m}^2$

Entendido, los diámetros de extremo a extremo de la elipse del túnel de viento son 119 cm y 60 cm. Vamos a calcular el área total de la sección transversal de la elipse en metros cuadrados.

$$\text{Area} = \pi \text{radio mayor} \text{radio menor} \quad \text{Área} = 3.14 \times (120/2) \times (61/2) = 0.5749 \text{ m}^2$$

Factor de Bloqueo DeltaOhm = $0.036 / 0.5749 = 0.0626$ Factor de Bloqueo Vaisala = $0.0305 / 0.5749 = 0.05305$ Aca se muestran los resultados de la calibracion del sistema.

Sistema versatil

Hacer pruebas con una rampa, y una lineallist

Capítulo 7

Conclusiones

7.1. Comportamiento del sistema

7.2. Conclusiones finales

7.3. Trabajo futuro

Apéndice A

Datalogger

A.1. 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.

A.2. Sección 2

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.

Apéndice C

Cálculo de incertidumbre combinada

Bibliografía

- [1] World Meteorological Organization. «Surface Wind Measurement». En: *Guide to Meteorological Instruments and Methods of Observation*. WMO-No. 8. World Meteorological Organization, 2018, Chapter 5.
- [2] Curso de Observaciones Meteorológicas - Centro regional de formación. *Parte 1 - Viento*. 2022. URL: <https://crf.smn.gob.ar/>.
- [3] Robótica (86.15), Facultad de Ingeniería, Departamento de Electrónica. *Teórica - Generación de Trayectoria*. <https://campusgrado.fi.uba.ar/enrol/index.php?id=384>. Notas de la materia sobre el método de Paul aplicado en generación de trayectorias. Jul. de 2023.
- [4] Embedded There. *RS485 Communication Protocol: Basics, Working Principle and Applications*. Accessed: July 23, 2024. 2023. URL: <https://embeddedthere.com/rs485-communication-protocol>.
- [5] 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>.
- [6] 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>.
- [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/en/products/itemis-create/documentation/user-guide/overview_what_are_itemis_create_statechart_tools.