UNIVERSIDAD NACIONAL DE SAN ANTONIO ABAD DEL CUSCO

FACULTAD DE INGENIERÍA ELÉCTRICA, ELECTRÓNICA, INFORMÁTICA Y MECÁNICA

ESCUELA PROFESIONAL DE INGENIERÍA ELECTRÓNICA

TESIS

MODELO DE CONSUMO ENERGÈTICO BASADO EN APRENDIZAJE AUTOMATICO PARA MIDDLEWARE EMBEBIDO DE IOT

PRESENTADO POR:

Br. HANAN RONALDO QUISPE CONDORI

PARA OPTAR AL TÍTULO PROFESIONAL DE INGENIERO ELECTRÓNICO

ASESOR:

Mgt. Ing. MILTON JHON VELASQUEZ CURO

CUSCO - PERÚ
2024

INFORME DE ORIGINALIDAD

(Aprobado por Resolución Nro.CU-303-2020-UNSAAC)

El que suscribe, **Asesor** del trabajo de investigación/tesis titulada: MODELO DE CONSUMO ENERGÈTICO BASADO EN APRENDIZAJE AUTOMATICO PARA MIDDLEWARE EMBEBIDO DE IOT

Presentado por: HANAN RONALDO QUISPE CONDORI DNI N° 75490126

presentado por: ..... DNI N°: .....

Para optar el título profesional/grado académico de INGENIERO ELECTRÓNICO

Informo que el trabajo de investigación ha sido sometido a revisión por 3 veces, mediante el Software Antiplagio, conforme al Art. 6° del ***Reglamento para Uso de Sistema Antiplagio de la UNSAAC*** y de la evaluación de originalidad se tiene un porcentaje de 10 %.

Evaluación y acciones del reporte de coincidencia para trabajos de investigación conducentes a grado académico o título profesional, tesis

Porcentaje	Evaluación y Acciones	Marque con una (X)
Del 1 al 10%	No se considera plagio.	X
Del 11 al 30 %	Devolver al usuario para las correcciones.	
Mayor a 31%	El responsable de la revisión del documento emite un informe al inmediato jerárquico, quien a su vez eleva el informe a la autoridad académica para que tome las acciones correspondientes. Sin perjuicio de las sanciones administrativas que correspondan de acuerdo a Ley.	

Por tanto, en mi condición de asesor, firmo el presente informe en señal de conformidad y **adjunto** las primeras páginas del reporte del Sistema Antiplagio.

Cusco, 20 de AGOSTO de 2025

Firma

Post firma Milton Jhon Velasquez Curo

Nro. de DNI 41053102

ORCID del Asesor 0000-0001-7521-8846

Se adjunta:

1. Reporte generado por el Sistema Antiplagio.
2. Enlace del Reporte Generado por el Sistema Antiplagio: **oid:** 27259:484709012

Hanan Quispe

Tesis_QC.pdf

Universidad Nacional San Antonio Abad del Cusco

Detalles del documento

Identificador de la entrega
trn:oid:::27259:484709012

Fecha de entrega
19 ago 2025, 6:50 p.m. GMT-5

Fecha de descarga
19 ago 2025, 6:53 p.m. GMT-5

Nombre de archivo
Tesis_QC.pdf

Tamaño de archivo
9.7 MB

148 Páginas

30.694 Palabras

164.924 Caracteres

10% Similitud general

El total combinado de todas las coincidencias, incluidas las fuentes superpuestas, para ca...

Filtrado desde el informe

- Bibliografía
- Texto citado
- Texto mencionado
- Coincidencias menores (menos de 8 palabras)

Fuentes principales

7% Fuentes de Internet
2% Publicaciones
6% Trabajos entregados (trabajos del estudiante)

Marcas de integridad

N.º de alerta de integridad para revisión

Caracteres reemplazados
194 caracteres sospechosos en N.º de páginas
Las letras son intercambiadas por caracteres similares de otro alfabeto.

Los algoritmos de nuestro sistema analizan un documento en profundidad para buscar inconsistencias que permitirían distinguirlo de una entrega normal. Si advertimos algo extraño, lo marcamos como una alerta para que pueda revisarlo.

Una marca de alerta no es necesariamente un indicador de problemas. Sin embargo, recomendamos que preste atención y la revise.

DEDICATORIA

Esta tesis está dedicada a tres personas fundamentales en mi vida, cuyo respaldo fue esencial para alcanzar las metas y propósitos que me tracé.

A mi madre *Corina*, quien me brindó su amor y esfuerzo incondicional a lo largo de mi formación profesional.

A mi tío *Jose*, por su apoyo constante e incondicional tanto en los momentos de alegría como en los de dificultad.

A mi hermano *Rodolfo*, quien me enseñó el hermoso mundo de la programación.

Hanan Ronaldo Quispe Condori

AGRADECIMIENTOS

Agradezco a la doctora *Denisse Muñante* por impulsar la colaboración internacional que acogió la presente tesis, así como también agradezco todos los consejos que me brindó.

Agradezco a los doctores *Denis Conan* y *Sophie Chabridon* por todas las asesorías en las que me brindaron todo su conocimiento especializado en el área de middleware de IoT. Aprendí enormemente durante estas sesiones, que constituyó un pilar importante en la realización de este trabajo de investigación.

Agradezco a mi asesor Ing. *Milton Velasquez Curo* por su apoyo constante durante todas las etapas del desarrollo de esta tesis.

Hanan Ronaldo Quispe Condori

Resumen

La presente investigación introduce un modelo para la estimación del consumo de energía para el middleware embebido IoTVar, basado en aprendizaje automático y desarrollado a partir de mediciones en una Raspberry Pi 4b. La principal contribución del modelo es la incorporación del tiempo como variable de estimación, aspecto no considerado por el método del estado del arte.

La fase inicial correspondiente al desarrollo de este modelo fue realizar las mediciones de consumo energético de IoTVar ejecutándose en la Raspberry Pi 4b. Dichas mediciones requirieron del desarrollo de un prototipo capaz de medir señales de voltaje y corriente en DC. Posteriormente, estas señales se utilizaron para calcular la potencia instantánea, valor que finalmente se transformó en lecturas de consumo energético (en joules). En la construcción del prototipo se incorporó el diseño de una PCB y el desarrollo del firmware destinado al microcontrolador RP2040, que funcionó como componente central.

Una vez listo el prototipo, se realizaron una serie de experimentos con el objetivo de capturar la dinámica del consumo energético en distintas configuraciones de IoTVar. Estos experimentos, junto con las mediciones de consumo energético, permitieron recopilar un conjunto de datos que fue utilizado para construir el modelo basado en aprendizaje automático.

Posteriormente, el dataset obtenido se utilizó para la fase de entrenamiento de una red neuronal feed-forward, mediante la cual se representó la relación entre las variables internas de IoTVar y la variación en la tasa de consumo energético. Esta red neuronal fue evaluada utilizando métricas correspondientes a la regresión cuantil no paramétrica.

Finalmente, el último capítulo evalúa el rendimiento del modelo desarrollado, comparándolo primero con mediciones reales y luego con el método del estado del arte. Con un periodo de actualización de 1 segundo, ambos modelos presentan una precisión similar (RMSE de 2,35 frente a 2,27). No obstante, al aumentar el periodo a 20 segundos, el modelo del estado del arte alcanza un RMSE de 637,34, mientras que el modelo propuesto se mantiene en 2,76. Dado que un RMSE menor indica mejor ajuste, se concluye que el modelo desarrollado no solo es competitivo, sino que también generaliza mejor al incluir el tiempo como variable explícita.

Palabras Claves: *Consumo energético, Middleware, IOT, Modelo de aprendizaje automático*

Abstract

This thesis presents an energy consumption estimation model for the IoTVar middleware architecture. The model is based on machine learning and was developed using energy consumption measurements of IoTVar running on a Raspberry Pi 4b. The primary novelty of the proposed model is the use of time as a variable for estimation, which is not addressed in contemporary state-of-the-art methods.

The process started with measuring the energy consumption of IoTVar running on a Raspberry Pi 4b, which necessitated the design of a prototype for acquiring voltage and current data. These signals were then used to calculate instantaneous power, which was finally converted into energy consumption readings (in joules). The development of this prototype involved designing a printed circuit board and firmware for the RP2040 microcontroller, which was used as the brain of the prototype.

Once the prototype was ready, a series of experiments were conducted to capture the dynamics of energy consumption for different configurations of IoTVar. These experiments and the energy consumption measurements were used to collect a dataset that was used to build the machine learning-based model.

Subsequently, the gathered dataset served to train a feed-forward neural network, which was applied to represent the relationship between IoTVar's internal variables and the variations in energy consumption. This neural network was evaluated using metrics corresponding to non-parametric quantile regression.

Finally, the last chapter evaluates the performance of the developed model, comparing it first with real measurements and then with the state-of-the-art method. For an update period of 1 second, both models exhibit similar accuracy (RMSE of 2,35 vs. 2,27). However, when the update period is increased to 20 seconds, the state-of-the-art model's RMSE rises sharply to 637,34, while the proposed model maintains a low RMSE of 2,76. Since a lower RMSE indicates a better fit, it is concluded that the developed model is not only competitive but also more general, as it explicitly incorporates time as a variable.

Keywords: *Energy consumption estimation, IOT middleware, Machine learning model*

Índice general

Índice de figuras

Índice de tablas

Glosario

ADC SAR SAR (Registro de aproximación sucesiva) ADC (Convertidor analógico a digital) es un tipo de ADC que convierte una señal analógica continua en una señal digital discreta mediante el uso de un algoritmo de búsqueda binaria para aproximar el voltaje de entrada paso a paso. Funciona muestreando y manteniendo la entrada analógica, luego realizando aproximaciones sucesivas usando un DAC (convertidor digital a analógico) y un comparador para determinar cada bit de la salida digital, proporcionando una alta resolución, velocidad moderada y potencia.Los ADC SAR, que normalmente ofrecen una resolución de 8 a 16 bits, se utilizan ampliamente en osciloscopios digitales, sistemas de adquisición de datos y aplicaciones de sensores donde se requiere un equilibrio entre velocidad, precisión y eficiencia energética.. 37

Arquitectura Distribuida de IoT Es un sistema de dispositivos interconectados que están diseñados para trabajar juntos para lograr un objetivo común. En este tipo de arquitectura, la inteligencia y la potencia de procesamiento se distribuyen entre múltiples dispositivos, en lugar de centralizarse en una única ubicación. Esto permite una mayor escalabilidad, flexibilidad y confiabilidad en los sistemas de IoT.. IX, 11

Diagrama de Caja Un diagrama de caja es una representación visual que resume un conjunto de datos a través de cinco medidas estadísticas: mínimo, Q1, mediana (Q2), Q3 y máximo. La caja cubre el rango intercuartílico y en su interior se marca la mediana. Los bigotes se extienden hacia los valores extremos que se encuentran dentro de un rango establecido, mientras que los datos atípicos aparecen como puntos fuera de ellos. Este gráfico es útil para analizar la dispersión, la forma de la distribución y la existencia de valores anómalos..

69

DMA DMA (Acceso directo a la memoria) es una característica de los sistemas informáticos que permite a los subsistemas de hardware transferir datos directamente entre la memoria y los periféricos sin involucrar a la CPU, lo que mejora significativamente la eficiencia de la transferencia de datos y libera a la CPU para realizar otras tareas. El controlador DMA gestiona estas transferencias manejando direcciones de memoria e intercambio de datos, operando a través de una secuencia de operaciones de lectura y escritura que ocurren independientemente de la CPU. Esta capacidad es crucial para aplicaciones de transferencia de datos de alta velocidad, como operaciones de unidades de disco, transmisión de audio y video y redes, donde es esencial minimizar la intervención de la CPU y maximizar el rendimiento de los datos.. 37

IoT El llamado Internet de las Cosas, cuyo término en inglés es Internet of Things y cuya abreviatura es IoT, hace referencia a un ecosistema de objetos físicos equipados con sensores, programas y diversas tecnologías que les permiten conectarse entre sí y con otros sistemas mediante Internet para intercambiar información.. 9

IoTVar IoTVar es un middleware diseñado para simplificar la interacción entre aplicaciones de usuario y plataformas IoT mediante el uso de IoT variables —objetos abstractos que representan datos proporcionados por sensores distribuidos. Este middleware permite a los desarrolladores declarar y gestionar variables IoT de forma transparente, facilitando su descubrimiento, vinculación y actualización sin necesidad de manejar directamente la complejidad de las plataformas subyacentes. IoTVar funciona a través de proxies, los cuales median entre las aplicaciones y las plataformas IoT (en la nube o en el borde), reduciendo significativamente la cantidad de código necesario para acceder a datos del entorno en tiempo real.. 49

Media Corresponde a una medida de tendencia central que resume la información de un conjunto de datos al señalar el valor que ocupa la posición central.. IX, 27

Mediana Se define como el valor que divide a un conjunto de datos ordenados en dos partes iguales.. 26

Middleware Middleware es un término utilizado para describir el software que actúa como mediador entre diferentes aplicaciones o componentes de aplicaciones en una arquitectura distribuida. Permite la comunicación y la conectividad entre estas aplicaciones, simplificando el proceso de creación de aplicaciones distribuidas al proporcionar un marco de mensajería común.. 12

Plataforma de IoT Es un paquete o servicio de software que proporciona herramientas y capacidades integradas para conectar cada "cosa" en un ecosistema de IoT. Incluye funciones como gestión del ciclo de vida del dispositivo, comunicación del dispositivo, análisis de datos, integración y habilitación de aplicaciones. En el contexto de esta investigación se usará la plataforma FIWARE/Orion. 11

Capítulo 1

Aspectos Generales

1.1. Título

Modelo de Consumo Energético Basado en Aprendizaje Automático para Middleware Embebido de IoT

1.2. Responsable

Hanan Ronaldo Quispe Condori

1.3. Asesor

Milton Jhon Velasquez Curo

1.4. Ámbito Geográfico

Este trabajo se llevó a cabo en el Laboratorio de Investigación e Innovación en Sistemas de Telecomunicaciones y Tecnologías de la Información (LIISTTI) de la Facultad de Ingeniería Eléctrica, Electrónica, Informática y Mecánica de la Universidad Nacional de San Antonio Abad del Cusco.

1.5. El Problema

1.5.1. Planteamiento del Problema

1.5.1.1. Problemática

El mundo está experimentando un aumento en el uso de energía eléctrica. Según Cozzi et al. [12], la proyección de la demanda mundial de energía es que aumentará un 25 % entre 2017 y 2040 debido a factores como el crecimiento demográfico, la urbanización y la industrialización. Además, acontecimientos recientes como la invasión rusa de Ucrania han llevado al mundo a su primera crisis energética global. La ciudad del Cusco no es ajena a este aumento del consumo de energía eléctrica. Según Quillama [13] para la ciudad del Cusco en el año 2024 existirá un déficit de 3,59MW, dicho déficit representa un aumento del 139,14 % con respecto al déficit de este año 2023 y un 224,4 % con respecto al año 2022.

En respuesta a esta situación se están fijando nuevas políticas para acelerar el uso de energías limpias. Alšauskas et al. [14] predice que las energías renovables a nivel mundial, en particular la solar fotovoltaica y la eólica, ganarán más terreno que cualquier fuente de energía en esta década, representando el 43 % de la generación de electricidad en todo el mundo en 2030, frente al 28 % actual. Actualmente, en el Perú la producción de energía eléctrica proviene en un 40 % de fuente hidráulica, él 60 % restante de fuentes no renovables (petróleo, carbón, gas) [15].

Se espera que Internet de las cosas (IoT) tenga un impacto significativo en el consumo de energía. Según estimaciones actuales [16, 17] para finales de 2025, habrá 100 mil millones de dispositivos IoT en todo el mundo. Además, el consumo global de energía de los dispositivos IoT está aumentando un 20 % anualmente y se prevé que alcance aproximadamente 46 TWh en 2025 [18]. Por tanto, tal y como afirma Almalki et al. [19], el aumento del consumo de energía de IoT es un desafío que debe abordarse para lograr aplicaciones de IoT sostenibles y respetuosas con el medio ambiente.

Los sistemas embebidos que ejecutan aplicaciones que consumen datos de sensores (*aplicaciones de consumo de IoT*) se están volviendo cada vez más populares en muchas áreas [20, 21]. Para abordar el problema de eficiencia energética en sistemas embebidos, se han propuesto di-

ferentes enfoques basados en hardware y software tales como: optimizaciones de procesamiento, optimizaciones de visualización, optimizaciones de comunicación inalámbrica y diseño de batería [22].

Un enfoque prometedor para la eficiencia energética es el *middleware consciente de la energía*. Como lo explica Noureddine et al. [23] La energía consumida por los dispositivos de hardware es causada indirectamente por el software. Por tanto, el middleware para aplicaciones distribuidas puede ayudar a optimizar o reducir el consumo de energía de los dispositivos de hardware, los servicios de software y la propia plataforma. Además, el middleware consciente de la energía podría permitir un consumo eficiente de energía en dispositivos que funcionan con baterías, prolongando la vida útil de la batería y minimizando el impacto ambiental.

Las estrategias actuales de eficiencia energética para el middleware de IoT se centran en la adaptación de la red, la descarga de tareas, la selección de nodos activos y el filtrado de datos en el lado de la aplicación del usuario final que normalmente tiene limitaciones de batería (teléfonos inteligentes, computadoras de placa única) [24]. Además, Borges et al. [10] ha demostrado que se puede emplear un modelo energético en un middleware de IoT para seleccionar enfoques adecuados para mantener el consumo de energía de una aplicación cliente de IoT dentro de un presupuesto energético específico.

En el contexto de modelos energéticos que faciliten estrategias de eficiencia energética en el middleware de IoT, el aprendizaje automático (ML) proporciona técnicas avanzadas para predecir el consumo de energía [25]. Además, las tendencias actuales apuntan a optimizar las redes neuronales que permiten la inferencia de modelos avanzados de aprendizaje automático en dispositivos con batería limitada [26]. Esto representa una oportunidad para mejorar aún más las estrategias de eficiencia energética en el middleware de IoT para dispositivos con batería limitada utilizando modelos de consumo de energía basados en aprendizaje automático.

1.5.1.2. Formulación del Problema

¿Cómo se puede estimar el consumo energético de middleware embebido de IoT para plantear estrategias de eficiencia energética en aplicaciones IoT?

1.6. Antecedentes

Los modelos de consumo de energía son cruciales para comprender y optimizar el uso de energía en los sistemas de IoT. Estos modelos ayudan a predecir patrones de consumo de energía e identificar áreas potenciales de mejora. Los métodos de análisis basados en datos se han utilizado ampliamente para pronosticar el consumo de energía nacional [27]. Investigaciones recientes sobre modelos de consumo de energía para dispositivos IoT con batería limitada apuntan a una amplia variedad de enfoques.

Sabovic et al. [28] desarrolló un modelo de consumo de energía basado en el software Energest para dispositivos IoT. En los experimentos realizados, este método logró resultados de exactitud del 96 %. Para lograr tales resultados, los autores integraron valores reales de consumo de energía de un analizador de potencia de CC N6705B a la plataforma Energest.

Kesrouani et al. [29] realizó un estudio sobre cómo los diferentes lenguajes de programación y compiladores afectan el consumo de energía en una Raspberry Pi 3b. Sus experimentos consistieron en recopilar datos de consumo de energía debido a la ejecución de varios algoritmos conocidos en Java, C, C++ y Python. Además, los autores propusieron un modelo de consumo de energía que consiste en una función lineal por partes (ecuación 1.1) que logra un error cuadrático medio de $2,8\,\%$.

$$\begin{aligned} &\text{CPU} \rightarrow [0,50\,\%] && P = 3{,}4495 \times u + 3{,}8563(W) \\ &\text{CPU} \rightarrow [50\,\%,100\,\%] && P = 1{,}4584 \times u + 4{,}7788(W) \end{aligned} \tag{1.1}$$

Donde u es un parámetro que depende del número de ciclos del CPU y se calcula mediante la siguiente ecuación $u[t] = \frac{c_{busy}[t]-c_{busy}[t-1]}{c_{total}[t]-c_{total}[t-1]}$.

Bhattacharya et al. [30] estudió el uso de redes neuronales profundas para modelar el consumo de energía en sensores IoT desplegados en playas, utilizando el conjunto de datos Chicago Park Beach. El modelo desarrollado se empleó para predecir la duración de la batería de una red de sensores integrados con capacidad energética limitada, la misma que fue utilizada para construir dicho conjunto de datos.

Kanso et al. [31] propuso un enfoque híbrido que utiliza mediciones del consumo de energía tomadas con un medidor de potencia y desarrolló una herramienta basada en software para generar automáticamente modelos de consumo de energía para diferentes configuraciones de hardware con recursos limitados. Los autores probaron su enfoque en diferentes dispositivos Raspberry Pi y descubrieron que su método es capaz de predecir el consumo de energía de toda la línea de dispositivos Raspberry Pi con márgenes de error tan bajos como $0,3\,\%$.

Borges et al. [10] creó un modelo matemático para evaluar el consumo de energía debido a las interacciones necesarias para actualizar los datos de los sensores requeridos por las aplicaciones de consumo de IoT en la arquitectura de middleware IoTVar [32]. Este modelo tiene en cuenta diferentes contribuciones de energía de variables dentro de IoTVar, el estado de la red y el procesamiento de la CPU. Los autores utilizaron datos de consumo de energía, tomados con un vatímetro, para calibrar las constantes del modelo y terminaron con una función (ecuación 1.2) dependiente de las variables internas de IoTVar que permiten el uso del modelo en tiempo de ejecución.

$$E = \sum_{i=0}^{nb_G} \frac{C_V \times nb_{V_{G_i}} + C_{net} \times M_{netS} \times M_{net1} + C_{cpu}}{R_{G_i}} \tag{1.2}$$

Dónde:

- $nb_{V_{G_i}}$, R_{G_i} y nb_G son las variables internas de IoTVar.
- C_V, $C_{net} \times M_{netS} \times M_{net1} + C_{cpu}$ son constantes que dependen del estado de la red y del procesamiento de la CPU.

De la revisión del estado del arte, el modelo energético propuesto por Borges et al. [10] es el único modelo energético reportado específico para el middleware de IoT.

1.7. Justificación

La proliferación de dispositivos de Internet de las cosas (IoT) ha dado lugar a una era de conectividad y automatización sin precedentes. Las aplicaciones de IoT abarcan varios dominios, desde hogares inteligentes y automatización industrial hasta atención médica y monitoreo

ambiental. Sin embargo, esta proliferación también ha planteado una preocupación crítica: el uso eficiente de la energía eléctrica de las aplicaciones IoT en sistemas embebidos, ya que, estos normalmente funcionan con baterías.

El middleware embebido para IoT cumple un papel crucial al facilitar la comunicación y el intercambio de datos entre dispositivos. No obstante, su consumo energético representa aún un desafío significativo. Un consumo subóptimo no solo reduce la vida útil operativa de los dispositivos IoT, sino que también incrementa los costos asociados a las aplicaciones y afecta negativamente la sostenibilidad ambiental, al generar una mayor cantidad de residuos electrónicos.

A medida que los dispositivos de IoT se integran cada vez más en nuestra vida diaria y procesos industriales, la demanda de soluciones energéticamente eficientes es primordial. Las estrategias actuales de eficiencia energética para el middleware de IoT, enfrentan limitaciones. A menudo se basan en modelos energéticos simples que pueden no capturar con exactitud la dinámica del consumo de energía de diversas aplicaciones y hardware de IoT.

El desarrollo exitoso de un modelo de consumo de energía basado en aprendizaje automático para middleware embebido de IoT tendría implicaciones de gran alcance:

- Duración prolongada de la batería: al optimizar el consumo de energía, los dispositivos IoT pueden funcionar durante períodos prolongados sin necesidad de reemplazar la batería con frecuencia.
- Costos de energía reducidos: un menor consumo de energía se traduce en ahorros de costos para las implementaciones de IoT, particularmente en aplicaciones industriales a gran escala.
- Sostenibilidad ambiental: la disminución de la huella energética de los dispositivos IoT se alinea con los esfuerzos globales para reducir las emisiones de carbono y promover la sostenibilidad.

En este trabajo se plantea un modelo de consumo de energético para middleware embebido de IoT. Evaluar la viabilidad y exactitud de las estimaciones basadas en aprendizaje automático para el consumo de energía permitirá guiar el desarrollo de estrategias de eficiencia energética

para middleware embebido de IoT, equilibrando la precisión de la estimación con la eficiencia energética.

Este trabajo abordará el tema de la eficiencia energética en el middleware embebido de IoT mediante el desarrollo de un modelo de consumo de energía basado en aprendizaje automático. El objetivo es lograr un equilibrio entre estimaciones exactas y consideraciones prácticas de implementación. Esta investigación está preparada para avanzar en el campo de la eficiencia energética de IoT y contribuir al desarrollo sostenible de la arquitectura de IoT.

1.8. Objetivos

1.8.1. Objetivo General

Implementar un sistema de estimación de consumo energético basado en datos para un middleware de IoT ejecutándose en un sistema embebido.

1.8.2. Objetivos Específicos

- Implementar una arquitectura de middleware de IoT en un sistema embebido.
- Medir el consumo energético de un sistema embebido ejecutando un middleware de IoT.
- Recopilar un dataset con los datos de consumo de energía y los estados de las variables del middleware implementado.
- Proponer una estructura matemática para estimar el consumo de energía del middleware embebido de IoT.
- Evaluar la estructura matemática propuesta en un sistema embebido mediante parámetros o indicadores de medición.

1.8.3. Alcances

- Se recopilará datos de consumo energético de middleware embebido de IoT en un Raspberry Pi 4B.

- Se desarrollará un modelo energético basado en aprendizaje automático.
- Se comparará la exactitud del modelo propuesto con métodos del estado del arte.

1.8.4. Limitaciones

- El middleware se implementará en un Raspberry Pi 4B lo que limitará el modelo propuesto a dicho hardware y a la configuración base del Raspberry Pi usado.
- El dataset recopilado para la implementación del modelo contendrá datos de consumo energético de la configuración base del Raspberry Pi 4B por lo que su utilización es exclusiva a este modelo.
- Todas las evaluaciones del modelo energético se realizarán en una arquitectura de middleware en específico por lo que el modelo energético estará limitado a la estimación de consumo energético en dicha arquitectura de middleware. Esta arquitectura de middleware fue desarrollada por el grupo de Sistemas Distribuidos e Ingeniería de Software (DiSSEM) del departamento de ciencias de computación del Telecom SudParis.

Capítulo 2

Marco Teórico

2.1. Internet de las Cosas

A pesar del entusiasmo mundial que rodea al Internet de las cosas (IoT), no existe una definición única y universalmente reconocida para este término. Diferentes grupos emplean definiciones distintas para retratar o defender sus propias perspectivas sobre el significado de IoT y sus características clave. La literatura contiene una variedad de definiciones, tales como:

1. Sengupta et al. [33]: *"IoT es un grupo de objetos estáticos y/o móviles interconectados, como dispositivos equipados con módulos de comunicación, sensores y actuadores conectados a través de Internet".*

2. Siboni et al. [34]: *"el IoT es un ecosistema global de tecnologías de la información y la comunicación destinado a conectar cualquier tipo de objeto (cosa), en cualquier momento y en cualquier lugar, entre sí y con Internet".*

3. Nord et al. [35]: *"una interconexión de máquinas y dispositivos a través de Internet, que permite la creación de datos que pueden arrojar luz sobre el rendimiento analítico y respaldar nuevas tecnologías".*

4. Luthra et al. [36]: *"IoT se puede utilizar para describir un sistema de objetos físicos que tienen comunicación independiente entre ellos".*

5. Asplund and Nadjm-Tehrani [37]: *"los sistemas embebidos conectados a Internet se pueden actualizar y adaptar a las necesidades cambiantes según la demanda, se puede recopilar información útil inmediatamente desde áreas geográficas remotas y el diagnóstico de fallas y los reinicios del sistema se pueden hacer más eficientes y rentable al no tener que enviar técnicos a lugares remotos".*

6. Chen et al. [38]: *"IoT conecta dispositivos sensores a Internet con el fin de intercambiar información".*

7. Al-Kadhim and Al-Raweshidy [39]: *"el IoT comprende numerosos nodos sensores con capacidades limitadas de procesamiento, almacenamiento y batería".*

De todas las definiciones encontradas en la literatura se puede decir que IoT representa una fuerza transformadora en la tecnología, que conecta el mundo físico con el ámbito digital. Sus beneficios potenciales en la automatización y toma de decisiones basadas en datos son significativos, por lo que abordar las preocupaciones de seguridad y energía es crucial para su crecimiento e impacto sostenibles.

2.1.1. Arquitectura de IoT

Un sistema de Internet de las cosas (IoT) es un ecosistema multifacético compuesto por varios elementos integrales, como se muestra en la Figura 2.1:

- Dispositivos IoT: Incluyen sensores y actuadores. Los sensores detectan información externa o fenómenos físicos y los convierten en señales eléctricas que representan valores medidos. Los actuadores, por otro lado, transforman entradas eléctricas en acciones físicas.

- Aplicaciones de usuario final: En el contexto de esta investigación, nuestro enfoque principal radica en las aplicaciones de consumo de IoT para dispositivos con batería limitada. Estas aplicaciones sirven como interfaz de usuario final del sistema IoT, aprovechando los datos generados por los sensores para diversos fines. Son los medios a través de los cuales los individuos y las organizaciones interactúan y se benefician de la información generada por IoT. Estas aplicaciones pueden variar desde sistemas de control doméstico inteligente hasta herramientas de optimización y monitoreo industrial.

- Plataforma de IoT, Gateways y Middleware IoT: Los entornos IoT se caracterizan por una gran diversidad de componentes de hardware y software. Para cerrar la brecha entre estos elementos dispares y facilitar una comunicación fluida con los dispositivos de IoT, confiamos en intermediarios estandarizados. Estos incluyen plataformas de IoT, puertas de enlace y middleware de IoT. Las plataformas de IoT sirven como sistemas de gestión generales que coordinan las interacciones de los dispositivos y los flujos de datos. Las puertas de enlace actúan como intermediarios que conectan los dispositivos de IoT a la red más amplia, lo que permite una transmisión de datos eficiente. El middleware de IoT abarca las capas de software que facilitan la comunicación y el procesamiento de datos dentro del ecosistema de IoT.

En esta arquitectura de IoT, las aplicaciones de IoT se distribuyen entre dispositivos de IoT, que están equipados con sensores y actuadores, así como en dispositivos de usuario final. Estos dispositivos funcionan de forma autónoma y están interconectados con otros nodos. Esta interconexión puede ocurrir directamente a través de una plataforma de IoT o mediante puertas de enlace. Tanto las plataformas como las puertas de enlace de IoT están conectadas a Internet, lo que garantiza que los datos recopilados de los dispositivos de IoT sean accesibles para las aplicaciones de IoT del usuario final.

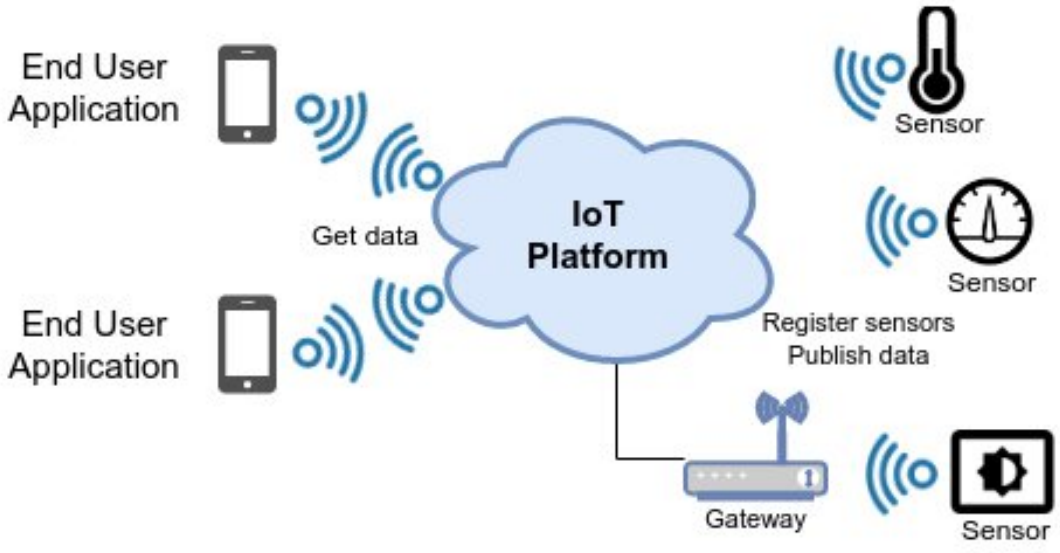

Figura 2.1: Arquitectura Distribuida de IoT [1]

2.1.2. Middleware Embebido de IoT

El Middleware de IoT desempeña un papel fundamental y dinámico a la hora de abordar la heterogeneidad inherente dentro de los sistemas de IoT. Al abstraer las complejidades específicas de los dispositivos de las aplicaciones y los usuarios, no solo simplifica el proceso de desarrollo, sino que también fomenta la interoperabilidad y garantiza una comunicación fluida entre dispositivos [40, 41, 42].

En el panorama en constante evolución de IoT, caracterizado por una multitud de variaciones de hardware y software, el middleware de IoT se ha convertido en un habilitador fundamental. Estas soluciones de middleware han evolucionado para adaptarse a dispositivos con diversas capacidades, funcionalidades y protocolos de red, proporcionando un marco versátil para el desarrollo de IoT [40, 43, 44].

En términos generales, el middleware de IoT sirve como una capa de software estratégicamente ubicada entre las capas de aplicación y detección física (como se ilustra en la Figura 2.2). Su función es doble: abstraer la complejidad de la interfaz con dispositivos físicos (Physical Sensing Layer) y gestionar eficientemente los datos que generan, permitiendo así a los desarrolladores centrarse más en los aspectos creativos del desarrollo de aplicaciones (Application Layer).

El middleware de IoT cumple esta doble función al:

- Facilitar interacciones fluidas entre dispositivos y usuarios, gestionando eficazmente la interoperabilidad tanto para los componentes de hardware como de software (IoT Middleware Layer).
- Implementación de técnicas de virtualización de dispositivos (Device Discovery).
- Agregación y procesamiento de flujos de datos (Gestión de contexto) [45, 46].

El middleware de IoT actúa como eje dentro de la arquitectura de IoT, permitiendo la comunicación entre dispositivos y aplicaciones de IoT mientras abstrae y maneja hábilmente las complejidades del sistema subyacente. Esta capa de middleware dinámica y adaptable desempeña un papel vital a la hora de impulsar la innovación y la eficiencia en el ecosistema de IoT en rápida expansión.

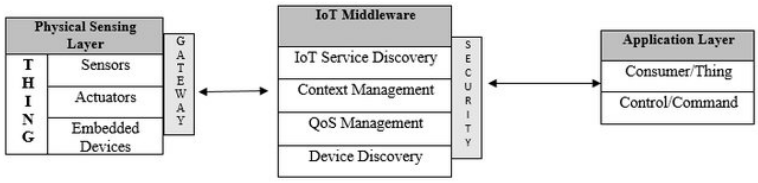

Figura 2.2: Arquitectura básica de un sistema IoT [2]

2.1.3. Plataforma de IoT

Una plataforma de IoT es un marco de software que simplifica el desarrollo, implementación y gestión de aplicaciones de IoT. Maneja funciones críticas como la recopilación, el almacenamiento, el análisis y la visualización de datos, lo que permite a los desarrolladores crear aplicaciones escalables y seguras que utilizan dispositivos conectados.

Como lo destaca Razzaque et al. [46], las aplicaciones modernas, incluidas aquellas en edificios inteligentes, ciudades inteligentes, agricultura, logística y manufactura, comparten requisitos comunes. Estos incluyen interactuar con dispositivos IoT a través de varios protocolos y satisfacer demandas extrafuncionales como seguridad, baja latencia, eficiencia energética, disponibilidad, confiabilidad y alto rendimiento.

Para abordar estos desafíos, la utilización de plataformas de IoT para implementar aplicaciones de IoT se ha convertido en una tendencia destacada. Las plataformas de IoT ofrecen servicios para implementar y ejecutar aplicaciones sobre una base de hardware y/o software [47]. Son esenciales para gestionar la diversidad de hardware y software en entornos de IoT, lo que hace posible que múltiples aplicaciones implementen y compartan estas plataformas. En resumen, las plataformas de IoT desempeñan un papel fundamental a la hora de simplificar el desarrollo y la gestión de aplicaciones de IoT y, al mismo tiempo, se adaptan a las complejidades del panorama de IoT.

2.1.4. FIWARE/Orion

Para este trabajo se considera la plataforma FIWARE IoT. FIWARE es una plataforma de IoT versátil y de código abierto cuyo desarrollo ha sido financiado por la Unión Europea, lo que ha permitido su consolidación como una iniciativa respaldada por esta entidad. Ofrece un conjunto de componentes interoperables conocidos como Generic Enablers (GE). Estas GES

simplifican el desarrollo de sistemas en diversos dominios de aplicaciones, incluida la gestión de contexto, el control de dispositivos, el almacenamiento de datos, el procesamiento de eventos, la seguridad y la creación de paneles.

En el núcleo de FIWARE se encuentra Orion Context Broker, un componente central para la gestión del contexto en aplicaciones de IoT. El compromiso de FIWARE con la apertura, la estandarización y la interoperabilidad lo ha convertido en un recurso valioso para los desarrolladores de IoT en todo el mundo.

Los Habilitadores Genéricos de FIWARE atienden una variedad de funcionalidades esenciales de IoT. Estos incluyen una gestión eficiente de entidades contextuales para la comprensión del estado en tiempo real, una gestión optimizada de dispositivos para un control eficaz de los dispositivos, un sólido almacenamiento de datos históricos, soporte para arquitecturas basadas en eventos, medidas de seguridad integrales y la creación de paneles personalizables para la visualización de datos.

Orion Context Broker permite acceder, actualizar y responder sin problemas a datos en tiempo real provenientes de diversas fuentes, actuando como el núcleo central de gestión de contexto dentro del ecosistema FIWARE. Más allá de sus raíces europeas, FIWARE ha ganado reconocimiento a nivel mundial, contando con una próspera comunidad internacional de usuarios que contribuye activamente a su desarrollo e innovación continuos.

En resumen, FIWARE se presenta como una plataforma de IoT versátil y con buen soporte que ofrece un sólido conjunto de herramientas y componentes. Su compromiso con la apertura y la interoperabilidad lo posiciona como una alternativa ventajosa para proyectos de IoT, pues brinda a los desarrolladores la posibilidad de diseñar soluciones escalables y flexibles en distintos ámbitos de aplicación.

2.1.5. Middleware IoTVar

IoTVar es una biblioteca de middleware de IoT diseñada para simplificar las interacciones entre las aplicaciones de IoT del cliente y las plataformas de IoT de gestión de contexto. Abstrae el proceso, reduciendo los esfuerzos de codificación para descubrir y utilizar sensores virtualizados. IoTVar utiliza proxies, es decir, componentes intermedios que representan variables de IoT, para

manejar sus interacciones con las plataformas, facilitando así una comunicación más simple y eficiente [32].

Además, IoTVar ofrece estrategias de optimización de energía para dispositivos que funcionan con baterías a través de su nueva *capa de energía* que se muestra en la figura 2.3 [10].

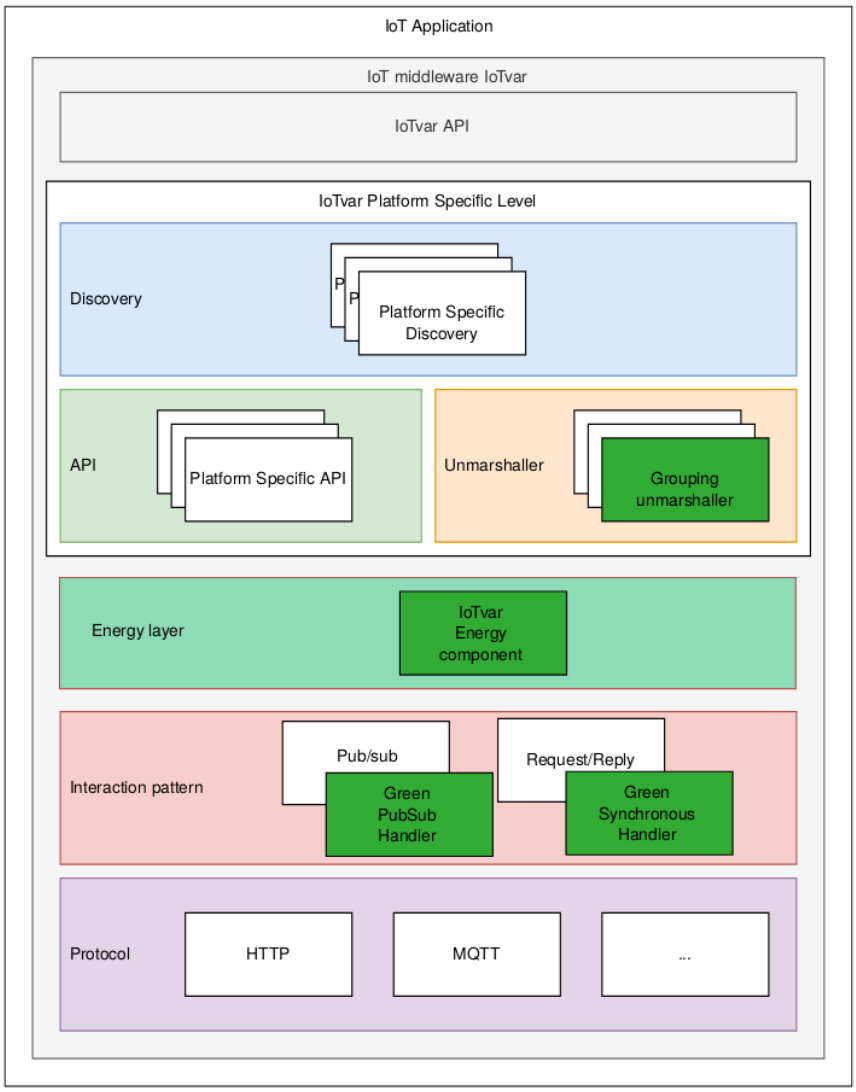

Figura 2.3: Componentes energéticos en la arquitectura IoTVar

2.2. Linux en sistemas embebidos

Linux en sistemas embebidos o Linux embebido hace alusión a la implementación de Linux como sistema operativo en entornos embebidos, que son sistemas informáticos especializados que realizan funciones dedicadas dentro de sistemas mecánicos o eléctricos más grandes. A diferencia de las computadoras de uso general, los embebidos generalmente están diseñados para realizar

tareas específicas y, a menudo, disponen de capacidades reducidas en cuanto a procesamiento, memoria y espacio de almacenamiento. Características de Linux embebido:

- Personalización: una de las principales ventajas de usar Linux en sistemas embebidos es su alto grado de personalización. Los desarrolladores pueden adaptar el kernel de Linux y la pila de software que lo acompaña para que se ajusten a los requisitos específicos del hardware y la aplicación, asegurando un uso eficiente de los recursos [48].
- Código abierto: Linux es un sistema operativo de código abierto, lo que significa que su código fuente está disponible gratuitamente para su modificación y distribución. Esta apertura permite a los desarrolladores modificar el sistema operativo para satisfacer las necesidades únicas de sus aplicaciones integradas sin costos de licencia [49].
- Robustez y estabilidad: Linux es conocido por su robustez y estabilidad, que son atributos críticos para los sistemas embebidos que a menudo operan en entornos que requieren alta confiabilidad [50].
- Amplio soporte de hardware: Linux admite una amplia gama de plataformas de hardware, desde pequeños microcontroladores hasta potentes procesadores multinúcleo. Esta versatilidad lo hace adecuado para una variedad de aplicaciones integradas, desde dispositivos simples como enrutadores hasta sistemas complejos como unidades de control de automóviles [51].

2.2.1. Componentes de sistemas Linux Embebido

Un sistema Linux embebido suele estar compuesto por varios elementos esenciales:

1. **Núcleo de Linux**: actúa como el corazón del sistema operativo, gestionando los recursos del hardware, la ejecución de procesos, la memoria y otras funciones clave.
2. **Bootloader o cargador de arranque**: programa ligero que se encarga de preparar el hardware y transferir el núcleo de Linux a la memoria durante el inicio del sistema.
3. **Sistema de archivos raíz**: alberga las bibliotecas esenciales, los ficheros de configuración y los ejecutables necesarios para que el sistema pueda operar correctamente.

4. **Drivers o controladores de dispositivos**: módulos de software diseñados para gestionar la comunicación entre el sistema operativo y el hardware instalado.

2.2.2. Herramientas y procesos de desarrollo

El desarrollo de un sistema Linux embebido implica varias herramientas y procesos especializados. La compilación cruzada se usa comúnmente, donde el código se compila en un sistema host (generalmente un escritorio o servidor potente) para ejecutarse en el hardware embebido de destino. Herramientas como el Proyecto Yocto proporcionan marcos para crear distribuciones de Linux personalizadas adaptadas a hardware embebido específico [52].

2.2.3. Aplicaciones de Linux embebido

Linux embebido se implementa en diferentes aplicaciones a través de varios campos de la industria, que incluyen:

- *Electrónica de consumo*: televisores inteligentes, decodificadores y dispositivos de automatización del hogar.
- *Automotriz*: sistemas de información y entretenimiento en el vehículo, sistemas avanzados de asistencia al conductor (ADAS) y unidades de control del motor.
- *Automatización industrial*: Robótica, sistemas de control de procesos y dispositivos IoT industriales.
- *Redes*: enrutadores, conmutadores y dispositivos de seguridad de red.
- *Agricultura de precisión*: monitoreo de humedad del suelo, temperatura, riego automatizado y salud de cultivos.
- *Gestión hídrica*: control de niveles en ríos, embalses y canales para riego o prevención de desbordes.

2.3. Medición del Consumo Energético en Sistemas Embebidos

Realizar mediciones exactas del consumo energético es fundamental para maximizar la autonomía de la batería en sistemas embebidos. Los métodos comunes incluyen la medición directa del uso de energía durante diferentes estados del dispositivo, lo que permite modelos de consumo energético realistas.

Dezfouli et al. [53] desarrolló un prototipo de hardware con el fin de cuantificar el consumo de energía de dispositivos IoT que admiten los estándares Wi-Fi IEEE 802.11 e IEEE 802.15.4. Se descubrió que su plataforma de hardware es capaz de medir el consumo de energía con errores de medición de $3,5\,\%$ y $2,5\,\%$ para cada estándar Wi-Fi respectivamente.

2.3.1. Medición de Corriente

2.3.1.1. Amplificadores de Detección de Corriente

Los amplificadores de detección de corriente, también llamados monitores de derivación de corriente, son amplificadores diferenciales especializados con una red de ganancia resistiva adaptada con precisión con las siguientes características [54]:

- Diseñados para monitorear el flujo de corriente midiendo la pérdida de voltaje en el dispositivo sensor, generalmente una resistencia shunt.
- Suelen ser más fáciles de usar, más precisos y menos propensos al ruido.
- Soporta corrientes desde los μA hasta los A.
- Admite de forma nativa voltajes de modo común de –16 V a +80 V y con circuitos adicionales de hasta cientos de voltios.

Parámetros clave:

1. **Rango de modo común**

   Esta especificación define el rango de voltaje CC en la entrada de un amplificador con respecto a tierra. Los amplificadores de detección de corriente generalmente están diseñados

para soportar voltajes de modo común mucho más allá del voltaje de suministro del chip. Por ejemplo, el INA240 es capaz de soportar un voltaje de modo común entre $-4V$ y $+80V$ mientras funciona con un suministro tan bajo como $2{,}7V$.

2. **Voltaje de offset**

   Este es un error de CC diferencial en la entrada del amplificador. Históricamente, para reducir el impacto de los amplificadores con altas compensaciones, se usaban resistencias de derivación de mayor valor para aumentar la caída de voltaje medida. Hoy en día, TI puede ofrecer amplificadores de detección de corriente con compensaciones tan bajas como 10 V, lo que permite mediciones de mayor precisión a corrientes bajas y permite el uso de resistencias de derivación de menor valor para mejorar la eficiencia del sistema.

3. **Ganancia**

   Los amplificadores de detección de corriente vienen con varias opciones de ganancia que tienen un rendimiento sólido sobre las variaciones de temperatura y proceso al integrar una red de ganancia resistiva adaptada con precisión. Las opciones de ganancia para amplificadores de ganancia fija varían de 0,125 V/V a 1000 V/V con errores de ganancia tan bajos como 0,01 %.

4. **Estabilidad de temperatura**

   Los amplificadores de detección de corriente integran el amplificador junto con todas las resistencias de ajuste de ganancia, lo que permite una deriva de temperatura pequeña y unificada. Esto permite mediciones de corriente sólidas en todo el rango de temperatura especificado. La estabilidad de temperatura lograda es una de las ventajas clave que tienen los amplificadores de detección de corriente sobre las implementaciones discretas.

Medición en lado alto. Detección entre bus de fuente y carga. Ventajas del método:

- Capaz de detectar carga en corto a tierra
- La corriente se monitorea directamente desde la fuente.
- Alta inmunidad a las perturbaciones de tierra.

Desafíos del método:

- El alto voltaje del bus limita la disponibilidad de dispositivos de alto voltaje de entrada en modo común.

Ventajas sobre el circuito de detección de corriente discreto:

- Las resistencias de ganancia integradas proporcionan una excelente adaptación para permitir un mayor rendimiento.
- Reducción de los requisitos de espacio en el tablero.
- La arquitectura de entrada única permite que el voltaje de modo común exceda en gran medida el voltaje de suministro del dispositivo.

Un uso reportado en la literatura es PowerScope [3]. Este un método notable que adquiere datos de energía a través de monitores de energía y del sistema.

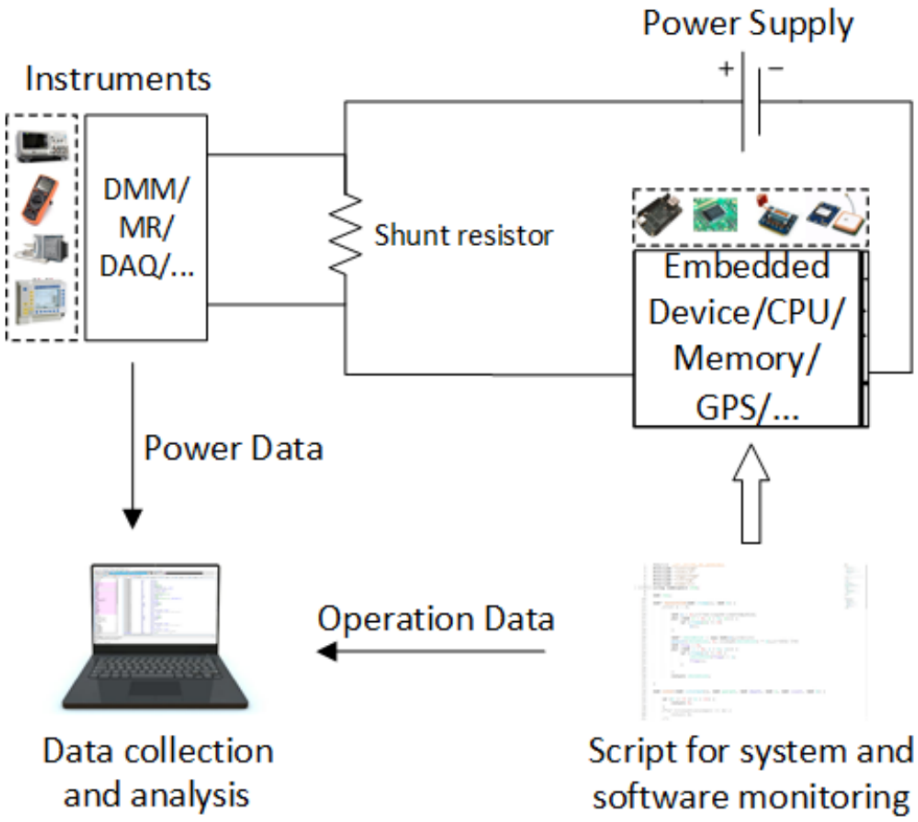

Figura 2.4: Esquema General de Medición de Consumo Energético para Sistemas Embebidos [3]

INA219 El INA219 es un sensor diseñado para medir potencia y corriente en derivación, equipado con una interfaz compatible con los protocolos I2C y SMBus. Este dispositivo es capaz de registrar tanto la caída de voltaje a través de la resistencia de derivación como el voltaje de alimentación del bus, ofreciendo además opciones para configurar el tiempo de conversión y el filtrado de las mediciones. Mediante un valor de calibración ajustable y un multiplicador interno, se obtienen lecturas directas de corriente expresadas en amperios. Asimismo, dispone de un registro adicional que permite calcular la potencia en vatios. Su interfaz admite hasta 16 direcciones programables para comunicación [4].

Características:

- Mide voltajes de bus entre 0V y 26 V.
- Reporta corriente, voltaje y potencia.
- 16 direcciones programables
- Registros de calibración
- Paquetes SOT23-8 y SOIC-8

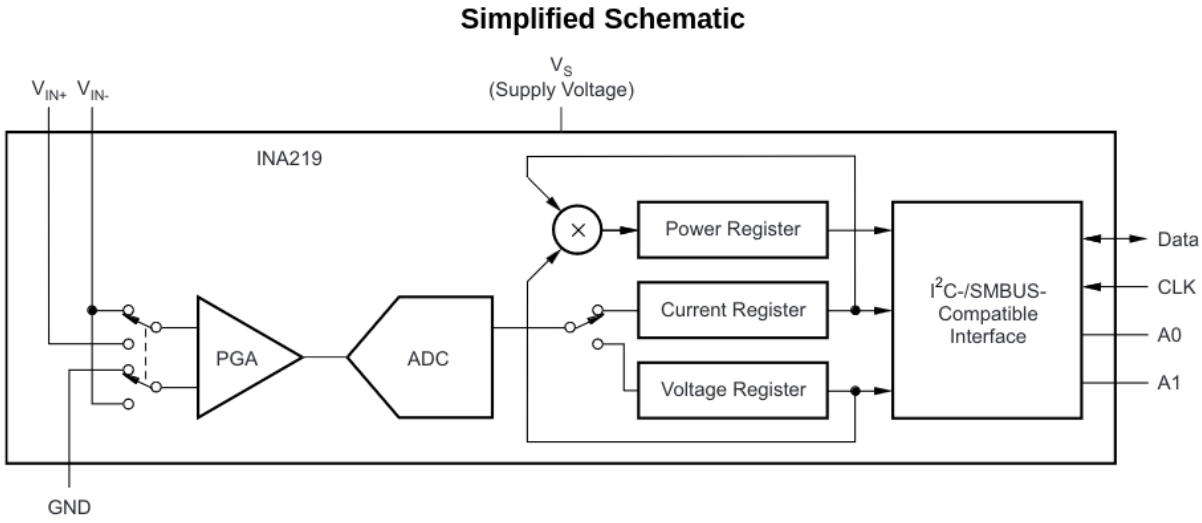

Figura 2.5: Esquema simplificado del INA219 [4]

2.3.2. Medición de Voltaje

La medición de voltaje requiere también de circuitos de acondicionamiento de señal, para propósitos del desarrollo de esta tesis se trabajó bajo la metodología descrita por Carter and Mancini [5]. Dicho autor describe lo siguiente con respecto al diseño de circuitos de acondicionamiento de señal usando amplificadores operacionales alimentados por una sola fuente.

En un amplificador operacional que opera de manera lineal, su función de transferencia puede expresarse mediante la ecuación de una línea recta.

$$y = \pm mx \pm b \tag{2.1}$$

La expresión 2.1 admite cuatro configuraciones posibles, determinadas por los signos de m y b. Como resultado, el conjunto de ecuaciones simultáneas ofrece cuatro variantes, y para cada una de ellas es necesario diseñar un circuito específico que represente dicha forma de la línea recta.

$$\begin{aligned} V_{OUT} &= +\text{m} \times V_{IN} + \text{b} \\ V_{OUT} &= +\text{m} \times V_{IN} - \text{b} \\ V_{OUT} &= -\text{m} \times V_{IN} + \text{b} \\ V_{OUT} &= -\text{m} \times V_{IN} - \text{b} \end{aligned} \tag{2.2}$$

Dado un conjunto de dos puntos de datos para VOUT y VIN, las ecuaciones simultáneas se resuelven para determinar m y b con el fin de que la ecuación que satisfaga los datos dados. Los signos de m y b determinan el tipo de circuito requerido para implementar la solución.

A continuación se muestran los circuitos requeridos para los cuatro casos:

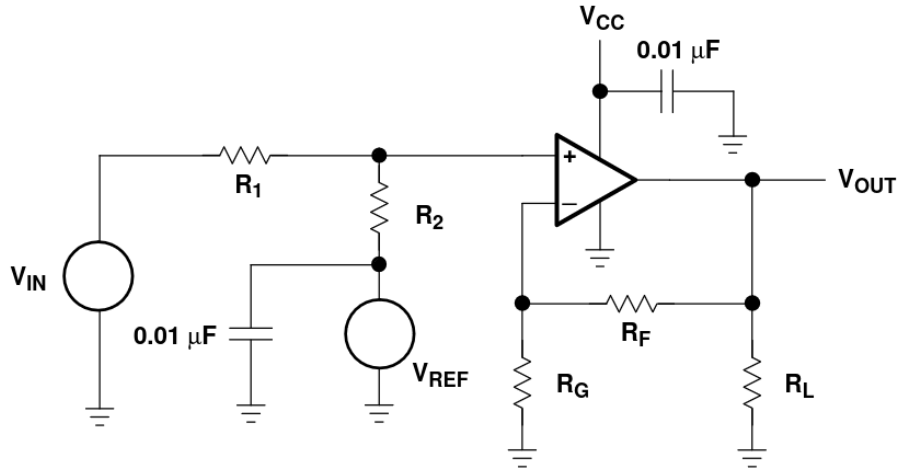

Figura 2.6: Caso 1 $+mV_{IN} + b$ [5]

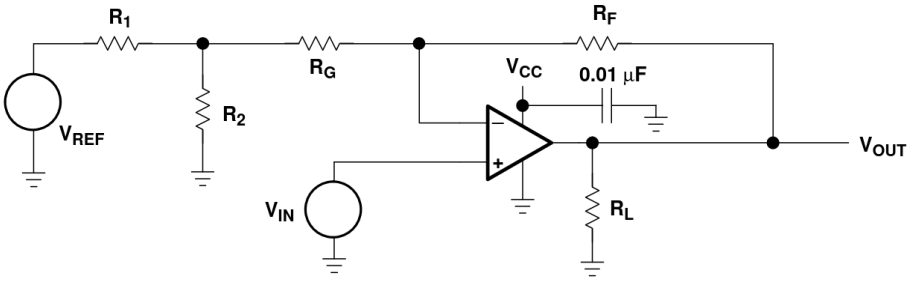

Figura 2.7: Caso 2 $+mV_{IN} - b$ [5]

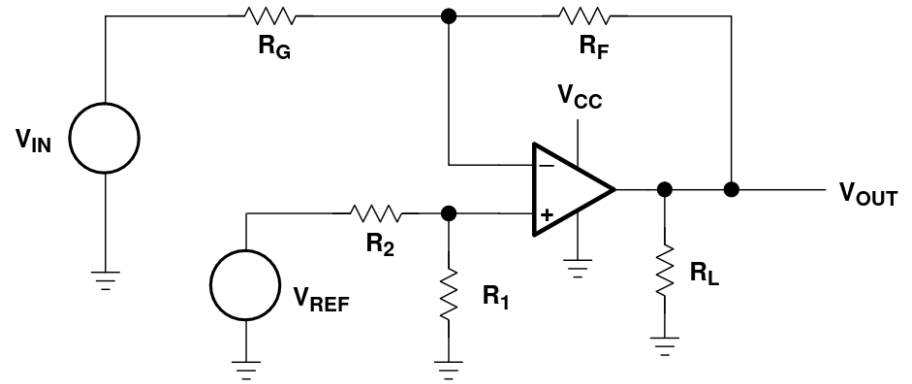

Figura 2.8: Caso 3 $-mV_{IN} + b$ [5]

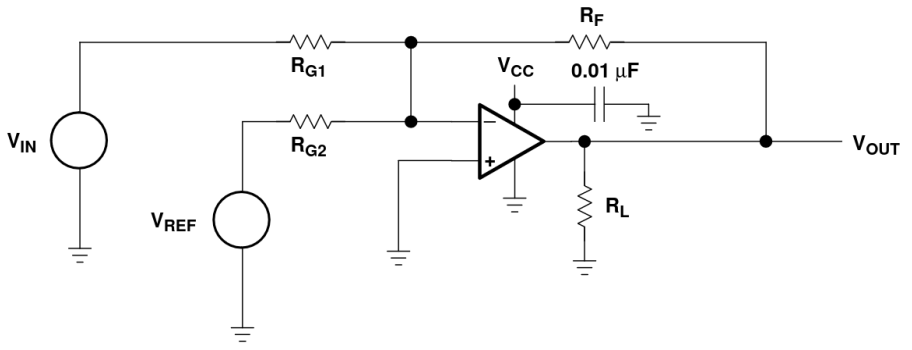

Figura 2.9: Caso 4 $-mV_{IN} - b$ [5]

2.4. Aprendizaje Automático

2.4.1. Funciones de Densidad de Probabilidad

La probabilidad constituye un campo de las matemáticas orientado a modelar y comprender fenómenos inciertos mediante el concepto de aleatoriedad. Un número aleatorio, X, es una variable que puede tomar diferentes valores cada vez que se observa, como ocurre en las mediciones de experimentos. Esto contrasta con una variable regular, x, cuyo valor permanece constante una vez asignado. Así, un número aleatorio representa un nuevo concepto matemático, distinto de los números en la matemática tradicional. Aunque el valor (muestra) de un número aleatorio puede influir de manera específica (incluso determinista) en los procesos, sigue siendo significativo. Dado que un número aleatorio puede cambiar de valor con cada observación, es útil describir las características de su distribución realizando múltiples pruebas. La frecuencia con la que aparecen los distintos valores es el aspecto más relevante a considerar, y esta frecuencia se refleja en la interpretación «frecuentista» de los números aleatorios. Por ejemplo, la función 2.3:

$$P(x) = P(X = x) \tag{2.3}$$

Describe la frecuencia con la que cada valor posible x de una variable discreta X ocurre. Tenga en cuenta que x es una variable regular, no un número aleatorio. El valor de $P(x)$ da la fracción de veces que obtenemos un valor x para el número aleatorio X si sacamos muchas muestras del número aleatorio. De esta definición se deduce que la frecuencia de tener cualquiera de los valores posibles es igual a uno, que es la condición de normalización 2.4.

$$\sum_{x} P(x) = 1 \tag{2.4}$$

En el caso de números aleatorios continuos, tenemos un número infinito de posibles valores para x de modo que la fracción de cada número se vuelve formalmente infinitamente pequeña. Por tanto, es necesario escribir la función de distribución de probabilidad como $P(x) = p(x), dx$, en la cual $p(x)$ corresponde a la función de densidad de probabilidad (fdp).

$$\int_x p(x)dx = 1 \tag{2.5}$$

Un valor de probabilidad finito solo tiene sentido para un cierto rango de números como [55]:

$$P(a < x < b) = \int_{x=a}^{b} p(x)dx \tag{2.6}$$

Hay un número infinito de *fdps* posibles. Sin embargo, se han adoptado algunas formas específicas. Muy útiles para describir algunos procesos específicos y por eso se les han dado nombres. A continuación se presenta una breve lista de ejemplos con algunos discretos y varios continuos. La lista pretende ofrecer una visión general de las distribuciones que a menudo son mencionados en trabajos científicos [55]:

- Distribución de Bernoulli
- Distribución Multinomial
- Distribución Uniforme
- Distribución Normal (Gaussiana)
- Distribución Chi cuadrada

2.4.1.1. Estimación de Densidad de Kernel

La estimación de densidad mediante núcleos, conocida como KDE (Kernel Density Estimation), es una técnica estadística no paramétrica empleada para aproximar la distribución de probabilidad de un conjunto de datos. A diferencia del histograma, que depende de la elección de intervalos discretos, la KDE construye una curva continua y suavizada que refleja mejor la estructura de los datos. Para ello, cada observación se asocia a una función núcleo *a menudo de forma gaussiana* y el grado de suavizado se controla mediante un parámetro denominado bandwidth. Al combinar estos núcleos se obtiene una representación que facilita la identificación de patrones, acumulaciones, asimetrías o posibles múltiples picos dentro de la distribución. Gracias a estas características, este método resulta más flexible y descriptivo que las representaciones tradicionales.

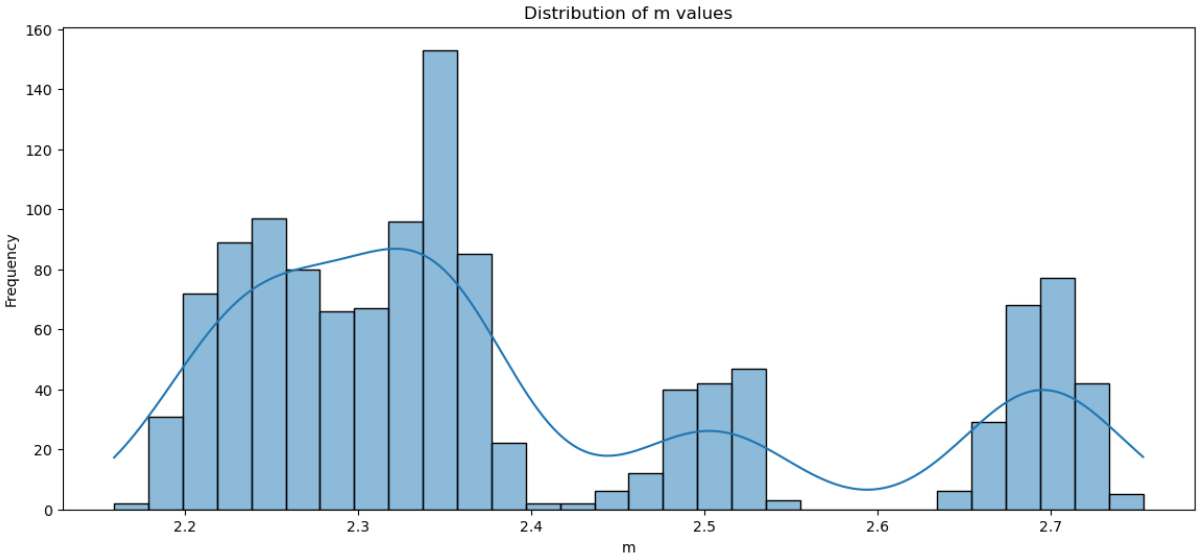

Figura 2.10: Histograma y *KDE* de un Conjunto de Datos

2.4.2. Regresión cuantil no paramétrica

La regresión cuantil es una extensión de la regresión lineal básica, sea un modelo lineal de la siguiente forma:

$$y(x; w_1, w_2) = w_1 x + w_2 \tag{2.7}$$

Donde:

- x es la variable independiente
- w_1, w_2 son los parámetros del modelo

Mientras el modelo lineal 2.7 tendera a estimar la Mediana de los datos, la regresión cuantil extiende este concepto a un percentil específico, siendo el percentil 50 el que da el valor de la mediana equivalente al que se obtiene usando la regresión lineal.

Para lograr esta generalización Takeuchi et al. [6] introdujo un término τ el cual con una función de perdida personalizada con el fin de que el modelo se acomode a la proporción τ de las muestras, sea por ejemplo.

- $\tau = 0{,}05$ significa que 5 % de los datos serán menores que lo estimado por el modelo.
- $\tau = 0{,}95$ significa que 95 % de los datos serán menores que lo estimado por el modelo.

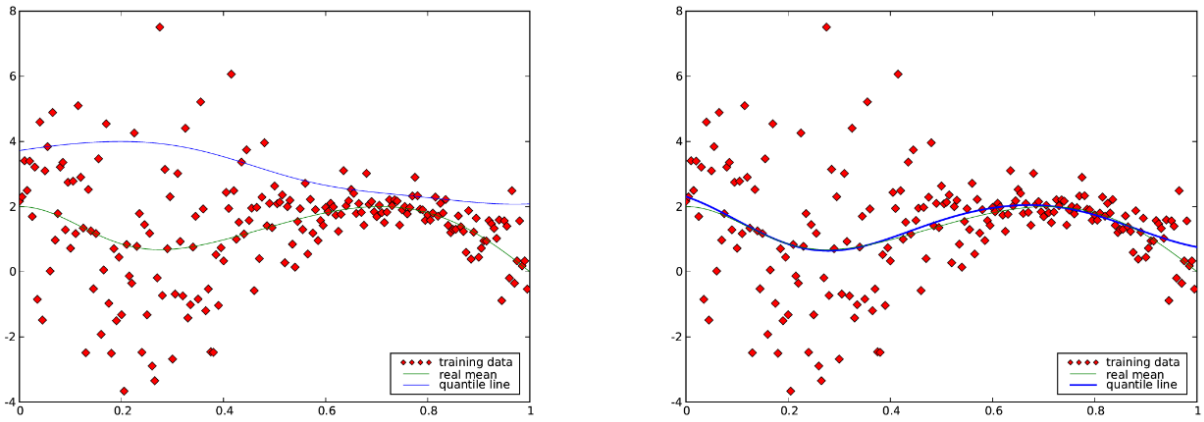

Figura 2.11: Ilustración de la regresión cuantil no paramétrica. A la izquierda, $\tau = 0,9$, a la derecha, $\tau = 0,5$. Nótese que la línea de regresión cuantil se aproxima mucho a la mediana de los datos (ya que ξ tiene una distribución normal (la mediana y la Media son idénticas))[6]

La función de perdida usada en la regresión cuantil no paramétrica se conoce como *pinball loss* y sigue la siguiente ecuación matemática:

$$L_\tau(y, \hat{y}) = \begin{cases} \tau(y - \hat{y}) & \text{si } y \geq \hat{y} \\ (1 - \tau)(\hat{y} - y) & \text{si } y < \hat{y} \end{cases} \tag{2.8}$$

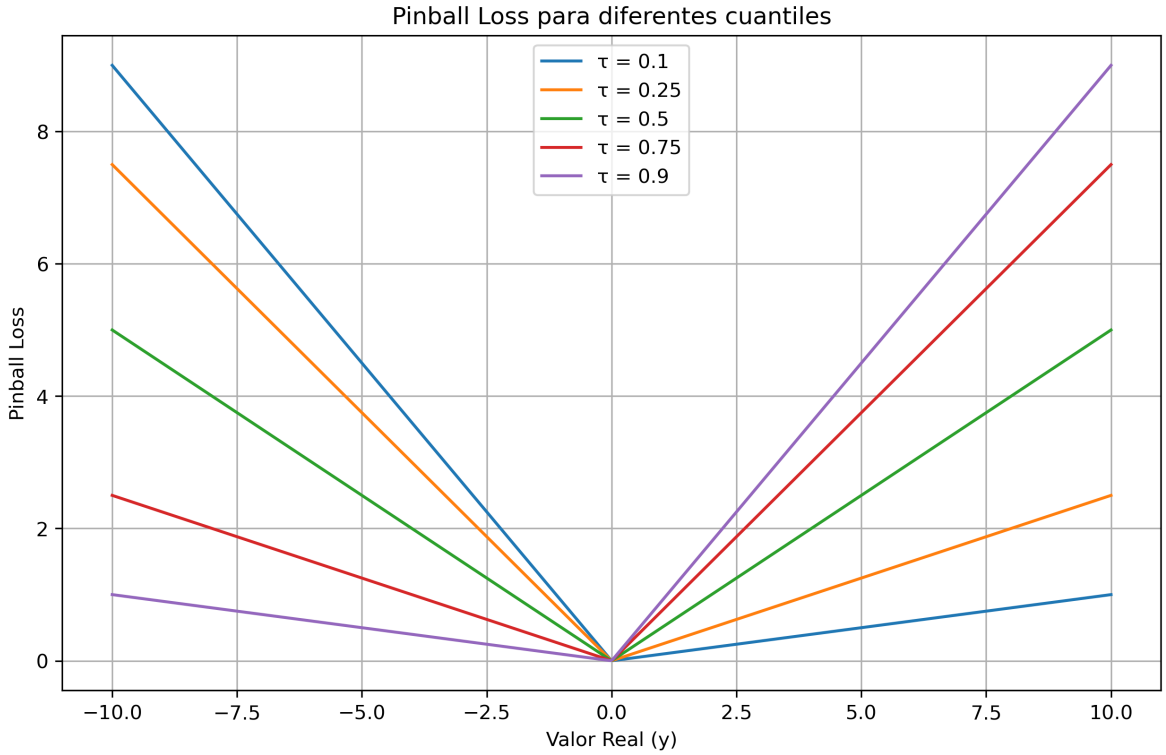

Figura 2.12: Función de perdida usada en la regresión cuantil no paramétrica.

Capítulo 3

Medición de Potencia Instantánea

A la hora de medir el consumo energético de un sistema embebido hay dos parámetros importantes que se deben conocer. El primer parámetro es el voltaje utilizado por el dispositivo y el segundo parámetro es la corriente consumida por el dispositivo. Posteriormente, esos dos parámetros se convierten a joules de acuerdo con las leyes de la física. Este proceso de conversión consta de dos pasos; el inicial consiste en obtener valores de potencia instantáneos a partir de valores de corriente y voltaje. Este proceso se realiza mediante la siguiente ecuación.

$$P(t) = V(t) \cdot I(t) \tag{3.1}$$

El segundo paso consiste en convertir la potencia en consumo energético.

$$E = \int_0^{T_0} P(t)dt \tag{3.2}$$

La interpretación física de E en la ecuación 3.2 es: «*E es el consumo de energía desde el instante de tiempo $t = 0$ hasta el instante de tiempo $t = T_0$*». Una diferencia importante entre $P(t)$ y E es que $P(t)$ tiene un valor único para un instante de tiempo dado t y E pertenece a un período de tiempo (desde $t = 0$ a $t = T_0$).

Cuando se utilizan mediciones del mundo real con un vatímetro, la integral en la ecuación 3.2 se convierte en una sumatoria, como se muestra en la siguiente ecuación.

$$E \approx \sum_{0}^{T_0} P(t_i)\Delta t \tag{3.3}$$

Dónde:

- Δt es el periodo de muestreo utilizado en el que se tomaron los datos.
- $P(t_i) = V(t_i) \cdot I(t_i)$ es la potencia instantánea en el instante t_i.

De acuerdo con la ecuación 3.3, medir el consumo energético requiere obtener muestras de voltaje y corriente consumidas por la Raspberry Pi 4b. El objetivo de este capítulo es medir la potencia instantánea consumida por la Raspberry Pi 4b, mediciones que serán utilizadas en capítulos posteriores para calcular su consumo energético. En el contexto de esta tesis, fue necesario realizar mediciones de la potencia instantánea de la Raspberry Pi 4b ejecutando el middleware IoTVar en diversas configuraciones. Para estas mediciones, se modificó el vatímetro desarrollado por Lambert et al. [7]. En su estudio, los autores diseñaron un vatímetro enfocado en medir la potencia instantánea de una Raspberry Pi 4b utilizando el sensor de corriente INA219A y un microcontrolador Teensy 4.0. Este diseño fue modificado para alcanzar frecuencias de muestreo más altas que las presentadas en dicho estudio. En esta tesis se utilizó un microcontrolador similar al Teensy 4.0, el RP2040, fabricado por la Raspberry Pi Foundation, integrado en la placa de desarrollo Raspberry Pi Pico, también desarrollada por la Raspberry Pi Foundation.

3.1. Cuello de Botella en el Diseño Original

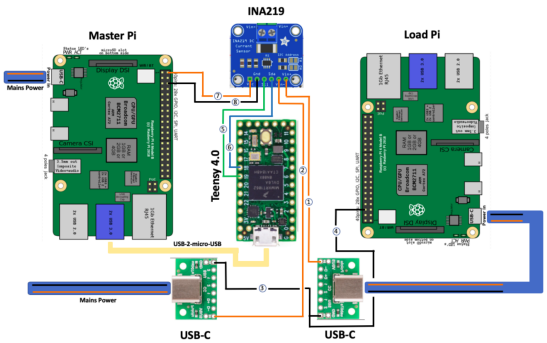

Figura 3.1: Posicionamiento y conexión de todos los cables jumper entre el Raspberry Pi 4 modelo B maestro, esclavo, amplificador de detección de corriente INA219A y el microcontrolador Teensy 4.0. [7]

En el diseño presentado por Lambert et al. [7] (figura 3.1) se usan dos Raspberry Pi 4b (uno actuando como principal y el otro como secundario). El principal se usa para recolectar los datos que vienen del microcontrolador Teensy 4.0 y el secundario funciona como carga, es el consumo energético del secundario el que se monitoreará.

Sin embargo, este diseño no toma en cuenta la frecuencia de muestreo mínima que se necesita para capturar la señal de potencia instantánea que consume la carga. Esta frecuencia de muestreo, para un sistema embebido, según Roeder et al. [56] depende de los siguientes factores:

1. Escalado dinámico voltaje-frecuencia (DVFS)
2. Número de instrucciones por ciclo de reloj
3. Instrucciones que se estén ejecutando

En el caso del Raspberry Pi 4b estos parámetros son controlados por un firmware interno desarrollado por la Raspberry Pi Foundation. En la literatura se reporta un caso muy cercano. Este es un estudio desarrollado por Roeder et al. [56], en dicho estudio los autores estudian el impacto de la frecuencia de muestreo usada en los posibles errores que puedan tener las mediciones de potencia. Sin embargo, este estudio usa el Odroid-XU4, un sistema embebido que ejecuta Linux embebido, para hacer sus mediciones. Los autores encontraron que para el Odroid-XU4 la frecuencia de muestreo mínima es de $500Hz$. Al haber utilizado otra placa distinta a la Raspberry pi 4b usada en esta tesis este resultado no se puede extrapolar a la Raspberry pi 4b. Por lo que para encontrar la frecuencia de muestreo mínima se replico el análisis utilizado por Roeder et al. [56]. Este análisis requiere de datos tomados a una frecuencia de muestreo alta por lo que se procedió a modificar el vatímetro desarrollado por Lambert et al. [7] con la finalidad de aumentar su frecuencia de muestreo.

El diseño presentado en la figura 3.1 usa el INA219A para medir voltaje y corriente, según la hoja de datos del INA219A, este sensor cuenta con un ADC programable que se usa tanto para medir voltaje como corriente según se ilustra en el diagrama mostrado la hoja de datos (figura 3.2).

Al configurar el INA219A para que haga lecturas tanto de voltaje como corriente este realizara la conversión de una muestra de voltaje (V_{IN-}) y seguidamente hará la conversión de una muestra

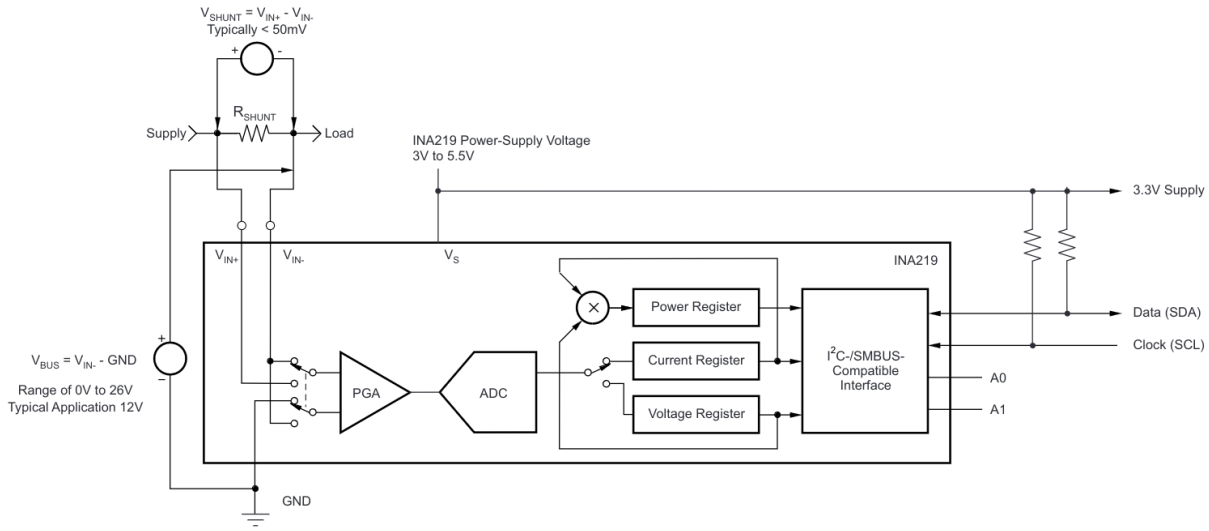

Figura 3.2: Diagrama del sensor INA219A, nótese que el ADC escribe sobre el registro de corriente o registro de voltaje [4]

de corriente ($V_{IN+} - V_{IN-}$). Esto causa que para una muestra de potencia (W), se tengan que hacer 2 conversiones, lo que duplica los tiempos de conversión que se muestran a continuación.

Tabla 3.1: Tiempos de conversión para distintas configuraciones del ADC del sensor INA219A [4]

Modo / Muestras	**Tiempo de Conversión**
9 bits / 1 muestra	$84\mu s$
10 bits / 1 muestra	$148\mu s$
11 bits / 1 muestra	$276\mu s$
12 bits / 1 muestra	$532\mu s$
12 bits / 2 muestras	$1{,}06ms$
12 bits / 4 muestras	$2{,}13ms$
12 bits / 8 muestras	$4{,}26ms$
12 bits / 16 muestras	$8{,}51ms$
12 bits / 32 muestras	$17{,}02ms$
12 bits / 64 muestras	$34{,}05ms$
12 bits / 128 muestras	$68{,}10ms$

Lambert et al. [7] utilizaron 12 bits a 2 muestras por conversión. Este tiempo de conversión da una frecuencia de muestreo de:

$$\begin{aligned} f &= \frac{1}{2 * 1{,}06 * 10^{-3}} \\ f &= 471{,}7Hz \end{aligned} \tag{3.4}$$

La frecuencia de muestreo base es de $471{,}7Hz$. Para aumentar la frecuencia de muestreo

sin perder resolución se decidió usar el INA219A solo para medir corriente y el ADC interno del RP2040 de 12 bits para medir el voltaje de bus (V_{IN-}). De esta forma se pudo duplicar la frecuencia de muestreo a la que el vatímetro puede medir voltaje y corriente.

3.2. Acondicionamiento del Voltaje de Bus

El INA219A tiene un amplificador de ganancia programable (PGA) lo cual posibilita la conexión de forma segura voltajes de hasta 26V. Por otro lado el ADC del RP2040 solo soporta voltajes de máximo $3{,}3V$, para leer el voltaje de bus (V_{IN-}) sera necesario acondicionar la señal.

La Raspberry Pi 4b requiere un voltaje de alimentación de al menos $4{,}84V$ y como máximo requiere de $5{,}17V$, sin embargo, se observaron eventos fortuitos que ocasionaban voltajes máximos de $5{,}5V$ y mínimos de $4{,}5V$, estos eventos son ajenos a la Raspberry Pi, pero en caso de ocurrir podrían dañar el ADC del microcontrolador. Para evitar daños en el ADC del microcontrolador se eligió un rango de trabajo de $0{,}8V \rightarrow 1{,}8V$. Para hacer este acondicionamiento de señal se siguió el procedimiento descrito por Carter and Mancini [5], con el objetivo de determinar el caso al que pertenece este acondicionamiento se solucionó el sistema de ecuaciones lineales tal y como lo describen los autores.

$$\begin{aligned} 0{,}8 &= m \times 4{,}84 + b \\ 1{,}8 &= m \times 5{,}17 + b \\ m &= 0{,}33, b = -4{,}576 \end{aligned} \tag{3.5}$$

Al calcularse en el procedimiento 3.5 que $m > 0$ y $b < 0$, el problema de acondicionamiento pertenece al caso 2 de acondicionamiento mencionado en el apartado *Medición de Voltaje* del marco teórico de la presente tesis. La figura 2.7 presenta el circuito utilizado. En la figura 2.7 se puede observar que el voltaje de entrada va directamente a la terminal no inversora del amplificador operacional, lo que hace que el diseño presentado por Carter and Mancini [5] requiera de una fuente de alimentación mayor a $5{,}17V$.

La Raspberry Pi Pico usada para este diseño cuenta con un regulador conmutado que opera a

una frecuencia de conmutación de 1MHz y convierte el voltaje de USB de 5V a 3,3V, conversión que se ilustra enseguida.

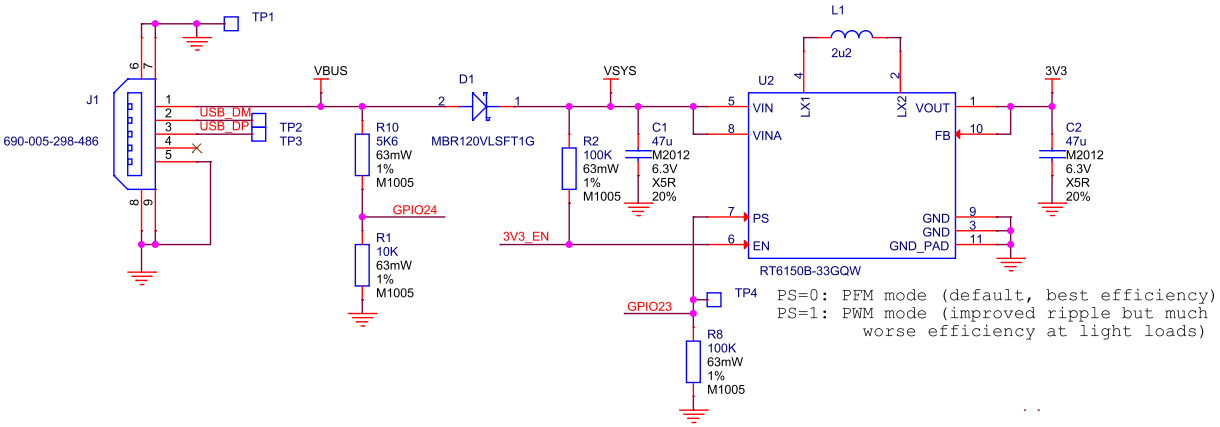

Figura 3.3: Etapa de potencia del Raspberry Pi pico Rev 3 [8]

Con fines de reproducibilidad y simplicidad, se optó por diseñar un circuito de acondicionamiento lo más sencillo posible. Esto llevó a adoptar como restricción principal el uso del mismo voltaje de alimentación de 5V que emplea la Raspberry Pi Pico, evitando la necesidad de fuentes adicionales. Durante la selección de componentes, se eligió el amplificador operacional LM324 debido a su alta disponibilidad en el mercado, bajo costo y adecuada tolerancia para aplicaciones de propósito general, lo que facilita la creación de prototipos con componentes fácilmente obtenibles.

El LM324 es un amplificador operacional de tipo single-supply, capaz de operar con alimentación de hasta 32V, pero que presenta una limitación importante: su rango de voltaje de entrada en modo común (V_{CM}) no alcanza el voltaje positivo de alimentación, sino que se restringe aproximadamente hasta $V_{CC} - 2\text{V}$ [57]. En este diseño, con $V_{CC} = 5\text{V}$, el voltaje máximo admisible en las entradas inversora y no inversora es de aproximadamente 3V. Esta limitación imposibilita conectar directamente señales cercanas a los 5V al amplificador.

Para resolver esta restricción, se diseñó un divisor de voltaje resistivo conectado a la entrada no inversora del LM324, tal como puede observarse en la figura 2.7. Su función es escalar la señal de entrada desde el rango original de 4,84V a 5,17V hacia un rango reducido de aproximadamente 1,51V a 1,61V, centrado dentro del margen seguro de operación del LM324. Este nuevo rango es suficientemente amplio para detectar con precisión las fluctuaciones de interés, y se mantiene dentro del rango operativo de 0V $\rightarrow$ 3,3V del ADC interno del RP2040, proporcionando tanto

seguridad eléctrica como fiabilidad en la lectura.

El cálculo del divisor de tensión siguió el siguiente procedimiento:

$$\begin{aligned} V_{out} &= 1{,}61V - 1{,}51V = 0{,}1V \\ V_{in} &= 5{,}17V - 4{,}84V = 0{,}33V \\ V_{out} &= \frac{R_2}{R_1 + R_2} \times V_{in} \\ \frac{0{,}1V}{0{,}33V} &= \frac{R_2}{R_1 + R_2} \\ 0{,}30 &= \frac{R_2}{R_2 + R_1} \\ 0{,}30 \times R_1 &= 0{,}7 \times R_2 \\ R_1 &= 2{,}3 \times R_2 \end{aligned} \tag{3.6}$$

Dada la última relación entre R_1 y R_2 se usaron valores comerciales de resistencias $R_1 = 2{,}2K\Omega$ y $R_2 = 1K\Omega$. Usando este valor la constante correspondiente al divisor de tensión tiene el siguiente valor:

$$c = \frac{R_2}{R_1 + R_2} = \frac{1K\Omega}{2{,}2K\Omega + 1K\Omega} = 0{,}3125 \tag{3.7}$$

Usando este divisor de tensión las ecuaciones para el acondicionamiento de tensión resueltas en el procedimiento 3.5 se tienen que volver a solucionar de la siguiente forma.

$$\begin{aligned} 0{,}8 &= m \times 1{,}51 + b \\ 1{,}8 &= m \times 1{,}61 + b \\ m &= 9{,}99, b = -14{,}29 \end{aligned} \tag{3.8}$$

Con los valores de m y b descritos en el procedimiento 3.8 se pueden usar las ecuaciones de diseño descritas por Carter and Mancini [5]. Para el caso 2 ($m > 0$ y $b < 0$) se tienen las siguientes ecuaciones de diseño.

$$
\begin{aligned}
m &= \frac{R1||R2 + RF + RG}{R1||R2 + RG} \\
|b| &= V_{REF} \cdot \frac{R2}{R2 + R1} \cdot \frac{RF}{R1||R2 + RG}
\end{aligned}
\tag{3.9}
$$

Donde V_{REF} es el voltaje de alimentación ($5V$ para este caso). Reemplazando m y b se tiene.

$$
\begin{aligned}
9{,}99 &= \frac{R1||R2 + RF + RG}{R1||R2 + RG} \\
14{,}29 &= V_{REF} \cdot \frac{R2}{R2 + R1} \cdot \frac{RF}{R1||R2 + RG}
\end{aligned}
\tag{3.10}
$$

El sistema no se puede solucionar analíticamente debido a que se tienen dos ecuaciones para cuatro variables. Ante este caso se decidió usar valores comerciales para $R1$ y $R2$, nótese que $R1, R2 \neq R_1, R_2$, los valores elegidos fueron $R1 = 3{,}3K\Omega$ y $R2 = 1{,}5K\Omega$, usando estos valores y la ecuación 3.10 se pueden calcular RF y RG de la siguiente forma.

En RG.

$$
\begin{aligned}
9{,}99 &= \frac{RF + RG + 1031{,}25}{RG + 1031{,}25} \\
9{,}99 \times (RG + 1031{,}25) &= RF + RG + 1031{,}25 \\
\frac{RF}{8{,}99} - RG &= 1031{,}25
\end{aligned}
\tag{3.11}
$$

En RF.

$$
\begin{aligned}
14{,}29 &= 5 \times \frac{1{,}5K}{4{,}8K} \times \frac{RF}{RG + 1031{,}25} \\
14{,}29 \times (RG + 1031{,}25) &= 1{,}5625 \times RF \\
RG + 1031{,}25 &= \frac{1{,}5625}{14{,}29} \times RF \\
\frac{1{,}5625}{14{,}29} \times RF - RG &= 1031{,}25
\end{aligned}
\tag{3.12}
$$

Sumando las ecuaciones 3.11 y 3.12 se tiene lo siguiente:

$$RF = 0 \tag{3.13}$$

De este resultado se puede interpretar que el sistema de ecuaciones es dependiente por lo cual tiene un número infinito de soluciones. Dado este escenario se eligió un valor comercial para $RF = 27K\Omega$. Reemplazando este valor en la ecuación 3.11 se tiene un valor para RG.

$$\begin{aligned} \frac{27K}{8{,}99} - RG &= 1031{,}25 \\ \frac{27K}{8{,}99} - 1031{,}25 &= RG \\ RG &= 1972\Omega \end{aligned} \tag{3.14}$$

Sea un valor comercial para $RG = 2K\Omega$.

Finalmente calculadas R_1, R_2, RF, RG, $R1$ y $R2$ se tiene listo el circuito de acondicionamiento que se expone en lo siguiente.

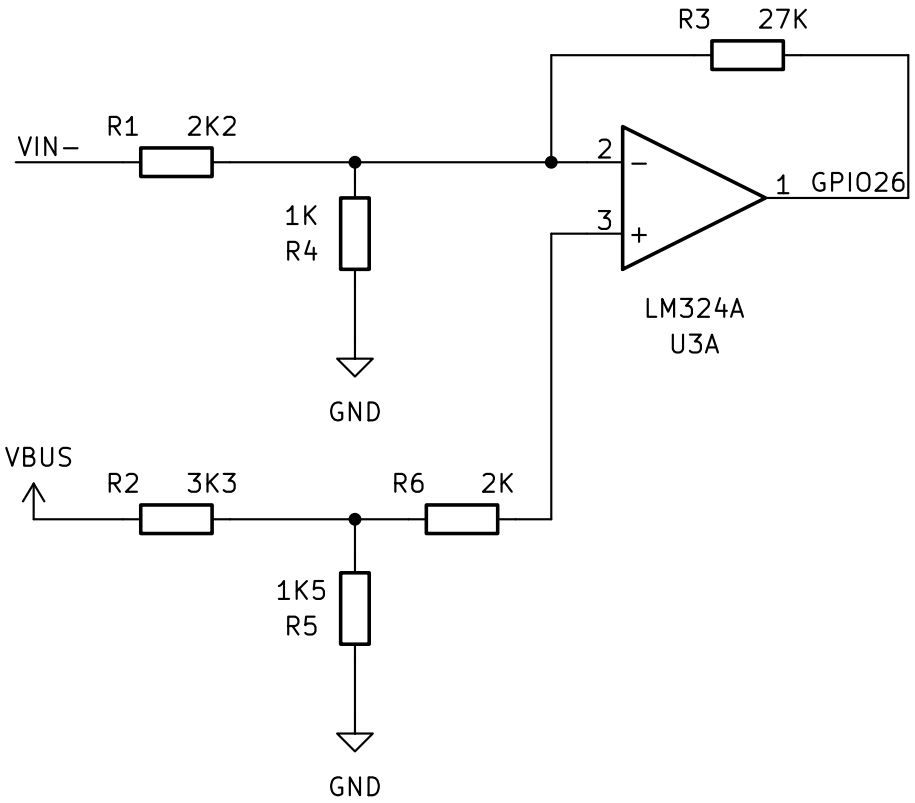

Figura 3.4: Circuito de acondicionamiento de señal para medir el voltaje de bus del Raspberry Pi 4b.

3.3. Captura de muestras del ADC del microcontrolador RP2040

Listo el circuito de acondicionamiento se puede usar el ADC del RP2040 para leer las lecturas de voltaje, el RP2040 tiene tres ADC SAR de los cuales uno está conectado a un sensor interno de temperatura y dos son programables, estos ADC cuentan con las siguientes características.

- Reloj independiente de $48MHz$
- La conversión de cada muestra toma 96 ciclos de reloj ($2\mu s$)
- Número efectivo de bits (ENOB): 8.7 bits, mide la resolución real del ADC considerando el ruido; aunque es de 12 bits, su precisión útil equivale a 8.7 bits.
- Generador de interrupciones e interfaz DMA

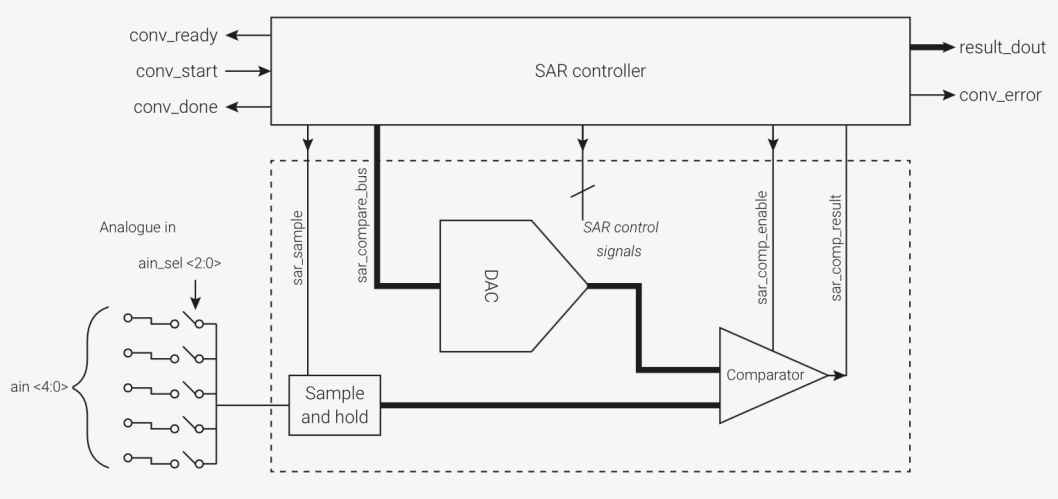

Figura 3.5: ADC del microcontrolador RP2040 [9].

Adicionalmente, el microcontrolador RP2040 cuenta con doce canales independientes de DMA (acceso directo a memoria). Cada uno de estos canales permite mover datos de un lugar en la memoria a otro, estos pueden ser:

1. Periférico a memoria
2. Memoria a periférico
3. Memoria a memoria

4. Periférico a periférico

Estas transferencias de datos pueden ser de hasta 32 bits cada ciclo de reloj, para cada transferencia se pueden configurar los siguientes aspectos.

- Tamaño de transferencia (8, 16 o 32 bits).
- Origen y destino
- Número de transferencias
- Evento de disparo de transferencia

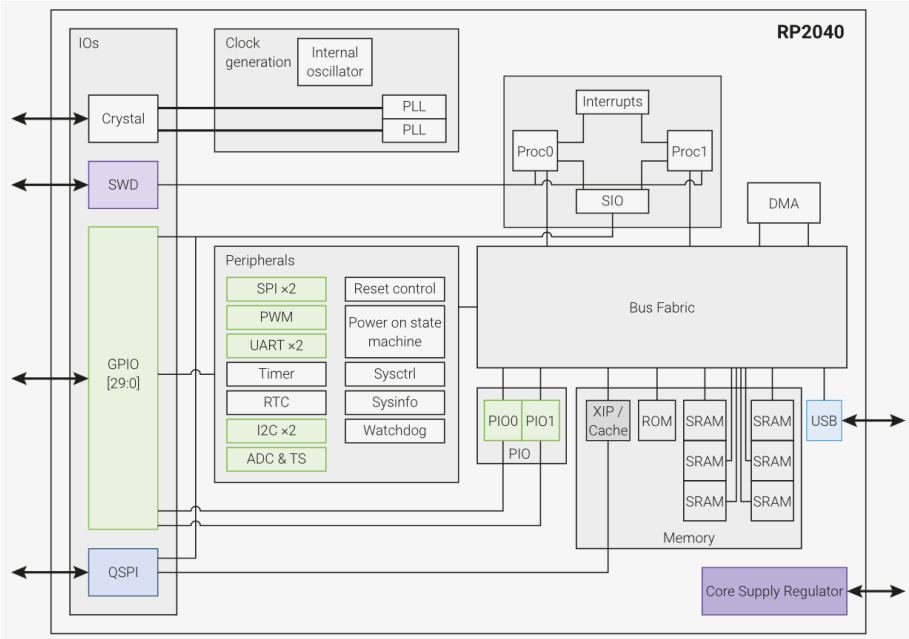

Figura 3.6: Diagrama de bloques del microcontrolador RP2040 [9].

Para incrementar la frecuencia de muestreo se usaron en conjunto el ADC y el DMA, permitiendo transferencias rápidas cada vez que una conversión del ADC se completaba. Dado que el número efectivo de bits del ADC es de 8.7 bits, el tamaño de transferencia se configuró a 8 bits. Además, se utilizó una rutina de interrupción para disparar las conversiones del ADC a una frecuencia de muestreo determinada. El proceso de conversión incluye un promediado de 8 muestras, lo que reduce el ruido presente en la señal y, en el dominio de la frecuencia, actúa como un filtro pasa bajos que atenúa las componentes de alta frecuencia, suavizando la señal resultante.

3.4. Medición de corriente consumida por la Raspberry Pi 4b

Una vez obtenidas las muestras de voltaje se procedió a adquirir las muestras de corriente correspondientes. Con este fin se usó el sensor de corriente INA219A utilizado en el diseño original por Lambert et al. [7]. Las conexiones se realizaron siguiendo la figura 3.1 al ser el INA219A un sensor de corriente de lado alto se conectó en serie con la carga entre la fuente de alimentación y la Raspberry Pi 4b.

Según la hoja de datos del INA219A [4], este sensor cuenta con un registro de calibración que se tiene que configurar para ajustar el rango de las mediciones de corriente. En el caso de la Raspberry Pi 4b se observaron picos de 1,3A, siguiendo el procedimiento descrito en la hoja de datos con $R_{SHUNT} = 0{,}1\Omega$ y para una corriente máxima se tiene el siguiente procedimiento.

Cálculo del rango posible de $Current_{LSB}$ (Min = 15 bits, Max = 12 bits)

$$\begin{aligned} Current_{LSB-min} &= \frac{\text{Corriente máxima esperada}}{32768} \\ Current_{LSB-min} &= \frac{1{,}3}{32768} \\ Current_{LSB-min} &= 39\mu A \\ Current_{LSB-max} &= \frac{\text{Corriente máxima esperada}}{4096} \\ Current_{LSB-max} &= \frac{1{,}3}{4096} \\ Current_{LSB-max} &= 317{,}4\mu A \end{aligned} \tag{3.15}$$

Del procedimiento 3.15 se sabe que $Current_{LSB}$ debe ser cercano a $39\mu A$ y menor a $317{,}4\mu A$. Dada esta condición de diseño se puede elegir cualquier valor que cumpla dichas condiciones, para el caso del desarrollo de esta tesis se eligió que este valor sea de $Current_{LSB} = 50\mu A$ por bit. Usando este último valor se puede calcular el valor de calibración del INA219A.

$$\begin{aligned} Cal &= trunc(\frac{0{,}04096}{Current_{LSB} \times R_{SHUNT}}) \\ Cal &= trunc(\frac{0{,}04096}{50\mu \times 0{,}1}) \\ Cal &= 8192 \end{aligned} \tag{3.16}$$

Este valor se configura en el INA219A de la siguiente manera.

Listing 1 Escritura en el registro de calibración del INA219A.

```
#define INA219_REG_CALIBRATION 0x05

uint16_t ina219_calValue = 8192;
uint8_t data[] = {INA219_REG_CALIBRATION, (ina219_calvalue >> 8) & 0xFF, ina219_calvalue & 0xFF};

i2c_write_blocking(ina219_i2cChan, ina219_i2caddr, data, sizeof(data), false);
```

Por otra parte, también es necesario configurar el amplificador de ganancia programable (PGA) y el tipo de conversión que se desea. La tabla 3.1 describe las distintas opciones de conversión, para hacer las primeras pruebas se usó la configuración más rápida de promediado de dos muestras de doce bits. Según la hoja de datos del INA219A tanto el PGA como el ADC se configuran escribiendo en el registro de configuración del INA219A.

Listing 2 Configuración INA219A PGA y ADC

```
#define INA219_REG_CONFIG 0x00

uint16_t config = 0x0000 | 0x1000 | 0x0480 | 0x0050 | 0x0001;
uint8_t data[] = {INA219_REG_CONFIG, (config >> 8) & 0xFF, config & 0xFF};

i2c_write_blocking(ina219_i2cChan, ina219_i2caddr, data, sizeof(data), false);
```

Tabla 3.2: Configuración inicial del PGA y ADC del INA219A [4]

Valor (hexadecimal)	**Configuración**
0x0000	Rango del voltaje de bus en $0 - 16V$
0x1000	Ganancia 4 para el voltaje en la resistencia de shunt (max $1{,}6A$)
0x0780	Voltaje de bus ADC 12 bits, 128 muestras (no utilizado)
0x0050	Voltaje de shunt ADC 12 bits, 2 muestras
0x0001	Solo conversión del voltaje de shunt

3.5. Implementación del prototipo

Para tomar datos cómodamente se diseñó una PCB que incluye todas las conexiones del INA219A y el circuito de acondicionamiento para la señal de bus del Raspberry Pi 4b. Este prototipo se implementó para la placa de desarrollo Raspberry Pi pico e incluye el amplificador operacional LM324, resistencias del circuito de acondicionamiento y el sensor INA219A. Todos los componentes mencionados se utilizaron en encapsulados through hole.

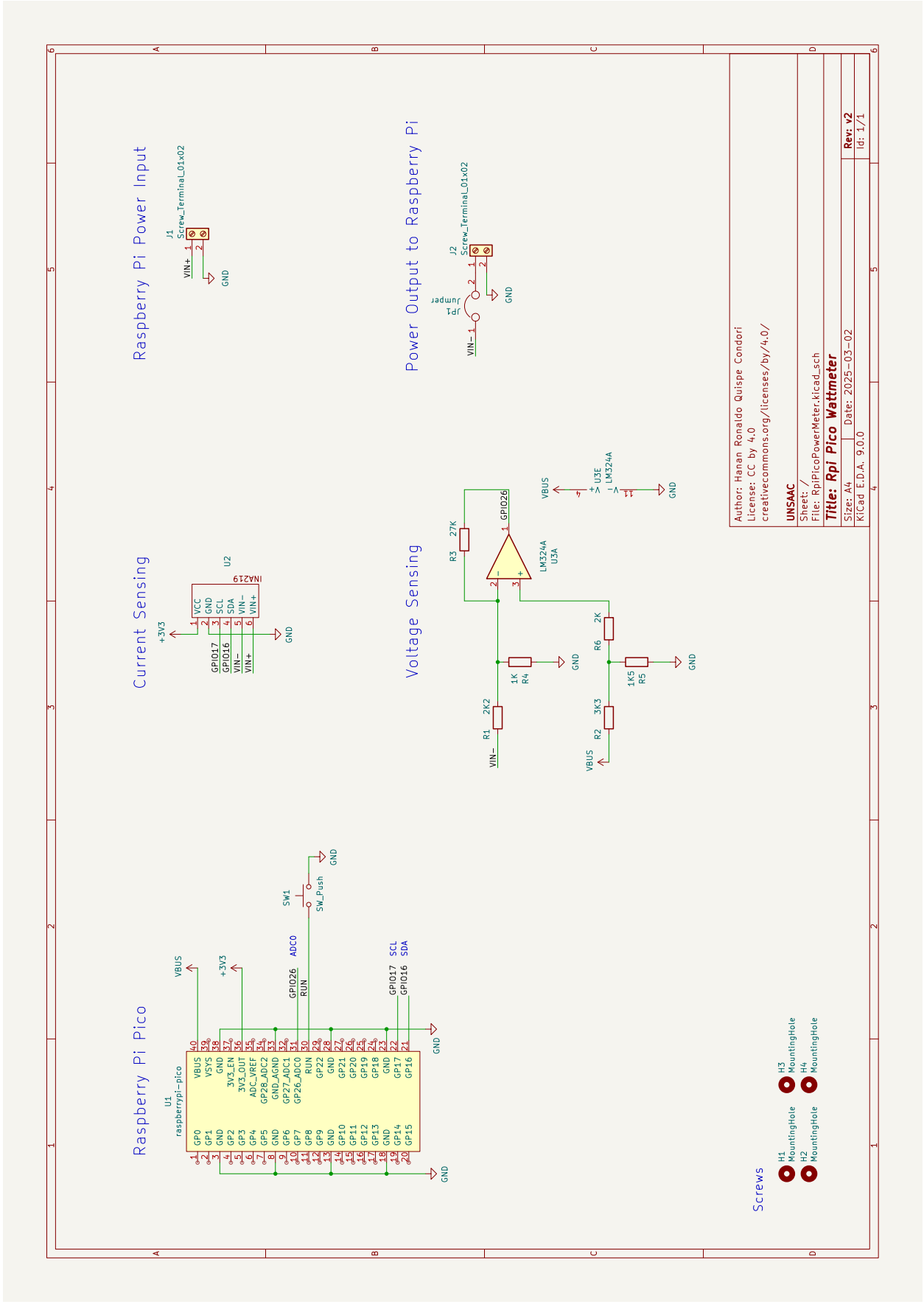

Figura 3.7: Representación esquemática del prototipo

Figura 3.8: Diseño PCB correspondiente al prototipo desarrollado

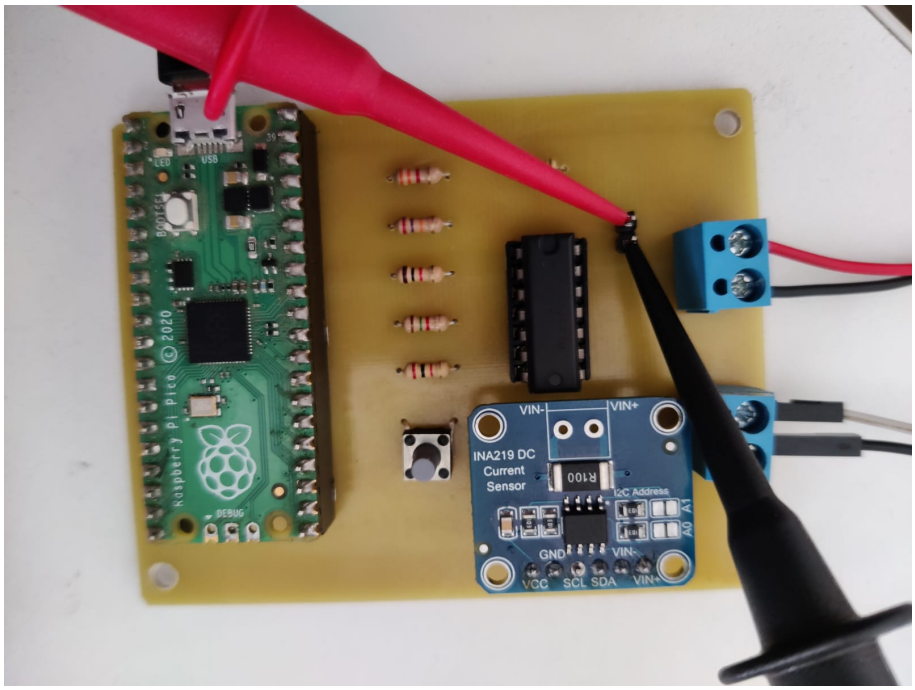

Figura 3.9: Implementación del prototipo desarrollado

Adicionalmente, a los componentes de medición de corriente y voltaje en la figura 3.9 se pueden observar dos borneras. En la figura 3.9 la bornera que se encuentra en la zona superior es la salida de $5V$ que se conecta al Raspberry Pi 4b. Por otro lado, la bornera que se encuentra en la zona inferior, es la entrada de $5V$ que se conecta a la fuente de $5V$ que alimenta al sistema. Adicionalmente, la placa de desarrollo Raspberry Pi pico cuenta con una entrada micro-USB que se usa enviar los datos recolectados a una PC externa, en el caso del estudio de Lambert et al. [7], otra Raspberry Pi 4b recolectaba los datos, en el arreglo desarrollado para esta tesis se utilizó una laptop, esta elección no afecta al proceso de recolección de datos de potencia.

3.5.1. Calibración del prototipo

Todos los cálculos de los componentes que se realizaron en el apartado anterior son válidos para valores de los componentes ideales, en la implementación física se observó que varias de las resistencias tenían valores dentro del porcentaje de error indicado en los encapsulados. Este hecho hace que las constantes y ganancias sean diferentes a comparación de los calculados usando valores teóricos.

3.5.1.1. Calibración del ADC del microcontrolador RP2040

El primer factor a calibrar es el factor de conversión del ADC, este procedimiento se llevó a cabo haciendo dos mediciones de voltaje con equipo de laboratorio y calculando la lectura del ADC. El factor de conversión se calcula usando la siguiente relación.

$$V = adc * conversion \tag{3.17}$$

Primera medición 1.271V medido por voltímetro Keysight 34465A corresponde a 101 en el ADC.

La ecuación para extraer un valor de voltaje del valor de adc es $V = adc * convertion$, entonces $1{,}271 = 101 * convertion$.

```
conv1 = 1.271/101
conv1
```

```
0.012584158415841583
```

Segunda medición 1.439V medido por voltímetro de laboratorio corresponde a 114 en el ADC.

La ecuación para extraer un valor de voltaje del valor de adc es $V = adc * convertion$, entonces $1{,}439 = 114 * convertion$.

```
conv2 = 1.439/114
conv2
```

```
0.01262280701754386
```

Tomando el promedio entre ambos factores de conversión encontrados.

```
(conv1+conv2)/2
```

```
0.012603482716692722
```

Finalmente, se tiene que el factor de conversión del ADC del microcontrolador RP2040 es de 0,012603482716692722.

3.5.1.2. Calibración del divisor de voltaje

La ecuación del divisor de voltaje es una constante c que multiplica la señal de entrada $Vout = c * Vin$. Calculemos la constante usando mediciones reales. Para un voltaje de entrada de $4{,}9879V$ se encontró que el voltaje de salida fue de $1{,}5722V$, entonces el valor de la constante del divisor de voltaje se puede calcular de la siguiente manera.

```
Vin = 4.9789
Vout = 1.5722
c = Vout/Vin
c
```

```
0.31577256020406114
```

3.5.1.3. Calibración de la pendiente m y constante b del circuito de acondicionamiento

El circuito de acondicionamiento amplifica el voltaje entregado por el divisor de tensión mediante una función lineal $m \cdot V + b$. Calculemos los valores de m y b. Como el circuito se representa mediante una ecuación lineal, dos puntos son suficientes para encontrar m y b.

Se realizaron dos mediciones, la primera medición tubo los siguientes voltajes en la entrada y salida. $V_{in1} = 1{,}57V$, $V_{out1} = 1{,}405V$. La segunda medición dio los siguientes voltajes $V_{in2} = 1{,}689V$, $V_{out2} = 2{,}516V$. Dados dos pares ordenados se puede calcular la ecuación de la recta que pasa por ambos puntos.

```
import numpy as np
# first point
Vin1 = 1.57
Vout1 = 1.405
# second point
Vin2 = 1.689
Vout2 = 2.516
Vin = np.array([[Vin1,-1],[Vin2,-1]])
Vout = np.array([[Vout1],[Vout2]])
[m,b] = np.matmul(np.linalg.inv(Vin),Vout)
print(m,b)
```

```
[9.33613445] [13.25273109]
```

De este último procedimiento se encontró que $m = 9{,}33613445$ y $b = 13{,}25273109$, estos valores son cercanos a los calculados teóricamente, por lo que se puede concluir que tanto el diseño como la implementación cumplen con los requerimientos planteados.

3.5.1.4. Integración de los factores de conversión en el firmware del prototipo

Calculados experimentalmente los factores de conversión se puede hacer las conversiones de los valores enteros que da el ADC a valores de voltaje. Esta conversión se realiza de la siguiente manera.

$$\begin{aligned} V_{ADC} &= adc \times convertion \\ V_{ADC} &= m \times V_{BUS} \times c - b \\ V_{BUS} &= \frac{V_{ADC} + b}{m \times c} \\ V_{BUS} &= \frac{adc \times convertion + b}{m \times c} \end{aligned} \tag{3.18}$$

En el firmware del prototipo este procedimiento se integró de la siguiente manera.

Listing 3 Conversión de la lectura del ADC del microcontrolador RP2040 en voltaje de bus

```
#define m 9.33613445
#define b 13.25273109
const float adc_conversion_factor = 0.012603482716692722;
#define GAIN_VOLT_DIVIDER 0.3157725602040611 4

 float adc2busVoltageOpamp(double adc_reading) {
  float adcVoltage = adc_reading * adc_conversion_factor;

  float busVoltage = (adcVoltage + b) / (m * GAIN_VOLT_DIVIDER);

  return busVoltage;
}
```

3.6. Calculo de Potencia consumida por la Raspberry Pi 4b

El cálculo de potencia sigue la ecuación 3.1. En los dos apartados anteriores se obtuvieron los valores de voltaje y corriente por lo que solo queda multiplicarlos para obtener la potencia

instantánea en un instante t. Este procedimiento se realizó en el firmware del prototipo, se muestra a continuación la sección que realiza dicha operación.

Listing 4 Calculo de potencia instantánea.

```
1 float current = INA219.getCurrent_mA();
2 float voltage = adc2busVoltageOpamp(adcData);
3 float power = current * voltage;
```

Los scripts mostrados en este capitulo fueron extraídos del firmware del prototipo. Este firmware completo así como archivos de configuración se pueden encontrar en los apéndices de la presente tesis.

3.7. Calculo de frecuencia de muestreo mínima

El calculo de la frecuencia de muestreo mínima utilizado por Roeder et al. [56] consiste en realizar varias mediciones de potencia instantánea consumida por la Raspberry pi 4b. Seguidamente estas mediciones se someten a un proceso de submuestreo a frecuencias menores que la mitad de la frecuencia a la que se tomaron los datos ($1kHz$), en este trabajo se emplearon frecuencias discretas comprendidas entre $1Hz$ y $500Hz$, específicamente: 1, 2, 3, 4, 5, 10, 20, 30, 40, 50, 80, 100, 150, 200, 250, 300 y $500Hz$. Posterior al proceso de submuestreo con cada señal resultante se toma la integral para calcular el consumo energético. Seguidamente, estos valores de consumo energético se comparan con el consumo energético de la señal original. Resultado de este procedimiento se tiene la siguiente gráfica.

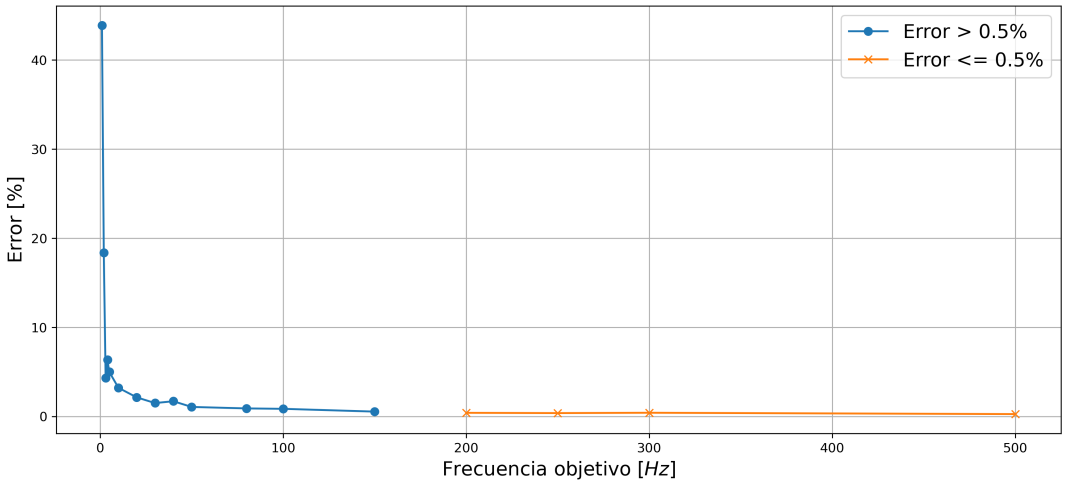

Figura 3.10: Errores de submuestreo.

Del análisis realizado se concluye que la frecuencia de muestreo mínima a la que se deben tomar los datos de potencia obteniendo un error menor al 0,5 % es de $200Hz$. Tomando en cuenta este hecho todas las mediciones se realizaron a una frecuencia de muestreo de $333Hz$ que al ser superior a $200Hz$ asegura un nivel de error aceptable (menor al 0,5 %).

Finalmente, la conversión de potencia instantánea (W) a consumo energético (J) se realiza como parte de la recolección del dataset usado para el desarrollo del modelo energético objetivo de esta tesis. Dicho proceso de conversión se realizará en el siguiente capitulo de esta tesis.

Capítulo 4

Modelo Energético

El desarrollo de este modelo tiene por objetivo la mejora del middleware IoTVar. En la figura 2.3 se pueden observar las diferentes capas de IoTVar, dentro de estas capas, la capa de energía implementa estrategias de eficiencia energética las cuales se busca que usen estimaciones del consumo energético del sistema embebido donde se ejecute IoTVar. La capa de energía tiene los siguientes componentes.

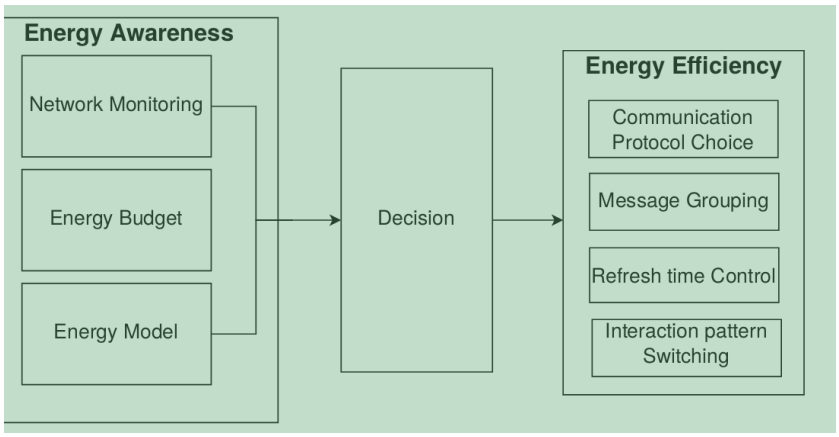

Figura 4.1: Componentes de la capa de energía de IoTVar [10]

Una de las estrategias de eficiencia energética consiste en el agrupamiento de solicitudes HTTP de varios sensores que realiza el middleware al broker IoT. Usando el agrupamiento de solicitudes Borges et al. [10] encontró que se puede obtener lecturas de hasta 200 sensores usando un máximo de $800J$, haciendo cada solicitud a un periodo de un segundo. Este capítulo se centrará en encontrar la relación entre el número de sensores y el periodo de actualización de las lecturas de los sensores con la evolución del consumo energético en el tiempo.

4.1. Toma de datos y diseño de los experimentos

El proceso de recolección de datos implicó la configuración de la Raspberry Pi 4B como un cliente que ejecuta el middleware IoTVar con la estrategia de agrupación de mensajes (HTTP). Para mantener la comparabilidad con un estudio previo, cada experimento comenzó con un período de calentamiento de 60 segundos, seguido de 300 segundos de ejecución. El número de sensores se incrementó de manera incremental en 25 unidades, comenzando desde 25 hasta alcanzar 200, mientras que el período de actualización se ajustó en siete intervalos calibrados: 1, 3, 5, 10, 15, 20 y 60 segundos. Estos intervalos fueron seleccionados en función de la variabilidad observada en el consumo de energía, donde los períodos entre 1 y 10 segundos mostraron mayores fluctuaciones, mientras que los períodos superiores a 20 segundos demostraron una mayor estabilidad. Para mejorar la validez estadística y reducir el ruido en las mediciones, cada combinación de cantidad de sensores y período de actualización se repitió 25 veces. Además, la PC utilizada para la recolección de datos seriales del vatímetro también ejecutó un simulador de sensores y un corredor local de FIWARE, permitiendo la transmisión en tiempo real a través de WiFi. Para automatizar los experimentos, se desarrolló un script que, mediante SSH, inicia la ejecución en la Raspberry Pi 4B, recupera la salida y sincroniza las mediciones de energía con la ejecución del experimento. Al finalizar, los metadatos y trazas de consumo de energía se almacenan correctamente para su posterior análisis. A continuación, se presenta una descripción general de los principales componentes y funcionalidades del script:

1. **Configuración e Inicialización**:
    - El script importa módulos personalizados para interactuar con el microcontrolador RP2040 (para mediciones de energía) y establecer conexiones SSH.
    - Define varias funciones auxiliares para tareas como tomar mediciones, construir comandos SSH y analizar marcas de tiempo.
2. **Configuración del Experimento**:
    - Establece parámetros para el experimento, incluyendo la duración, el período de

actualización de datos y las convenciones de nomenclatura de archivos.

- Inicializa conexiones con el RP2040 para mediciones de energía y con una Raspberry Pi 4B (vía SSH) para ejecutar los experimentos principales.

3. **Preparación para la Recolección de Datos**:

   Se prepara para recolectar tres tipos principales de datos:

   a) Consumo de energía (usando el microcontrolador RP2040)

   b) Uso de CPU (en la Raspberry Pi que ejecuta los experimentos)

   c) Metadatos de ejecución (marcas de tiempo, conteo de sensores, etc.)

4. **Bucle Principal del Experimento**:

   - Ejecuta una estructura de bucles anidados para probar diferentes configuraciones:
   - Bucle externo: Varía el número de sensores simulados (de 25 a 200, en pasos de 25)
   - Bucle interno: Ejecuta 25 iteraciones para cada cantidad de sensores

   Para cada iteración:

   - Ejecuta el middleware IoTvar en la Raspberry Pi 4B remota
   - Registra la marca de tiempo de inicio y otros metadatos

5. **Gestión de Datos**:

   - Registra continuamente los metadatos del experimento en un archivo CSV
   - Recopila datos de consumo de energía durante toda la duración del experimento

6. **Limpieza y Recuperación de Datos**:

   - Una vez completados todos los experimentos, detiene todas las mediciones
   - Transfiere el archivo de registro de uso de CPU desde la Raspberry Pi 4B a la máquina local
   - Cierra todas las conexiones y archivos

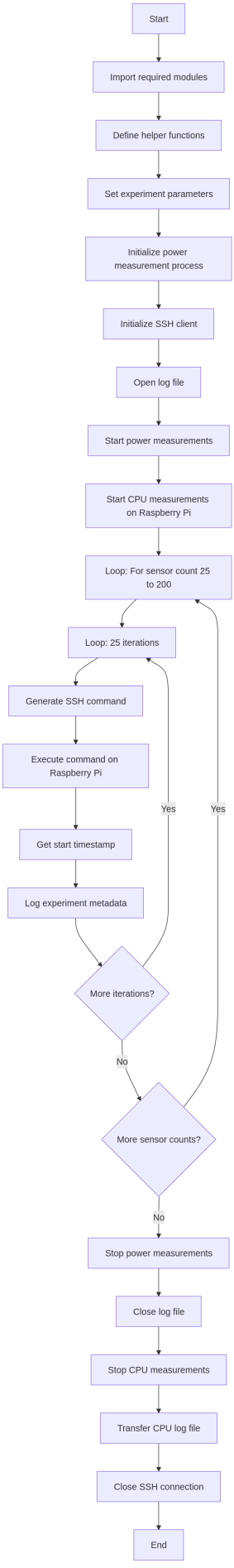

Figura 4.2: Diagrama de flujo del script de toma de datos

Este script se ejecutó en una laptop Dell XPS 13 9360 la cual cuenta con un procesador Intel Core i5-7200U CPU @ 2.50GHz y 8.0 GiB de memoria RAM. Así mismo, en esta laptop también se ejecutó el broker IoT FIWARE y un simulador de valores de sensores. Este simulador es configurable y publica lecturas de temperatura simuladas. Tanto el número de sensores que publican datos y la frecuencia a la que los datos son publicados es configurable en este simulador.

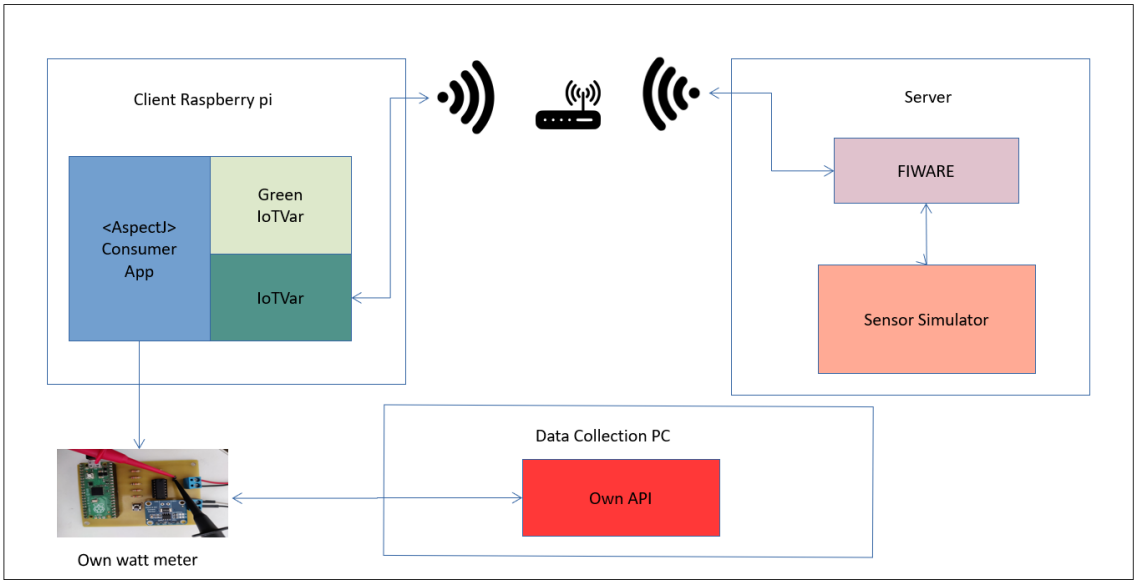

Figura 4.3: Esquema general de los experimentos

Adicionalmente, la figura 4.3 muestra también la conexión wifi que la Raspberry Pi 4b uso para conectarse al broker FIWARE.

Los experimentos se ejecutaron modificando diversos parámetros del middleware IoTVar. Estos parámetros incluyeron el número de sensores, el tiempo de ejecución, la URL de la plataforma, la estrategia empleada, el periodo de actualización y la URL de notificación.

Dentro del middleware IoTVar las variables IoT se declararon de la siguiente forma para todos los experimentos.

Listing 5 Declaración de variables IoT en el middleware IoTVar.

```
1 IoTVariableFiware<String> var = new IoTVariableFiware<String>("sensor" + (i + 1), "Sensor",
  ↪ "temperature", null, newFreshness(freshnessFrequency, TimeUnit.SECONDS), 10, (String) null, orion,
  ↪ handlerStrategy);
```

Como se puede apreciar en el script 5. Las variables IoT se crean con los siguientes parámetros:

- *@param* id

- *@param* type
- *@param* attribute
- *@param* location
- *@param* freshness
- *@param* historySize
- *@param* filters
- *@param* orion
- *@param* handlerStrategy

Para el caso de los experimentos todas las variables IoT pertenecen a sensores de temperatura, estos valores de temperatura se encuentran en el broker FIWARE.

4.2. Recopilación del dataset de consumo energético

El prototipo implementado en el capítulo anterior envía lecturas de voltaje (voltios), corriente (miliamperios) y potencia instantánea (mili vatios) en el siguiente formato.

Listing 6 Datos de salida del prototipo

```
1
2  uint64_t current_time_us_after_ina219 = time_us_64();
3
4  adc_dma_sample(cap_buf);
5  double adcData = 0.0;
6  for (int i = 0; i < NSAMP; i++) {
7    adcData += cap_buf[i];
8  }
9
10 adcData = adcData / NSAMP;
11 adcData = round(adcData);
12
13 float current = INA219.getCurrent_mA();
14
15 float voltage = adc2busVoltageOpamp(adcData);
16
17 float power = current * voltage;
18
19 printf("%lld,%.1f,%.4f,%.4f\n", current_time_us_after_ina219, current, voltage, power);
```

A continuación se muestran gráficas de las curvas de voltaje, corriente y potencia instantánea de uno de los experimentos.

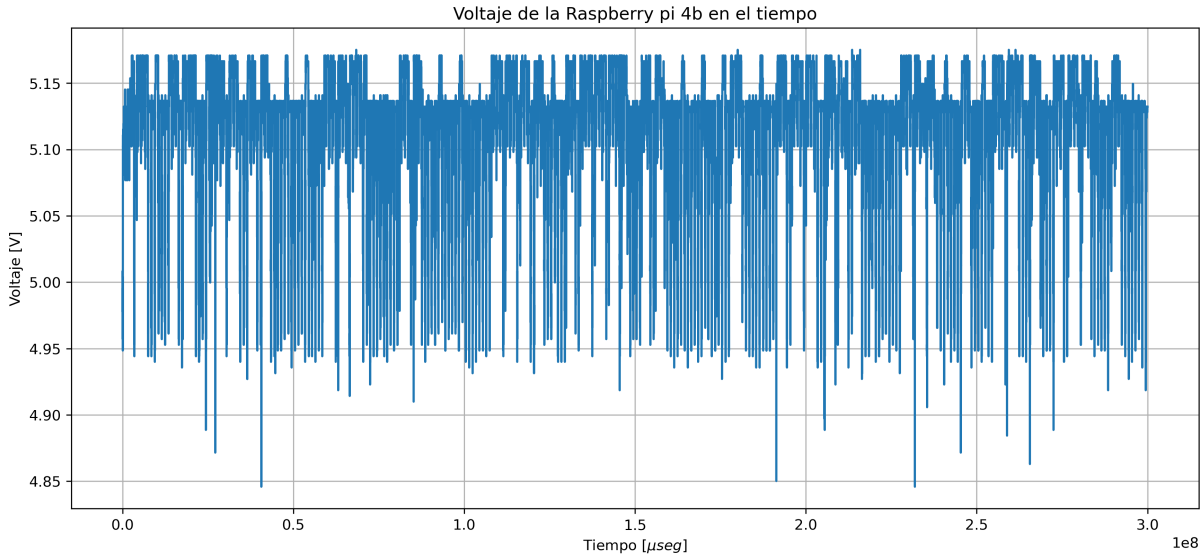

Figura 4.4: Evolución del voltaje de la Raspberry Pi 4b durante uno de los experimentos

La figura 4.4 muestra el voltaje con respecto al tiempo. El rango de voltajes observado es el que se mencionó al diseñar el circuito de acondicionamiento de señal, no se observan los eventos anómalos por lo que se puede concluir que la Raspberry Pi 4b estuvo funcionando con normalidad durante todo el experimento.

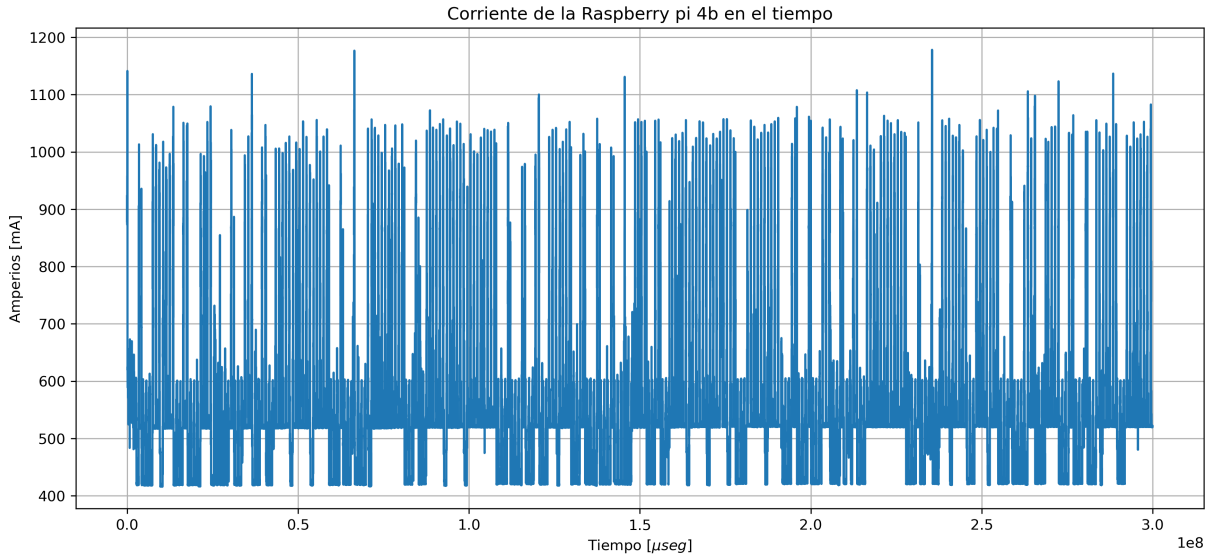

Figura 4.5: Evolución de la corriente consumida por la Raspberry Pi 4b durante uno de los experimentos

La figura 4.5 muestra la corriente consumida por la Raspberry Pi 4b durante uno de los experimentos, al igual que en la gráfica del voltaje los valores de corriente se encuentran en su rango de operación normal el cual es de máximo $3A$ [58].

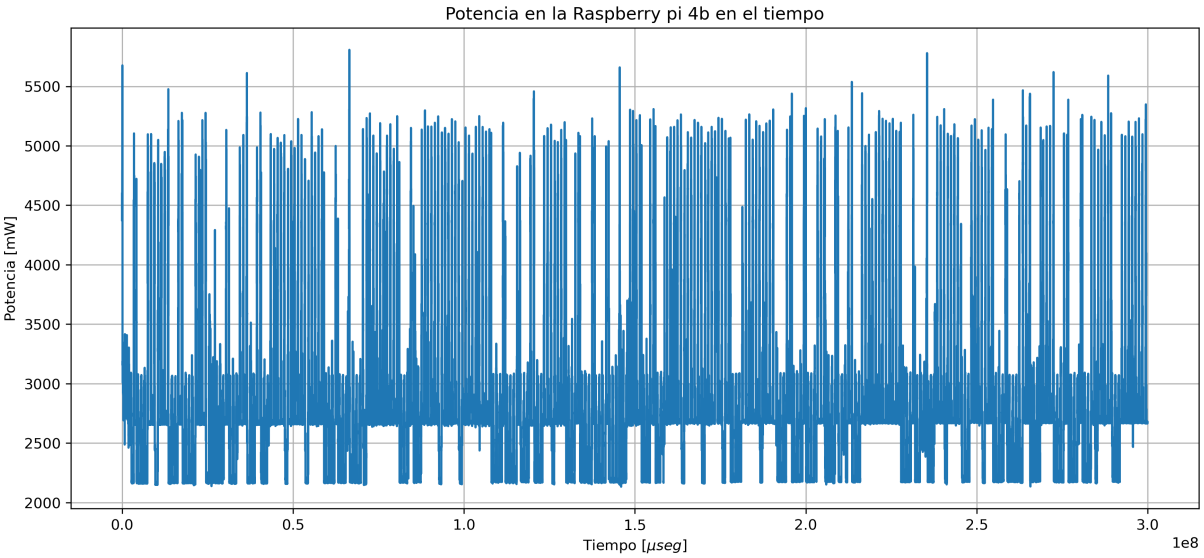

Figura 4.6: Evolución de la potencia consumida por la Raspberry Pi 4b durante uno de los experimentos

La figura 4.6 de igual forma que las curvas de voltaje y corriente también exhibe un rango esperado por debajo de los $15W$ [58].

Usando las mediciones de voltaje y corriente se puede observar también como varia la impedancia de la Raspberry pi 4b.

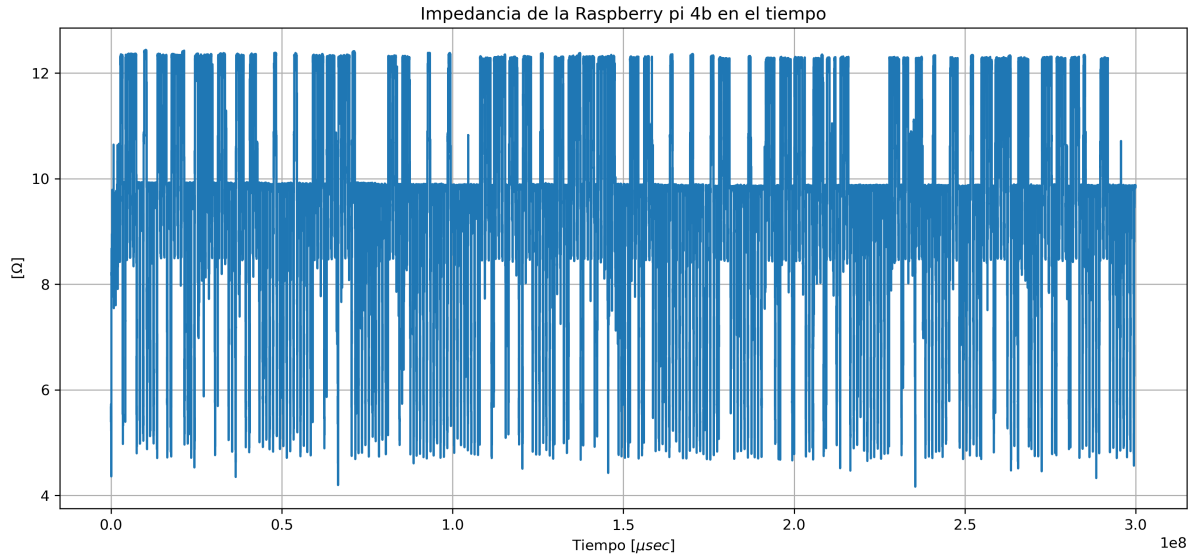

Figura 4.7: Evolución de la impedancia de la Raspberry pi 4b durante uno de los experimentos

La figura 4.7 muestra como varía la impedancia a través del tiempo. En esta curva se puede observar el comportamiento no lineal del dispositivo desde un punto de vista eléctrico. Esta no linealidad motiva al uso de métodos basados en datos para su modelado.

Dentro de las curvas de voltaje, corriente y potencia, la curva de potencia contiene la información que es de interés para el desarrollo del modelo energético, para convertir medidas de potencia (W) a watt-hora (Wh) o joule (J) se tiene que integrar las medidas en un periodo de tiempo determinado siguiendo la siguiente fórmula matemática.

$$\begin{aligned} E[J] &= T \times \sum_{i=0}^{i=t} W_i \\ E[Wh] &= \frac{T}{3600} \times \sum_{i=0}^{i=t} W_i \end{aligned} \tag{4.1}$$

Donde:

- $E[J]$ es la energía expresada en joule.
- $E[Wh]$ es la energía expresada en watt-hora.
- T es el periodo de muestreo al que se hicieron las muestras.
- W_i es la potencia instantánea en el instante i.
- t es el último instante el cual marca el final de la integración.

De las ecuaciones 4.1 se puede derivar la siguiente expresión.

$$E[Wh] = \frac{1}{3600} \times E[J] \tag{4.2}$$

La conversión mostrada en 4.2 no es más que una conversión de unidades, el uso de J o Wh es equivalente, la única diferencia es que J es una unidad estándar del sistema internacional de unidades y Wh no lo es.

El propósito del modelo energético es el de estimar el consumo energético de la Raspberry Pi 4b en un instante t en el futuro. Al ser este momento t una variable con el fin de tener la mayor cantidad de datos con el que entrenar al modelo se calculó el consumo energético para todos los t que era físicamente posible medir. Esto se realizó mediante el siguiente procedimiento.

$$
\begin{aligned}
E_1[J] &= T \times (W_0 + W_1) \\
E_2[J] &= T \times (W_0 + W_1 + W_2) \\
E_3[J] &= T \times (W_0 + W_1 + W_2 + W_3) \\
E_4[J] &= T \times (W_0 + W_1 + W_2 + W_3 + W_4) \\
E_5[J] &= T \times (W_0 + W_1 + W_2 + W_3 + W_4 + W_5) \\
&\vdots
\end{aligned}
\tag{4.3}
$$

Siguiendo el procedimiento 4.3 se obtiene un vector de muestras correspondiente a los instantes de tiempo $T, 2T, 3T, \ldots$. Nótese que mientras más pequeño sea el periodo de muestreo al que se capturan las muestras mejor se podrá capturar la dinámica interna del consumo energético de la Raspberry pi 4b.

Graficando los datos de consumo energético que se quieren estimar se tiene lo siguiente.

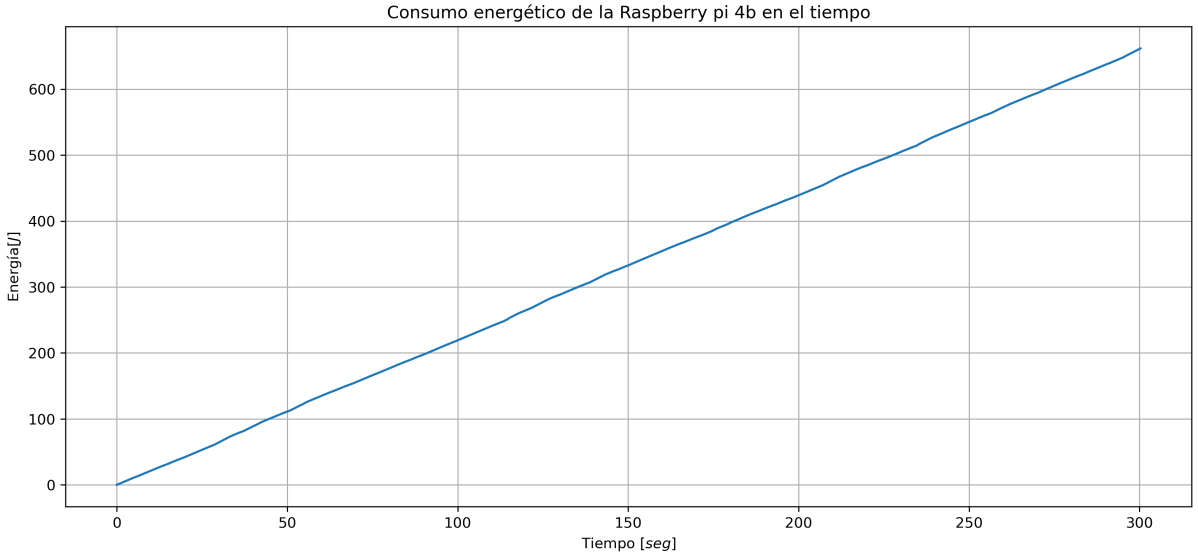

Figura 4.8: Evolución del consumo energético de la Raspberry Pi 4b en uno de los experimentos.

En la figura 4.8 se puede observar que a medida que pasa el tiempo, el consumo energético crece a razón aparentemente constante, este hecho inspiro la arquitectura final del modelo energético que se desarrolla en esta tesis.

Finalmente, estos vectores de muestras se anotaron con los parámetros del middleware IoTVar, lo que resulto en un dataset de series de tiempo. Este dataset se almacenó en formato JSON por

la estructura de los datos. Se muestra a continuación la estructura de dicho dataset.

```
import polars as pl
df =pl.read_ndjson('/home/han4n/2023-iotvar-hardware/Code/Phase_3/IoTVar_PowerProfiler/data/final_dat
↪  a/coefficients_fixed_b/curves/60sec_curves.json')
print(df)
```

```
refresh_period  number_sensors  energy
---             ---             ---
i64             i64             list[f64]

60              125             [0.01279, 0.019185, ... 661.7020...
60              175             [0.01789, 0.026381, ... 675.1011...
60              50              [0.013618, 0.023155, ... 660.918...
...               ...               ...
60              75              [0.021232, 0.030414, ... 677.180...
60              75              [0.01987, 0.029491, ... 671.7154...
60              50              [0.017747, 0.025758, ... 667.999...
```

Estos archivos JSON se almacenaron en una carpeta y fueron separados por periodo de actualización como se muestra a continuación.

```
cd /home/han4n/2023-iotvar-hardware/Code/Phase_3/IoTVar_PowerProfiler/data/final_data/coefficients_fi
↪  xed_b/curves
tree
```

```
10sec_curves.json
15sec_curves.json
1sec_curves.json
20sec_curves.json
3sec_curves.json
5sec_curves.json
```

```
 60sec_curves.json


0 directories, 7 files
```

4.3. Múltiples mediciones para las mismas condiciones

Los experimentos con el middleware IoTVar fueron diseñados, tomando como punto de partida la investigación previa. En un estudio seminal, Borges et al. [10] llevaron a cabo una exhaustiva evaluación del consumo energético atribuible al middleware IoTVar. Su metodología implicó una variación sistemática en el número de sensores ($nb_{V_{G_i}}$), abarcando un espectro desde 25 hasta 200, con incrementos graduales de 25 sensores en cada iteración experimental. Este enfoque metódico permitió a Borges et al. [10] recopilar y reportar datos detallados sobre el consumo energético para una gama de configuraciones que incluían 25, 50, 75, y así sucesivamente hasta alcanzar los 200 sensores. En el protocolo experimental, el middleware IoTVar fue programado para solicitar datos al broker correspondientes a cada sensor con una frecuencia de una vez por segundo.

En el contexto de la presente tesis, se adoptó el mismo rango de sensores y el mismo incremento, manteniendo así una continuidad metodológica con la investigación precedente. No obstante, se introdujo una variable adicional de considerable importancia: la variación del periodo de actualización (R_{G_i}). Este nuevo parámetro fue cuidadosamente calibrado para incluir periodos de 1, 3, 5, 10, 15, 20 y 60 segundos. La selección de estos intervalos se fundamentó en observaciones preliminares sobre la variabilidad del consumo energético. Se constató que los periodos de actualización comprendidos entre 1 y 10 segundos exhibían una mayor fluctuación en los datos de consumo, mientras que aquellos superiores a 20 segundos demostraban una estabilidad notablemente superior. Con el objetivo de robustecer la validez estadística de los resultados y mitigar el impacto del ruido en las mediciones, cada combinación específica de número de sensores y periodo de actualización fue sometida a 25 repeticiones.

Las curvas de consumo energético para el conjunto de mediciones correspondientes a 150 sensores y periodo de actualización de 10 segundos se ven de la siguiente manera.

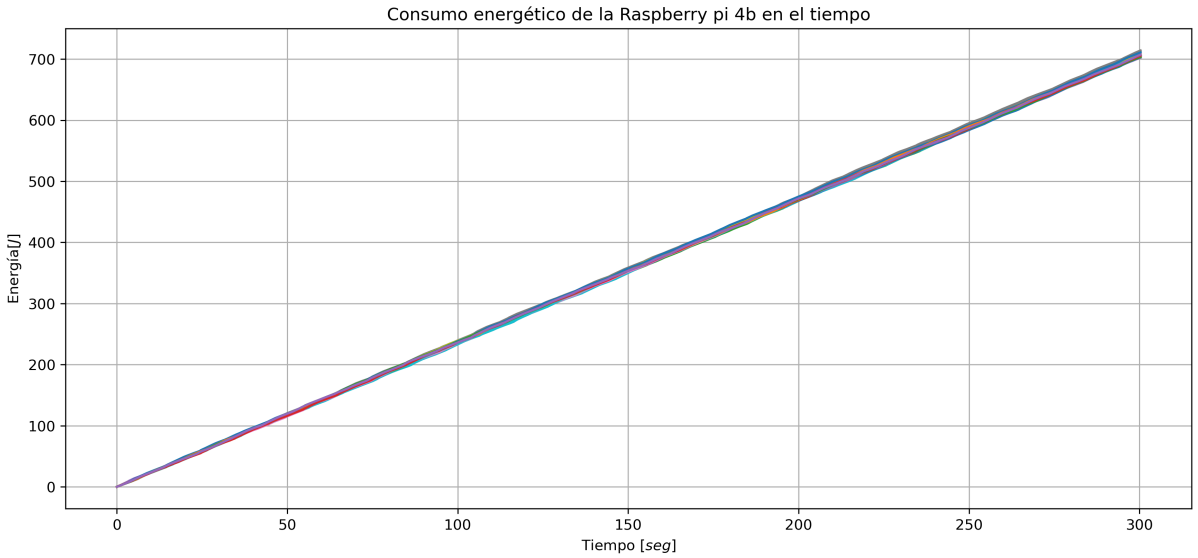

Figura 4.9: Evolución del consumo energético de la Raspberry Pi 4b para el grupo de experimentos $nb_{V_{G_i}} = 150$, $R_{G_i} = 10$.

Acercando a la figura 4.9 se tiene.

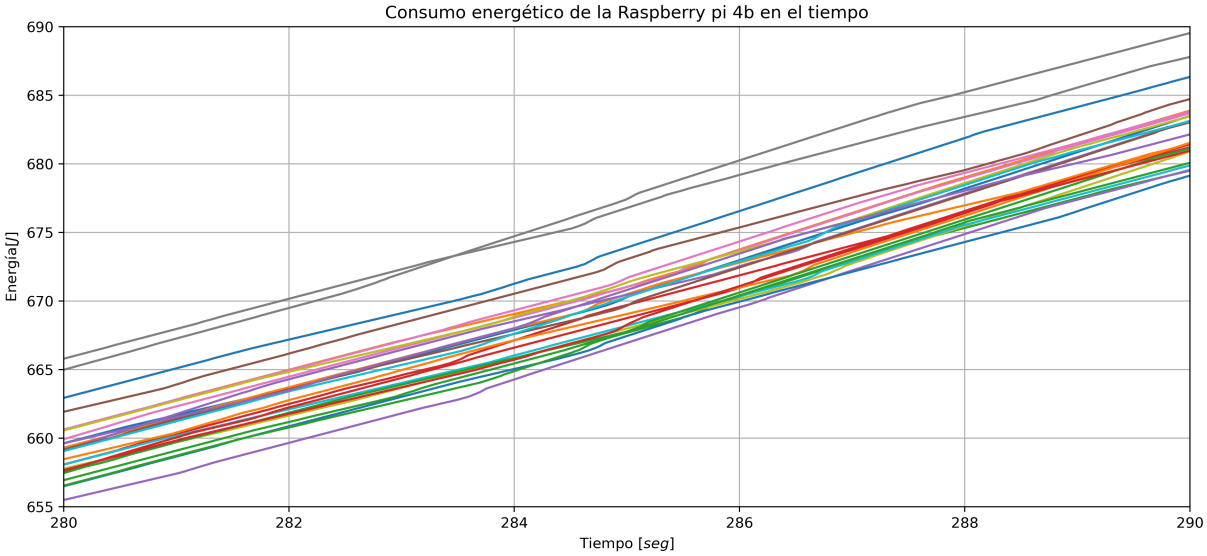

Figura 4.10: Evolución del consumo energético de la Raspberry Pi 4b para el grupo de experimentos $nb_{V_{G_i}} = 150$, $R_{G_i} = 10$, instantes $t = 280$ al $t = 290$.

El acercamiento mostrado en la figura 4.10 muestra un fenómeno muy particular. Las mediciones con un mismo grupo de parámetros ($nb_{V_{G_i}} = 150$, $R_{G_i} = 10$) tienen una distribución que varía en el tiempo. Este hecho no es trivial porque implica el desarrollo de modelos complejos para estimar funciones de densidad de probabilidad dependientes del tiempo, se deja la exploración de dichos modelos a discreción del lector. Para el desarrollo de esta tesis se vio por conveniente simplificar el problema tomando una serie de aproximaciones lineales para cada

grupo de experimentos.

4.4. Aproximaciones lineales

Todos los conjuntos de experimentos se simplificaron haciendo aproximaciones lineales, para el conjunto ($nb_{V_{G_i}} = 150$, $R_{G_i} = 10$) estas aproximaciones se ven de la siguiente manera.

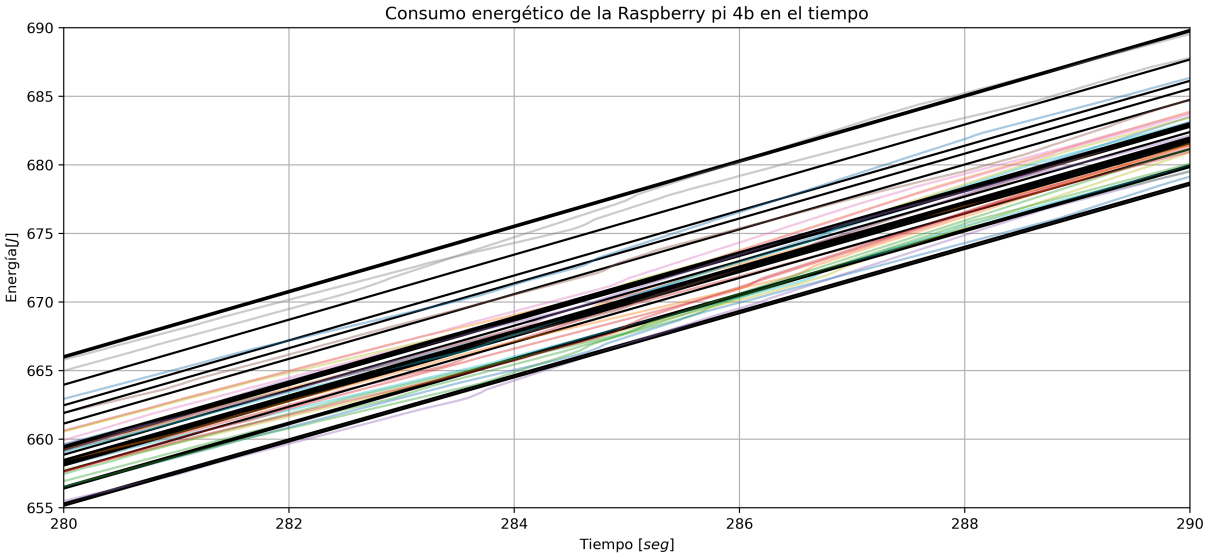

Figura 4.11: Evolución del consumo energético de la Raspberry Pi 4b para el grupo de experimentos $nb_{V_{G_i}} = 150$, $R_{G_i} = 10$, instantes $t = 280$ al $t = 290$ y aproximaciones lineales.

Para evaluar que tan buenas son estas aproximaciones lineales se usó el coeficiente de determinación R^2, este coeficiente tiene el valor de 1 cuando la regresión lineal es perfecta. En general se busca que el coeficiente de determinación sea lo más cercano a 1. El coeficiente de determinación se calcula de la siguiente manera.

$$\begin{aligned} R^2 &= 1 - \frac{SSR}{SST} \\ SSR &= \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \\ SST &= \sum_{i=1}^{n} (y_i - \bar{y})^2 \end{aligned} \tag{4.4}$$

Calculando la distribución de R^2 para uno de los grupos de experimentos se tiene.

En la figura 4.12 se puede observar que los valores del coeficiente de determinación con muy

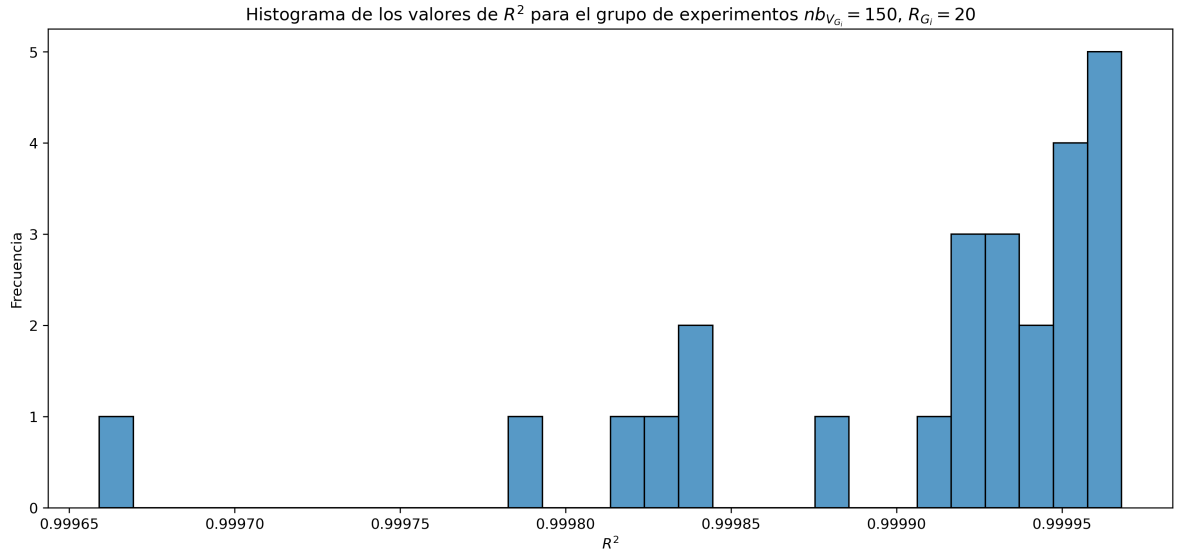

Figura 4.12: Distribución de los valores de R^2 para el grupo de experimentos $nb_{V_{G_i}} = 150$, $R_{G_i} = 20$.

cercanos a 1. Distribuciones similares fueron encontradas para todos los conjuntos. De este hecho se puede afirmar que las aproximaciones lineales son buenas.

Además de los valores de R^2 las distribuciones de la pendiente (m) y la intersección (b) son también de interés. Primero se observarán los valores que toma las pendientes (razón de cambio de la energía).

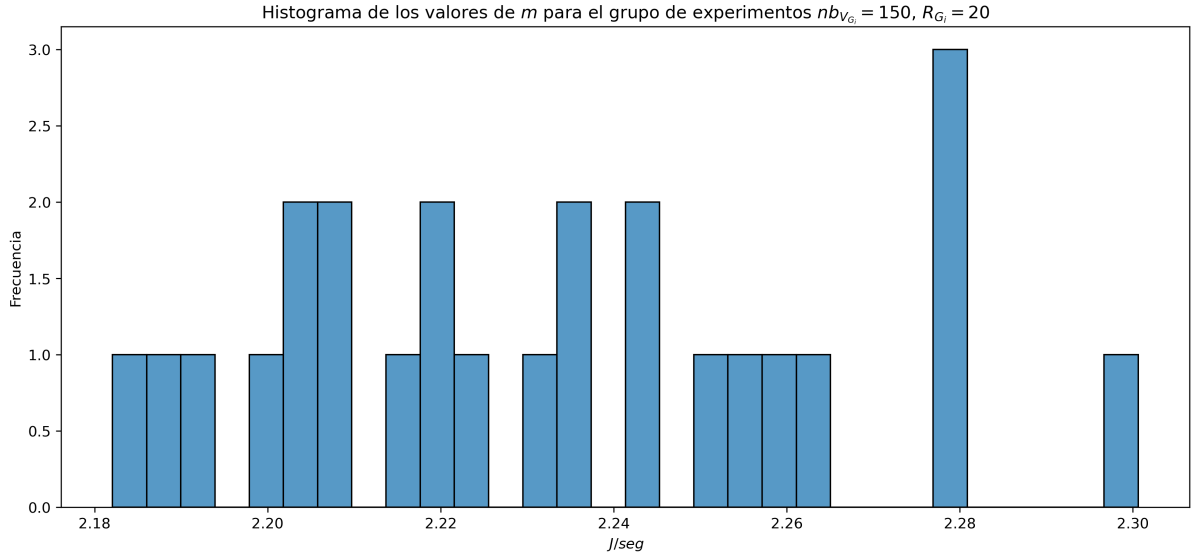

Figura 4.13: Distribución de los valores de m para el grupo de experimentos $nb_{V_{G_i}} = 150$, $R_{G_i} = 20$.

Para este grupo de experimentos la razón de cambio a la que la energía aumenta tiene un rango que aumenta desde los 2,18$\frac{J}{seg}$ hasta los 2,30$\frac{J}{seg}$. Además, se puede observar que los datos

tienen una distribución aparentemente uniforme. A continuación, se muestra la distribución de los valores de b (consumo energético en $t = 0$).

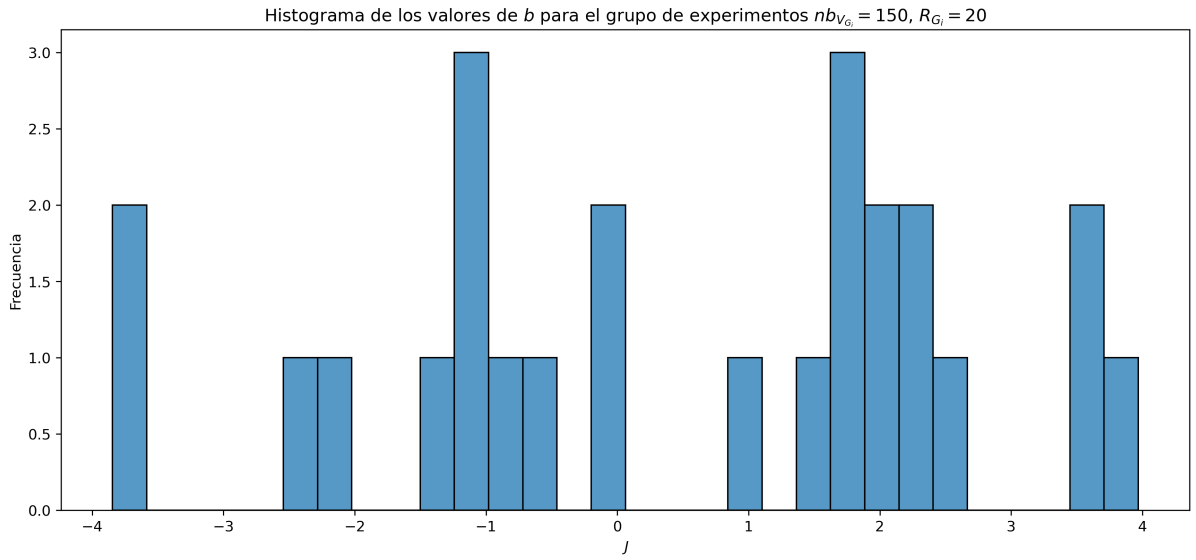

Figura 4.14: Distribución de los valores de b para el grupo de experimentos $nb_{V_{G_i}} = 150$, $R_{G_i} = 20$.

La distribución de los valores de b es muy interesante. La distribución presenta valores negativos, esto desde un punto de vista físico es imposible, ya que no existe consumo energético negativo, a no ser que durante el proceso se genere energía en cuyo caso el signo negativo indicaría que la energía se entrega en vez de ser consumida. Sin embargo, por las conexiones mostradas en la figura 4.3 no existe ningún elemento que genere energía. Además, según el procedimiento descrito en la ecuación 4.3, para calcular el primer dato de consumo energético en $t = 1$ se usaron W_0 y W_1. El cálculo de b en $t = 0$ hubiera requerido de W_{-1}, este es el valor de la potencia un instante antes del inicio del experimento. No tiene sentido incluir mediciones de potencia de antes de experimento debido a que estas no están afectadas por IoTVar.

Entonces, ¿Cuál es el origen de estos valores negativos?. Para responder a esta incógnita se tiene que entender como funciona el algoritmo de minimización que calcula m y b dados un conjunto de datos (mediciones de consumo energético).

Mediante el método de mínimos cuadrados, se logra ajustar un modelo a los datos experimentales garantizando que la suma de los cuadrados de las diferencias entre las observaciones reales y las predicciones del modelo alcance su valor mínimo. En el contexto de la aproximación lineal considerada en esta tesis, la expresión adoptada es la siguiente:

$$y_i = b + mx_i \tag{4.5}$$

Donde:

- y_i representa el valor medido u observado.
- x_i corresponde a la variable independiente.
- m y b son los parámetros del modelo que se desean determinar.

Mediante el método de mínimos cuadrados se busca establecer los valores óptimos de m y b que reduzcan al mínimo la suma de los cuadrados de los residuales. Dicho planteamiento se expresa como un problema de optimización, orientado a minimizar la función de costo asociada $J(b, m)$, que está definida como:

$$J(b, m) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{n} (y_i - (b + mx_i))^2 \tag{4.6}$$

Donde:

- $y_i = b + m \times x_i$ es el valor predicho por el modelo.

Para encontrar los valores óptimos de m y b, debemos derivar la función de costo $J(b, m)$ con respecto a m y b; y establecer las derivadas iguales a cero. Esto nos lleva a las siguientes derivadas parciales:

- Derivada con respecto a b:

$$\frac{\partial J(b, m)}{\partial b} = -2 \sum_{i=1}^{n} (y_i - b - mx_i) \tag{4.7}$$

- Derivada con respecto a m:

$$\frac{\partial J(b, m)}{\partial m} = -2 \sum_{i=1}^{n} x_i (y_i - b - mx_i) \tag{4.8}$$

Al establecer en cero las derivadas parciales y resolver el sistema de ecuaciones resultante, se obtienen las estimaciones óptimas de m y b.

$$
\begin{aligned}
m &= \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n} (x_i - \bar{x})^2} \\
b &= \bar{y} - m\bar{x}
\end{aligned}
\tag{4.9}
$$

Donde $\bar{x}$ y $\bar{y}$ son las medias de los valores de x y y, respectivamente.

Para que b sea negativo de acuerdo a la ecuación 4.9 se tiene que cumplir la siguiente condición.

$$
\begin{aligned}
\bar{y} - m\bar{x} &< 0 \\
m\bar{x} &> \bar{y} \\
m &> \frac{\bar{y}}{\bar{x}}
\end{aligned}
\tag{4.10}
$$

Según la ecuación 4.10 basta que la razón de cambio (m) sea más grande que la división entre los valores medios de las mediciones de consumo energético ($\bar{y}$) y el vector de tiempo ($\bar{x}$). Esta condición se cumple para varios experimentos y es el motivo por el cual se tienen valores de energía en $t = 0$ negativos.

Del análisis anterior se puede afirmar que los valores negativos de b no tienen significado físico, sino que son valores virtuales generados por el algoritmo de regresión lineal. Para eliminar estos valores virtuales se decidió añadir un desplazamiento temporal a las mediciones, dicho desplazamiento temporal tiene por objetivo hacer que el consumo energético en el instante $t = 0$ sea un valor medido físicamente.

Después del desplazamiento temporal las regresiones lineales se convierten de $E(t)$ a $E(t+T)$. Además con este cambio de fija el valor de b a E_1 para todos los casos. Entonces las nuevas aproximaciones lineales tienen la siguiente forma.

$$
E(t + T) = m \times (t + T) + E_1 \tag{4.11}
$$

Donde:

- T es el periodo de muestreo de los datos.
- t es el instante de tiempo en el cual se requieren las estimaciones.
- E_1 es la energía que consume la Raspberry Pi 4b desde el instante $t = 0$ al instante $t = T$ (ecuación 4.3)

Esta nueva ecuación requiere de la modificación del algoritmo de mínimos cuadrados mostrado anteriormente. La modificación requiere que el parámetro b se fije y que el algoritmo solo elija el valor de m. Esta modificación se realizó usando la función $\text{curve}_{\text{fit}}()$ de la librería SciPy de Python. El procedimiento se realizó usando el siguiente script.

Listing 7 Regresion lineal fijando el parametro b

```
from scipy.optimize import curve_fit
def lin_fit(t,m,b):
    return m*t+b
popt, pcov = curve_fit(lambda t, m: lin_fit(t, m, b_fixed), t,vectors[i])
```

Después de esta modificación al algoritmo de regresión lineal se tiene siguiente distribución de los valores de b.

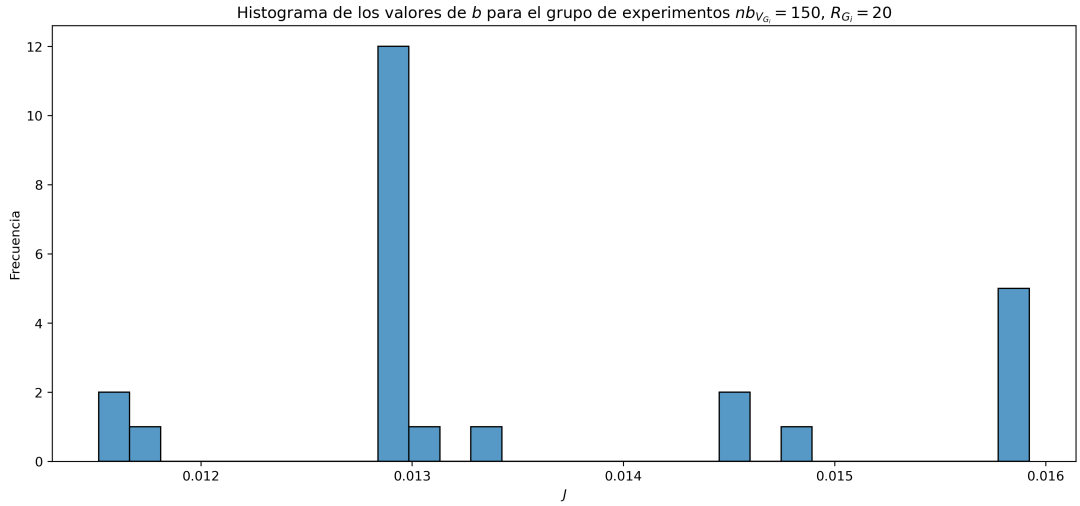

Figura 4.15: Distribución de los valores de b para el grupo de experimentos $nb_{V_{G_i}} = 150$, $R_{G_i} = 20$, fijando el valor de b.

La figura 4.15 ilustra cómo se distribuyen los valores de b al fijar estos valores a E_1. A diferencia de los valores mostrados en la figura 4.14, estos valores tienen significado físico lo cual es importante para el desarrollo del modelo energético objeto de esta tesis.

Listas las aproximaciones lineales, se obtuvieron las distribuciones de m y b para todos las

combinaciones de periodo de actualización $(1, 3, 5, 10, 15, 20, 60$ segundos) y todas las cantidades de sensores $(25, 50, 75, 100, 125, 150, 175, 200)$.

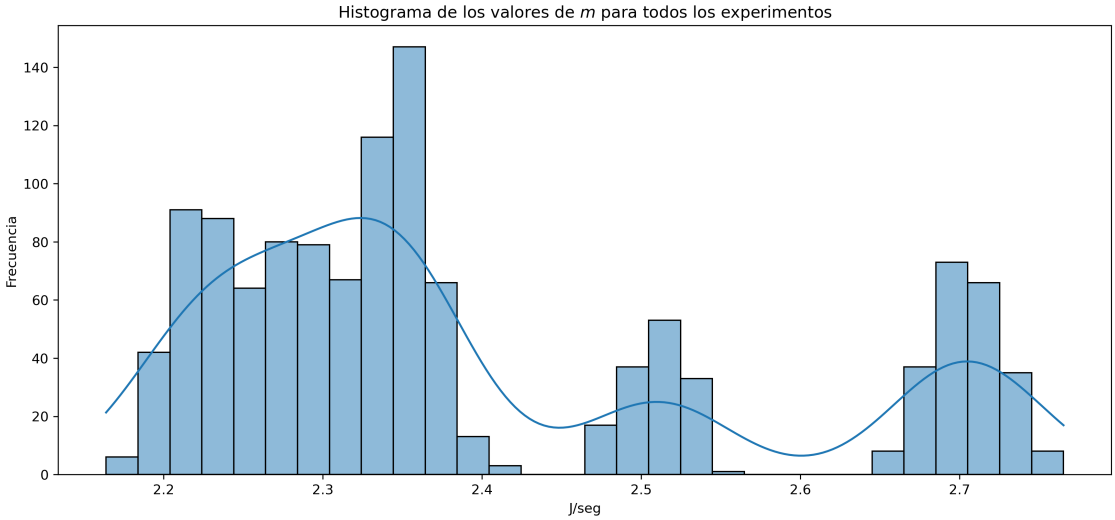

Figura 4.16: Distribución de todos los valores de m

La distribución de datos de la figura 4.16 muestra tres grupos de valores de las pendientes, el primero alrededor de 2,3$\frac{J}{seg}$ el segundo alrededor de 2,5$\frac{J}{seg}$ y el tercero alrededor de 2,7$\frac{J}{seg}$, mas adelante se verá que estos grupos están relacionados a los parámetros de IoTVar.

También se muestra a continuación la distribución de todos los valores de b.

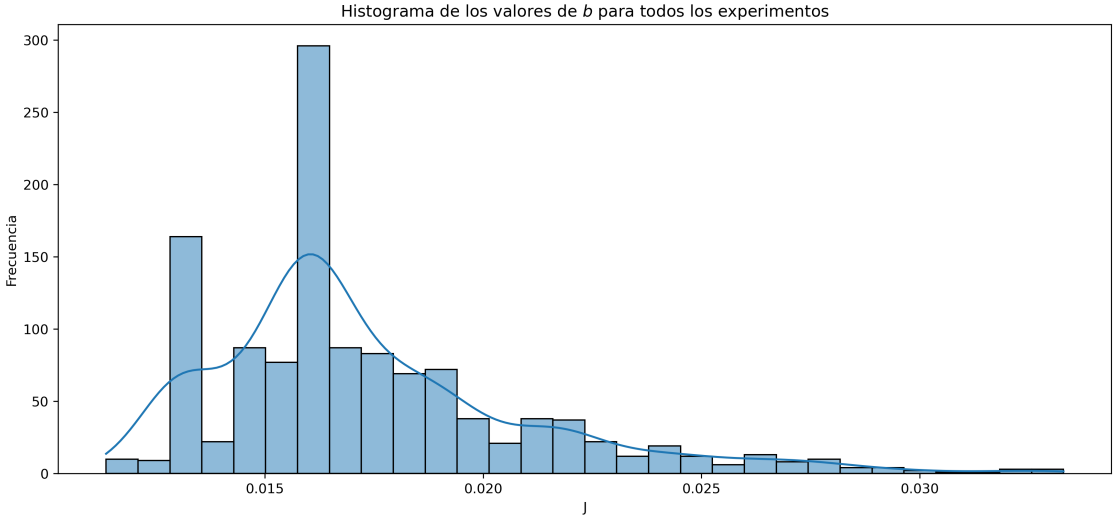

Figura 4.17: Distribución de todos los valores de b

La distribución de los valores de b mostrada en la figura 4.17 presenta una distribución unimodal. Como se indico anteriormente, estos valores se fijaron a E_1 el cual es la primera lectura de consumo energético calculada.

4.5. Regresión cuantil no paramétrica y redes neuronales

Llegados a este punto ya se puede comenzar a construir el modelo energético para estimar el consumo energético en un instante t en el futuro. Como etapa inicial en la elaboración del modelo, se realiza una simplificación adicional. Para entender la motivación de esta simplificación veamos nuevamente la figura 4.10. Esta figura muestra que para instante de tiempo la distribución de los datos es distinta. Para ilustrar este punto grafiquemos la figura 4.10 con menos puntos y comparemos las distribuciones usando el diagrama de caja de cada distribución.

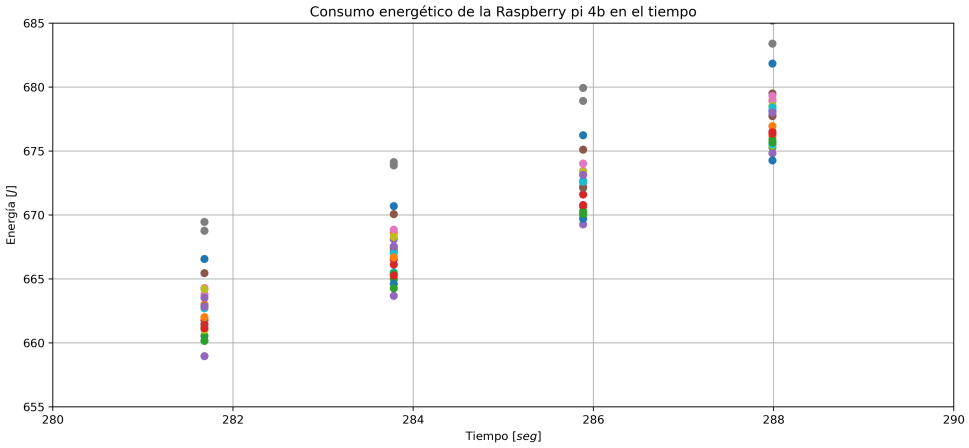

Figura 4.18: Instantes del consumo energético de la Raspberry 4b para el grupo de experimentos $nb_{V_{G_i}} = 150$, $R_{G_i} = 10$, instantes $t = 280$ al $t = 290$.

La figura 4.18 es el resultado de hacer un sub muestreo a las curvas de consumo energético. Ahora grafiquemos el Diagrama de Caja para estos instantes de tiempo.

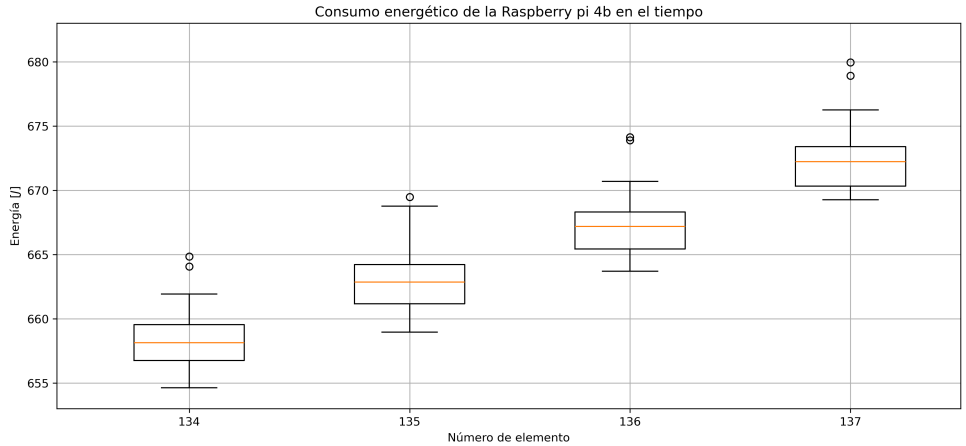

Figura 4.19: Diagrama de caja para instantes del consumo energético de la Raspberry Pi 4b para el grupo de experimentos $nb_{V_{G_i}} = 150$, $R_{G_i} = 10$, instantes $t = 280$ al $t = 290$.

La figura 4.19 muestra que cada instante de tiempo tiene una mediana que crece linealmente. Así también los cuartiles crecen linealmente. Por otro lado, cada instante de tiempo presenta distintos valores atípicos.

Anteriormente, se mencionó también que intentar estimar distribuciones dependientes del tiempo implica el uso de modelos complejos, es por ello que la simplificación que se plantea para el desarrollo del modelo energético de esta tesis consiste en estimar percentiles de la distribución en vez de estimar la distribución completa. Los percentiles elegidos fueron el quinto percentil (5 %), el cincuentavo percentil (50 %) y el noventa y quinto percentil (95 %). Estos tres percentiles permiten estimar el valor mínimo, mediana y valor máximo de la distribución en cada instante de tiempo. Grafiquemos esta idea.

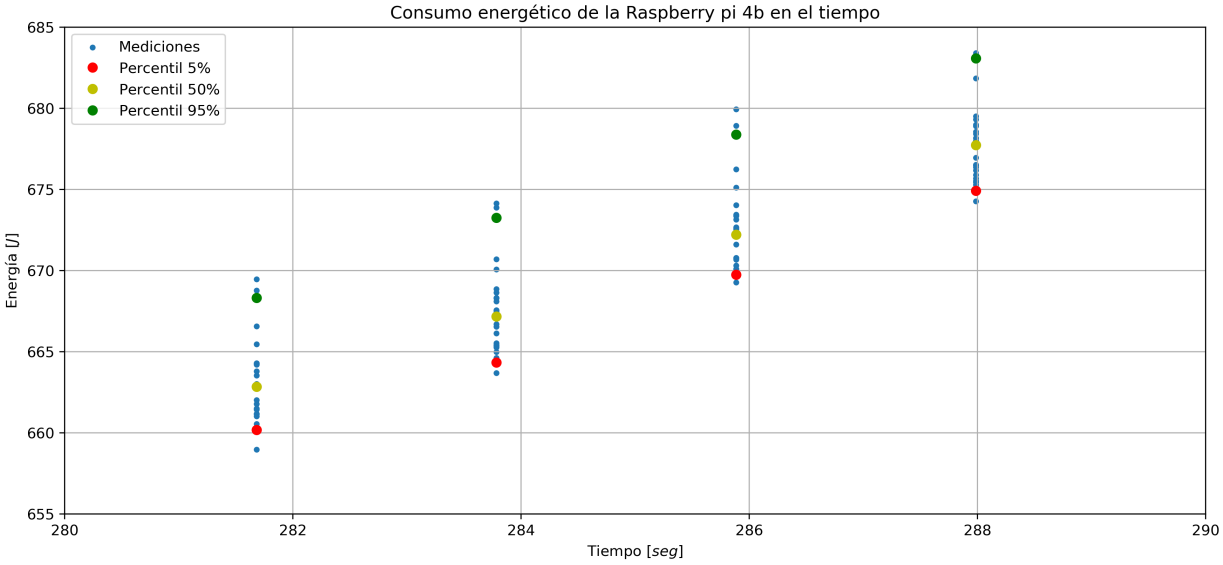

Figura 4.20: Percentiles 5 %, 50 % y 95 % para instantes del consumo energético de la Raspberry Pi 4b para el grupo de experimentos $nb_{V_{G_i}} = 150$, $R_{G_i} = 10$, instantes $t = 280$ al $t = 290$.

En la figura 4.20 se pueden observar los distintos percentiles que serán estimados por el modelo energético que se desarrollará. También se puede observar que estos tres percentiles son capaces de capturar los puntos importantes de la distribución. Además de no asumir que las distribuciones sigan alguna distribución conocida como la normal para realizar estimaciones.

Concentrémonos ahora en el modelo energético. Anteriormente, se planteó que las regresiones lineales con un desplazamiento temporal tienen la forma de la ecuación 4.11, y se mencionó que los parámetros conocidos de IoTVar que se variaron en los experimentos fueron el periodo de actualización R_{G_i} y el número de sensores $nb_{V_{G_i}}$.

Lo que se busca es modelar el consumo energético en función de los parámetros $nb_{V_{G_i}}$ y R_{G_i}, construyendo una función $E(t, nb_{V_{G_i}}, R_{G_i})$. En la ecuación 4.11 se vio que la dinámica del consumo energético se puede modelar usando una función dependiente del tiempo de la forma $E(t+T)$. Entonces, lo que queda es relacionar los parámetros $nb_{V_{G_i}}$ y R_{G_i} al miembro restante de la ecuación 4.11, la pendiente m. Haciendo que la pendiente m sea función de $nb_{V_{G_i}}$ y R_{G_i} da como resultado una nueva función $g(nb_{V_{G_i}}, R_{G_i})$. La introducción de esta nueva función a la ecuación 4.11 resulta en una nueva función $E(t+T, g(nb_{V_{G_i}}, R_{G_i}))$. Dicha función se muestra a continuación.

$$E(t+T, nb_{V_{G_i}}, R_{G_i}) = g(nb_{V_{G_i}}, R_{G_i}) \times (t+T) + E_1 \tag{4.12}$$

Nótese que la ecuación 4.12 presenta el cambio de la pendiente m por la función $g(nb_{V_{G_i}}, R_{G_i})$ en la ecuación 4.11.

La ecuación 4.12 muestra la forma casi terminada del modelo energético. La ultima modificación que se realizo a la ecuación 4.12 fue la eliminación del término E_1. Se decidió prescindir de dicho término debido a que este término tiende a cero para frecuencias de muestreo suficientemente altas, esto se muestra en el siguiente procedimiento 4.13.

$$\begin{aligned} E_1 &= \frac{W_0 + W_1}{f} \\ f &\gg W_0 + W_1 \\ E_1 &\approx 0 \end{aligned} \tag{4.13}$$

La eliminación de este termino hace que la ecuación 4.12 tome la siguiente forma.

$$E(t+T, nb_{V_{G_i}}, R_{G_i}) = g(nb_{V_{G_i}}, R_{G_i}) \times (t+T) \tag{4.14}$$

La ecuación 4.14 presenta el modelo energético objetivo de esta tesis. El modelo planteado tiene como núcleo a la función $g(nb_{V_{G_i}}, R_{G_i})$. La función $g(nb_{V_{G_i}}, R_{G_i})$ mapea los parámetros de IoTVar a las pendientes m observadas en los datos (figura 4.16). Observemos el mapeo de la

función $g(nb_{V_{G_i}}, R_{G_i})$.

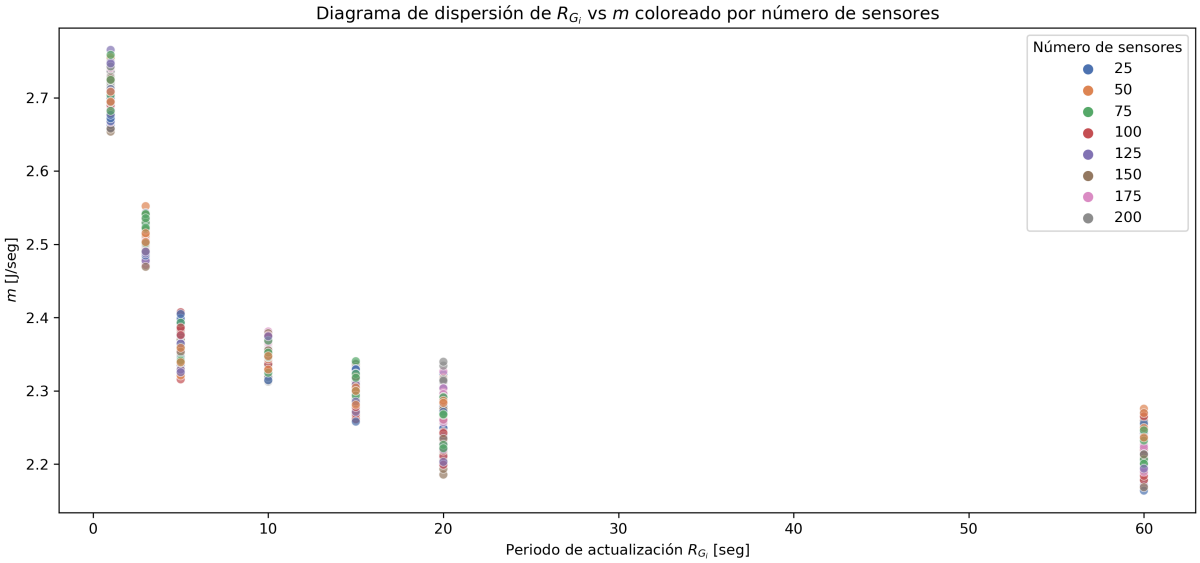

Figura 4.21: Gráfica m vs R_{G_i}

La figura 4.21 muestra como varían los valores de m cuando varía R_{G_i}. Se puede observar una relación de proporcionalidad inversa que indica que a para R_{G_i} pequeños la razón de cambio de la energía será menor. Este comportamiento es esperado debido a que periodos de actualización altos indican que el middleware IoTVar realizó solicitudes al broker con mayor frecuencia. Esto conlleva a que los subsistemas usados en la comunicación y procesamiento de los datos se usaron más veces y, por lo tanto, se utilizó más energía para su funcionamiento. El mismo fenómeno se puede observar cuando R_{G_i} es alto, esto quiere decir que las solicitudes al broker se hicieron con menor frecuencia y esto llevo a un menor consumo energético. Adicionalmente, se puede observar que aproximadamente a partir de $R_{G_i} = 20$ la razón de cambio en la energía se estabiliza con un valor medio de aproximadamente $2{,}2\frac{J}{seg}$.

Ahora usemos el número de sensores como predictor de m.

Al igual que en la figura 4.21, la figura 4.22 muestra el efecto de R_{G_i} en m. Adicionalmente, se puede ver la distribución de m para todos los números de sensores vistos en los experimentos, esta distribución es similar para todos los experimentos, con algunas diferencias en $nb_{V_{G_i}} = 125$.

La estimación de percentiles para un conjunto de datos se conoce en la literatura como regresión cuantil no paramétrica. Esta se puede integrar como función de perdida para entrenar redes neuronales logrando capacidades de generalización para una gran variedad de problemas.

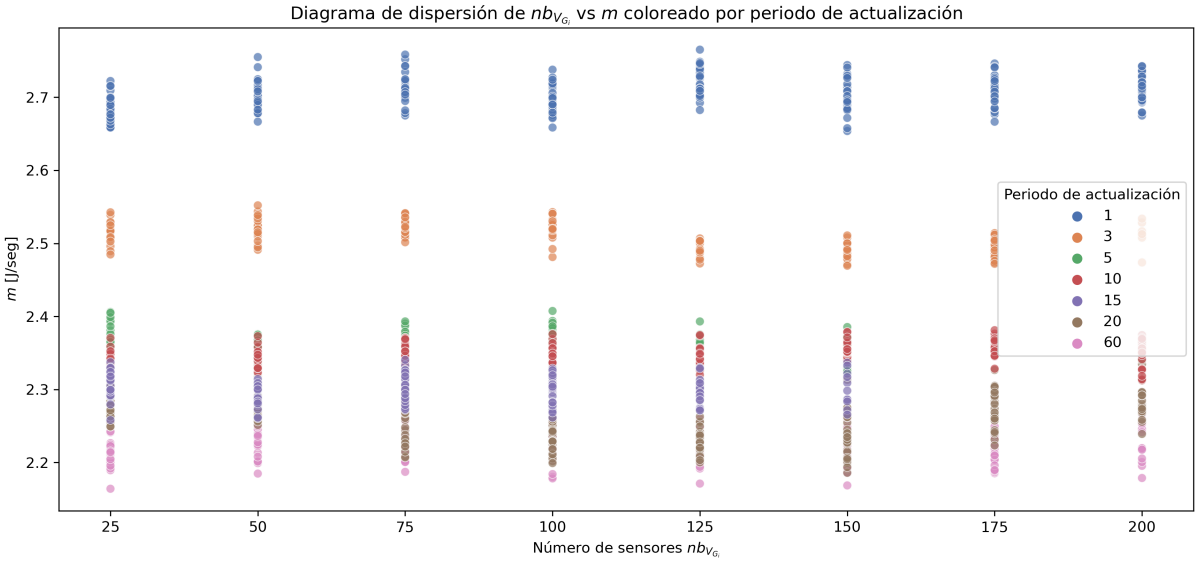

Figura 4.22: Gráfica $nb_{V_{G_i}}$ vs m

En el contexto del desarrollo de esta tesis se vio por conveniente usar este método para obtener la función $g(nb_{V_{G_i}}, R_{G_i})$. La función $g(nb_{V_{G_i}}, R_{G_i})$ buscada dará los percentiles 5 %, 50 % y 95 % para cada par $(nb_{V_{G_i}}, R_{G_i})$ de entrada que reciba.

El diseño del modelo de red neuronal incluyó dos neuronas en la capa de entrada $(nb_{V_{G_i}}, R_{G_i})$, dos capas ocultas con la función de activación estándar (Leaky ReLU) y tres neuronas de salida (percentiles 5 %, 50 % y 95 %). Las tres neuronas de salida usaron la función de perdida pinball (figura 2.12) con el parámetro τ correspondiente al percentil deseado. Adicionalmente, se utilizaron las librerías de TensorFlow y Keras para la implementación de la red. El proceso de ajuste hiperparámetros de la red (tasa de aprendizaje, tamaño de batch, número de épocas, número de neuronas de capas ocultas) se realizó usando la herramienta *optuna* [11]. *Optuna* es una herramienta de búsqueda automática de hiperparámetros en un espacio de búsqueda determinado, como se buscó que la red sea lo mas pequeña posible, el número de capas ocultas se fijó a dos, ya que fue el número mínimo de capas ocultas que capturaban correctamente la relación entre m y los parámetros de entrada.

Antes del entrenamiento, los valores de m fueron normalizados eliminando su media y escalando a varianza unitaria para mejorar la estabilidad del aprendizaje. Posteriormente, el conjunto de datos se dividió en entrenamiento (80 % de los registros) y prueba (20 % restante). La arquitectura final de la red que modela $g(nb_{V_{G_i}}, R_{G_i})$ se presenta a continuación.

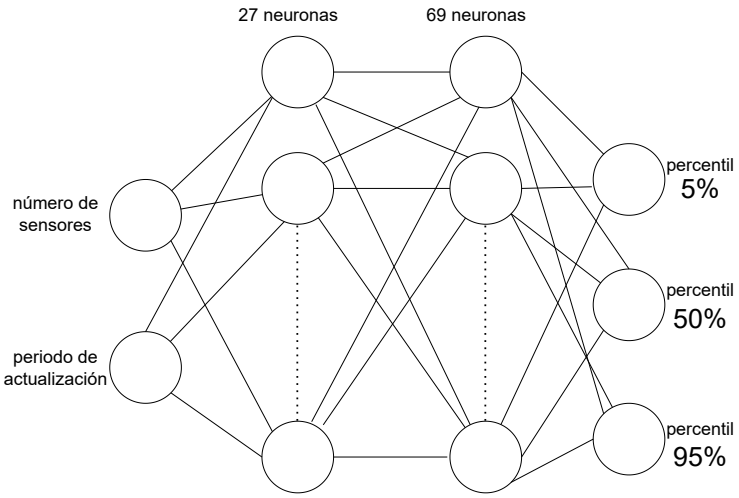

Figura 4.23: Arquitectura de la red neuronal que modela la función $g(nb_{V_{G_i}}, R_{G_i})$

La red mostrada en la figura 4.23 se entrenó con los siguientes hiperparámetros.

Tabla 4.1: Valores numéricos de los hiperparámetros encontrados usando *optuna* [11]

Hiperparámetro	**Valor numérico**
Número de épocas	200
Tasa de aprendizaje	0.011379105781918687
Número de neuronas en primera capa oculta	27
Número de neuronas en segunda capa oculta	69
Tamaño de batch	25

Estos parámetros dieron las curvas de evoluciones de las funciones de perdida para cada una de las tres neuronas de salida mostradas en la figura 4.24.

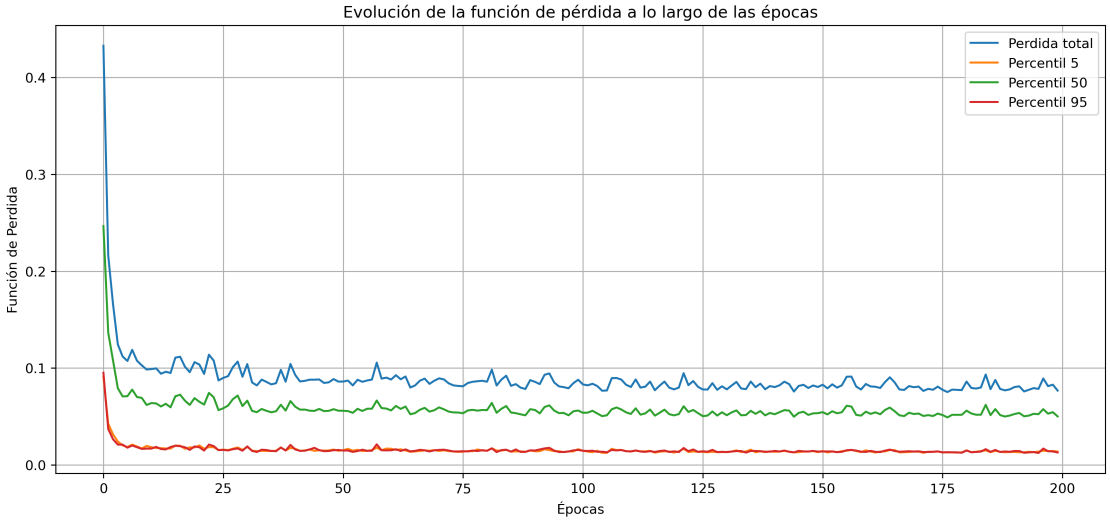

Figura 4.24: Desarrollo del entrenamiento de la red

En la figura 4.24 se muestra como se estabilizan las funciones de perdida para cada una de las

neuronas de salida. Adicionalmente, se aprecia la curva de perdida total que se calcula sumando los valores de las tres funciones de perdida usadas en cada neurona, esta curva se estabiliza en la época 50.

4.6. Evaluación de $g(nb_{V_{G_i}}, R_{G_i})$

Para evaluar la red que mapea la función $g(nb_{V_{G_i}}, R_{G_i})$ se usaron las siguientes métricas.

1. **Pérdida promedio de la función pinball** Esta métrica se emplea para valorar el rendimiento de modelos de regresión cuantílica. Su cálculo consiste en obtener el error medio entre las predicciones y los valores reales, aplicando una penalización asimétrica que depende de si la estimación se encuentra por encima o por debajo del cuantil deseado. Resulta especialmente adecuada para evaluar la precisión de las predicciones en distintos percentiles.

2. **Ancho medio del intervalo** Es el promedio del tamaño de los intervalos de predicción generados por un modelo. Un intervalo más estrecho indica una mayor precisión del modelo, ya que sugiere que el rango de valores predichos es más cercano al valor real. Esta métrica es clave para evaluar la confianza y la precisión de las predicciones realizadas por modelos estadísticos y de aprendizaje automático.

3. **Probabilidad de cobertura al** 90 % Es una métrica que indica la frecuencia con la que los intervalos de predicción contienen el valor real del parámetro que se está estimando. Si un modelo tiene una probabilidad de cobertura del 90 %, significa que el 90 % de los intervalos de predicción generados deberían incluir el valor verdadero, lo que refleja la fiabilidad del modelo en la estimación de intervalos.

De acuerdo a Quispe [59] la evaluación se realizó entrenando diez veces la red, reportando la media y la desviación estándar para cada métrica. Esta es una práctica recomendada para garantizar que las métricas de rendimiento del modelo sean confiables y no solo el resultado de fluctuaciones aleatorias durante el entrenamiento. También proporciona una comprensión más completa del comportamiento del modelo en diferentes ejecuciones, lo cual es particularmente

importante cuando se trata de redes neuronales que pueden exhibir variabilidad debido a inicializaciones aleatorias y procesos de entrenamiento estocástico.A continuación se exponen los valores cuantitativos asociados a las métricas

Tabla 4.2: Métricas de rendimiento de la red neuronal

Métrica	**Media**	**Desviación Estándar**
Pérdida promedio de la función pinball	0.0116	0.0012
Ancho medio del intervalo	0.0749	0.0041
Probabilidad de cobertura al 90 %	0.8963	0.0214

La tabla 4.2 muestra los valores numéricos de las métricas elegidas. La *perdida promedio de la función pinball* muestra el valor medio y desviación estándar vistos en el entrenamiento (figura 4.24). El *ancho medio del intervalo* indica una diferencia de 0,0749$\frac{J}{seg}$ entre los percentiles 5 % y 95 %, este valor es similar al valor observado en los datos de entrenamiento (figura 4.21). Finalmente, la *probabilidad de cobertura al* 90 % se busca sea lo más cercano a 0,9, para el caso de la red esta tiene una media del 89,63 % siendo de 91,77 % en el mejor de los casos y 87,49 % en el peor de los casos.

Capítulo 5

Resultados y Evaluación

Los primeros capítulos de esta tesis detallaron como se implementó el modelo energético. Se narraron las etapas de adquisición de mediciones, procesamiento de datos y finalmente el modelado de estos datos. Este capítulo se centra en hacer las estimaciones de consumo energético usando el modelo desarrollado. Para la valoración del rendimiento del modelo, se recurrirá a las tres métricas que se utilizaron previamente para evaluar el desempeño de $g(nb_{V_{G_i}}, R_{G_i})$ con la adición del error cuadrático medio entre la mediana de las mediciones reales y la mediana obtenida por el modelo. Comparar las medianas de las mediciones es importante debido a que permite comparar directamente el valor obtenido usando el modelo con el valor en el que se encontraron el mayor número de mediciones reales.

5.1. Utilización del modelo

El uso del modelo consiste en la utilización de la ecuación 4.14. Cabe mencionar que el modelo tiene un desplazamiento en el tiempo por lo que tiene las siguientes características.

1. Las entradas del instante del tiempo tienen que llevar un offset igual al periodo de muestreo que fue usado durante toma de mediciones $(t - T)$.

2. Las estimaciones son válidas para $t > T$, instantes t en el rango $0 \leq t \leq T$ ocasionan que el modelo produzca valores de consumo energético virtuales los cuales no tienen interpretación física.

Las estimaciones las realiza la capa de energía del middleware IoTVar (figura 4.1) usando el número de sensores ($nb_{V_{G_i}}$), el periodo de actualización (R_{G_i}) y el instante ($t - T$) en el cual se desea estimar el consumo energético. Dadas estas entradas el modelo devuelve tres estimaciones de consumo energético correspondientes a los tres percentiles (5 %, 50 % y 95 %). Con propósitos de evaluación se realizarán estimaciones dentro de todo el rango de mediciones $T < t < 300$. Primeramente, se observarán las curvas en el tiempo para compararlas con las mediciones reales, seguidamente se mostrará la evolución en el tiempo de las métricas y finalmente se comparará el modelo desarrollado con métodos del estado del arte.

A continuación se muestran las mediciones reales versus las estimaciones hechas por el modelo energético.

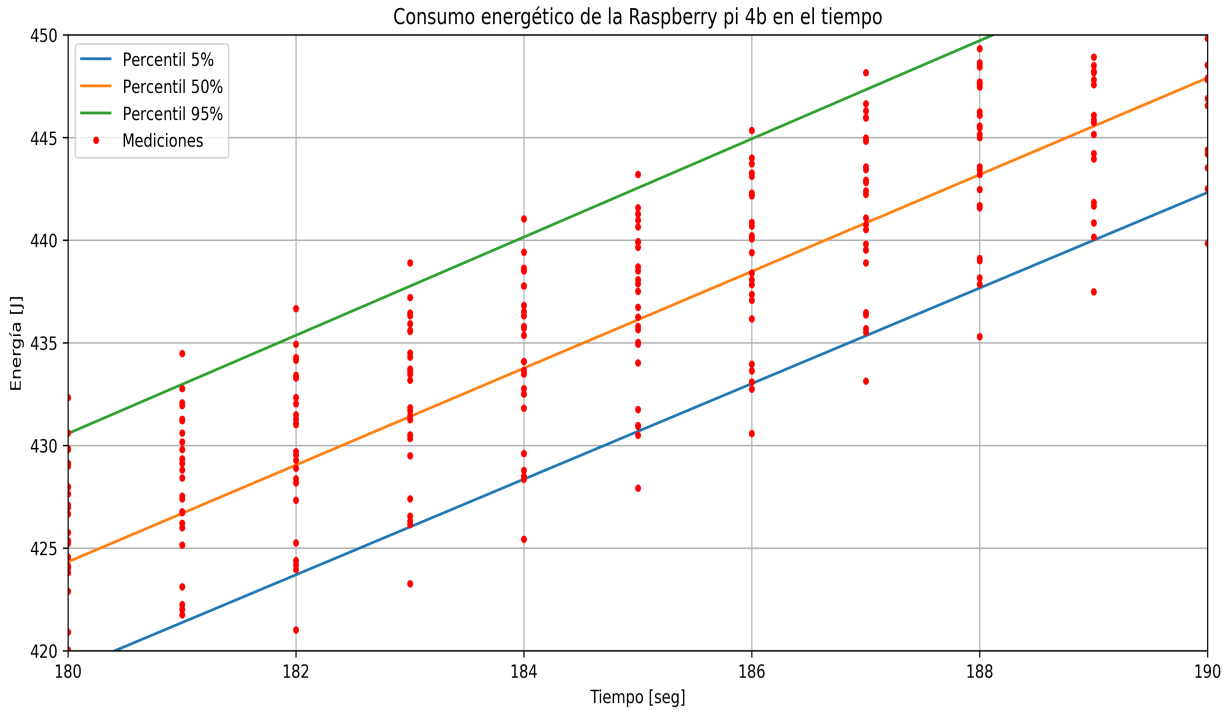

Figura 5.1: Mediciones reales y estimaciones del modelo

En la figura 5.1 se pueden apreciar los percentiles estimados por el modelo y las mediciones hechas usando el prototipo desarrollado, se puede observar que efectivamente los percentiles marcan el 5 %, 50 % y 95 % de las mediciones reales. Para medir el rendimiento del modelo ahora se procederá a extraer las métricas anteriormente mencionadas.

La primera métrica a evaluar es la *pérdida promedio de la función pinball*, esta métrica indica que tan cerca se encuentran los percentiles reales de los percentiles estimados por el modelo.

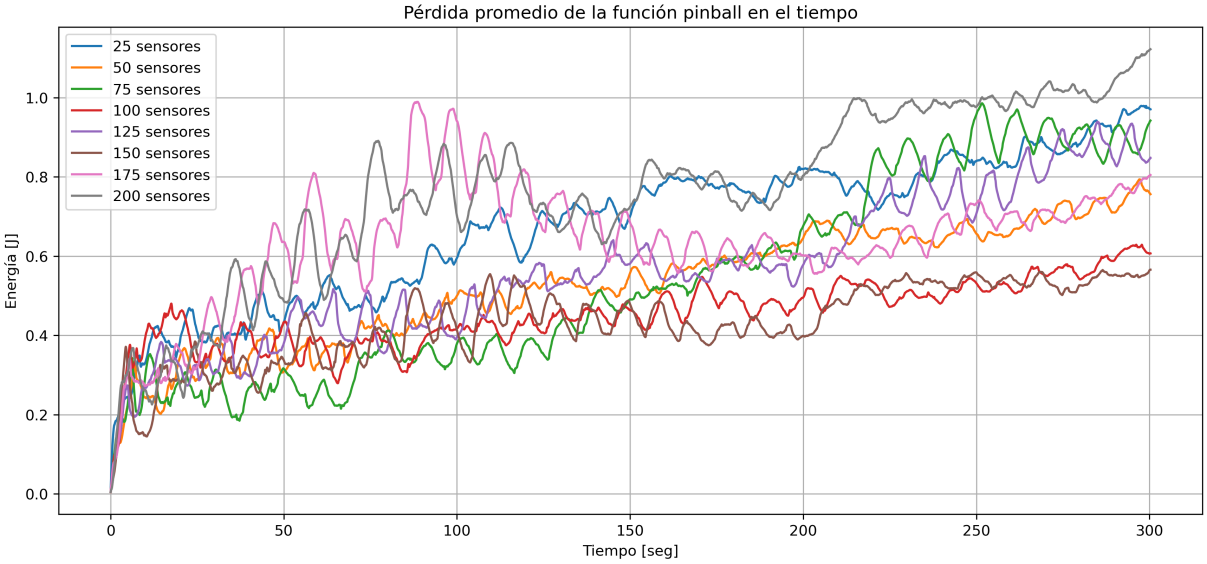

Figura 5.2: Evolución de la métrica *pérdida promedio de la función pinball* en el tiempo

La figura 5.2 muestra como evoluciona el valor promedio de la *función pinball* a lo largo del tiempo, se puede observar que a medida que pasa el tiempo los valores promedio tienden a aumentar, este comportamiento es esperable debido a que instantes t más alejados del instante de estimación tienen más incertidumbre que valores cercanos al instante en el que las estimaciones se realizan. La siguiente métrica a analizar es el *ancho medio del intervalo*.

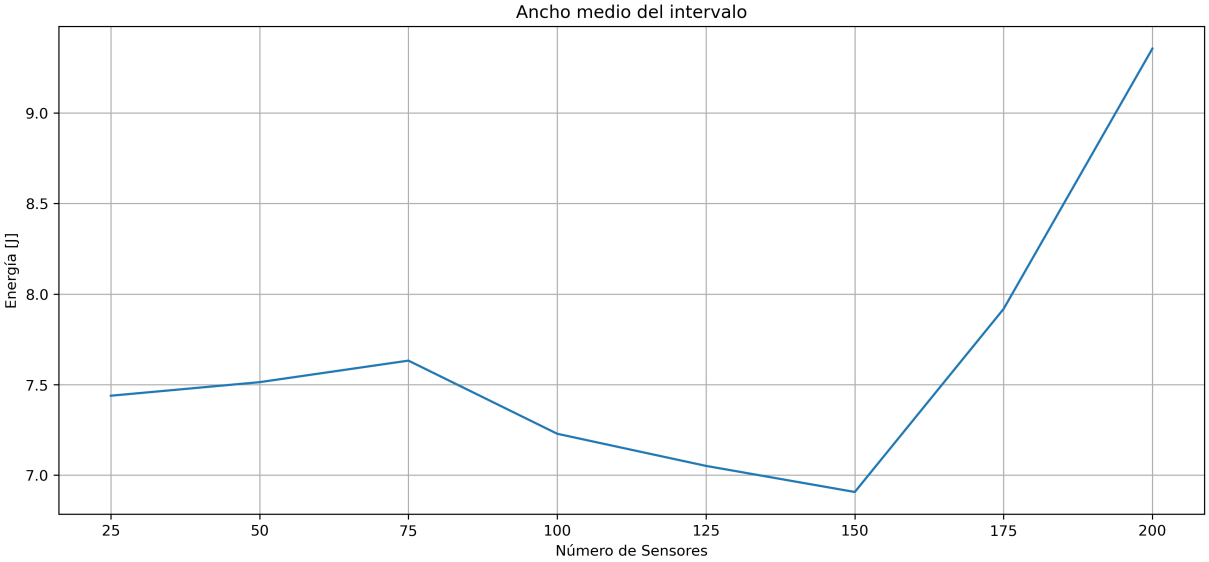

Figura 5.3: Ancho medio del intervalo para todos los números de sensores usados

La figura 5.3 muestra el valor medio de la diferencia $E_{95} - E_5$ para el intervalo de tiempo $T < t < 300$. Los valores de hasta $10J$ para 200 sensores son similares a los que presentan los datos originales. Ya que la red neuronal se entrena para imitar a los datos de entrenamiento, los

valores son esperables. La penúltima métrica a analizar es la *probabilidad de cobertura al* 90 %.

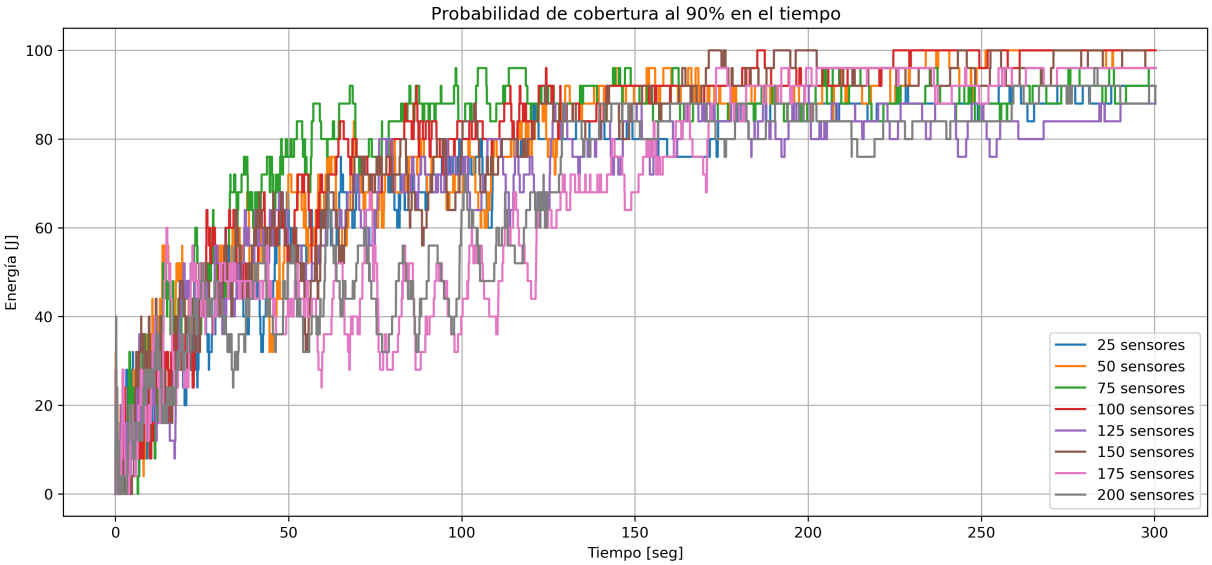

Figura 5.4: Probabilidad de cobertura al 90 % para todos los sensores

La gráfica de la probabilidad de cobertura al 90 % (figura 5.4) muestra valores bajos para los primeros instantes, esto es debido a la aproximación hecha en la ecuación 4.13. A medida que pasa el tiempo se puede observar como aumenta el valor de la métrica llegando a valores cercanos del 90 % para todos los casos. Finalmente, se tiene el error cuadrático medio entre las medianas.

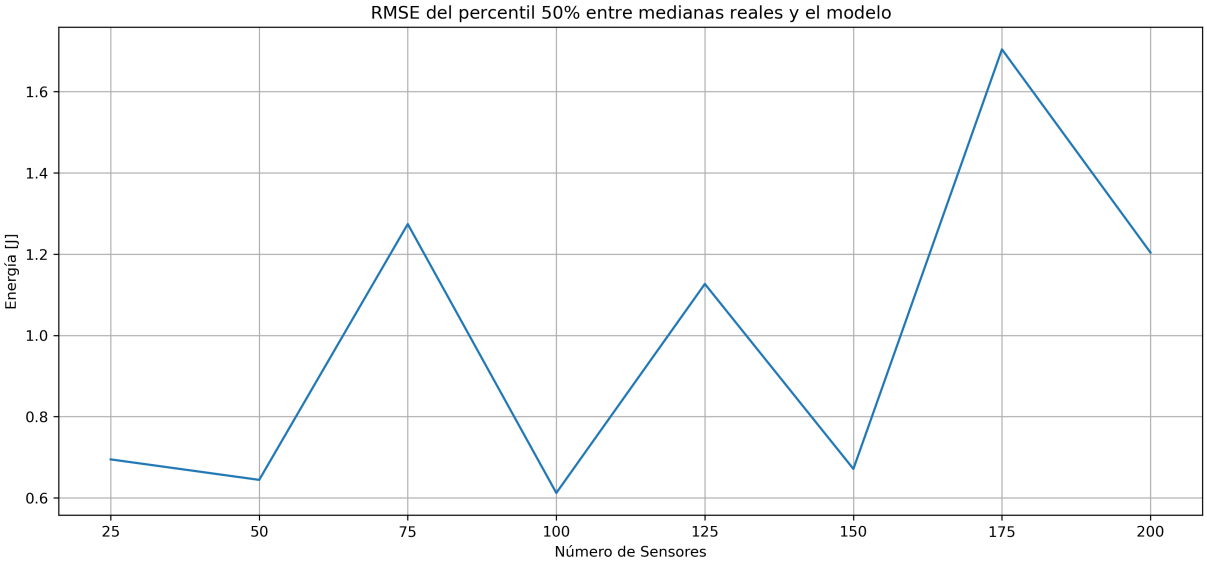

Figura 5.5: Error cuadrático medio la mediana estimada por el modelo y la mediana de las mediciones para todos los números de sensores usados.

En la gráfica 5.5 se pueden apreciar que el error cuadrático medio entre las medianas es de

1,7040440658842018J en el peor de los casos y es de 0,6121746150094582J en el mejor de los casos. Ambos valores comparado con el rango de mediciones (cientos de Joules, figura 5.1) son pequeños, esta es una muestra del buen rendimiento del modelo. También se puede observar que el error es distinto para cada número de sensores usado en la evaluación, este fenómeno abre la posibilidad de trabajos futuros que estudien el efecto del número de sensores en la precisión de las estimaciones.

5.2. Comparación con métodos del estado del arte

En la literatura se reportan diversos modelos para la estimación de consumo energético de una Raspberry. Se presenta a continuación la comparación cualitativa del modelo desarrollado en esta tesis con los modelos del estado del arte reportados.

Tabla 5.1: Comparación con metodos del estado del arte

Modelo	**Uso**	**Estimación de Consumo en el Tiempo**
Energy measurement model [29]	Estimación de consumo energético debido a ejecución de software.	no estima
Automated power modeling of computing devices [31]	Generación de modelos de consumo energético para Raspberry pi.	no estima
A Middleware Architecture for Mastering Energy Consumption in Internet of Things Applications [10]	Estimación de consumo energético para el middleware IoTVar.	no estima
Modelo de Consumo Energético Basado en Aprendizaje Automático para Middleware Embebido de IoT (esta Tesis)	Estimación de consumo energético para el middleware IoTVar.	si estima

Dentro de los métodos del estado del arte, el método propuesto por Borges et al. [10] es el único que puede ser comparado cuantitativamente con el modelo desarrollado en esta tesis. A continuación se procederá a realizar dicha comparación cuantitativa.

La comparación requiere de la replicación de resultados del estudio del estado del arte por lo que se analizarán las partes, utilización y calibración del modelo planteado por Borges et al. [10].

5.2.1. Replicación de resultados

El modelo del estado del arte tiene la siguiente estructura.

$$E = \sum_{i=0}^{nb_G} \frac{(C_V * nb_{V_{G_i}}) + (C_{net} * M_{netS} * M_{net1}) + C_{cpu}}{R_{G_i}} \tag{5.1}$$

La ecuación 5.1 tiene las siguientes variables.

- R_{G_i} es el periodo de actualización.
- $nb_{V_{G_i}}$ es el número de sensores por grupo.
- nb_G es el número de grupos (parámetro controlado por IoTVar).

Así mismo, la ecuación 5.1 tiene las siguientes constantes.

- C_V es un valor de energía constante para cada variable que está dentro de un grupo. Representa el costo del pequeño procesamiento que agrega cada variable dentro de un grupo.
- C_{net} es una constante dada para el cálculo de la red. Es la energía estimada que se consume al realizar una solicitud para un grupo.
- M_{netS} es un modificador para el cálculo de la red que depende de la latencia, pérdida de paquetes y accesibilidad de la plataforma IoT.
- M_{net1} es un modificador que depende de la interfaz de red que se utilice (WiFi o Ethernet).
- C_{cpu} es una constante dada para el procesamiento de la CPU de un grupo de variables en IoTVar.

El estudio hecho por Borges et al. [10] indica los siguientes valores numéricos para las constantes de su modelo.

Tabla 5.2: Valores numéricos de las contantes del modelo del estado del arte.

Constante	Valor
C_V	0.02
C_{net}	90
M_{netS}	1
M_{net1}	10
C_{cpu}	87

El estudio del estado del arte presenta resultados para las siguientes condiciones:

- Tiempo de ejecución = 300[seg]
- Un grupo ($nb_G = 1$)
- Periodo de actualización de un segundo ($R_{G_i} = 1$)
- Número de sensores en incrementos de veinticinco sensores $25, 50, 75, 100, 125, 150, 175$ y 200 (los mismos que fueron usados en el desarrollo de esta tesis).

Usando estas condiciones y los valores numéricos de las constantes de la tabla 5.2 Borges et al. [10] presenta la siguiente gráfica de resultados.

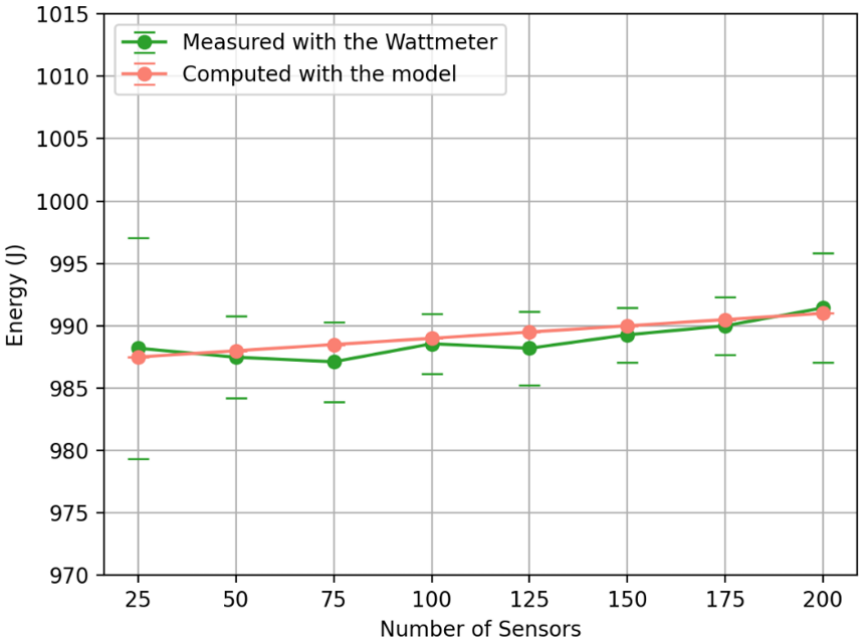

Figura 5.6: Resultados del estudio del estado del arte [10]

La figura 5.6 muestra la estimación del modelo de la ecuación 5.1 y las compara con mediciones de un vatímetro.

Tabla 5.3: Comparación del modelo del estado del arte con mediciones de un vatímetro [10]

Número de Sensores	**Vatímetro** $[J]$	**Modelo** $[J]$
25	988.2 (±8,86)	987.5
50	987.48 (±3,3)	988
75	987.12 (±3,19)	988.5
100	988.56 (±2,43)	989
125	988.2 (±2,95)	989.5
150	989.28 (±2,2)	990
175	990.0 (±2,32)	990.5
200	991.44 (±4,39)	991

La tabla 5.3 muestra los valores numéricos que se muestran en la figura 5.6, se puede apreciar también la baja desviación estándar de las mediciones que dio vatímetro Yocto-Wattmeter que los autores del estudio usaron.

Para entender los valores numéricos del modelo del estado del arte será necesario analizar la matemática del modelo del estado del arte. Debido a que en su forma general (ecuación 5.1), el modelo acepta distintos números de grupos con distintos periodos de actualización. Sin embargo, solo se presentaron resultados para un único grupo con un único periodo de actualización de un segundo. Con estas condiciones el modelo general (ecuación 5.1) toma la siguiente forma.

$$\begin{aligned} E(nb_{V_{G_i}}, R_{G_i}, nb_G) &= \sum_{i=0}^{nb_G} \frac{(C_V * nb_{V_{G_i}}) + (C_{net} * M_{netS} * M_{net1}) + C_{cpu}}{R_{G_i}} \\ E(nb_{V_{G_0}}, 1, 1) &= \sum_{i=0}^{1} \frac{(C_V * nb_{V_{G_0}}) + (C_{net} * M_{netS} * M_{net1}) + C_{cpu}}{1} \\ E(nb_{V_{G_0}}) &= C_V * nb_{V_{G_0}} + C_{net} * M_{netS} * M_{net1} + C_{cpu} \end{aligned} \tag{5.2}$$

Sea $C_1 = C_V$, $C_2 = C_{net} * M_{netS} * M_{net1} + C_{cpu}$. Entonces, el modelo energético toma la forma de una función lineal:

$$\begin{aligned} f(x) &= a * x + b \\ E(nb_{V_{G_0}}) &= C_1 * nb_{V_{G_0}} + C_2 \end{aligned} \tag{5.3}$$

Dados los valores de las constantes mostrados en la tabla 5.2. Las constantes C_1 y C_2 toman los siguientes valores numéricos

$$\begin{aligned} C_1 &= 0{,}02 \\ C_2 &= 90 * 1 * 10 + 87 = 987 \end{aligned} \tag{5.4}$$

Finalmente, el modelo del estado del arte toma la siguiente forma para los casos reportados en el estudio:

$$E(nb_{V_{G_0}}) = 0{,}02 * nb_{V_{G_0}} + 987 \tag{5.5}$$

Como se puede observar en la ecuación 5.5, el modelo del estado del arte no incluye el tiempo como variable. Siendo este capaz de entregar estimaciones de energía únicamente para el instante de tiempo $t = 300[seg]$. Esta es la principal mejora comparada con el modelo que se presenta en esta tesis, ya que el modelo desarrollado es dependiente del tiempo.

La replicación de resultados se realizó con base en la ecuación 5.5. Para ello se creó una matriz que contiene los distintos números de sensores para después aplicarle la ecuación lineal anteriormente descrita, esto da como resultado una segunda matriz que contiene las estimaciones de consumo energético. Finalmente, se graficaron estos valores con fines de comparación.

Listing 8 Script para la replicación de resultados del modelo del estado del arte

```
import numpy as np
import matplotlib.pyplot as plt
# Declaración de constantes C1 y C2
c1 = 0.02
c2 = 987
# Declaración de la matriz que contiene el número de sensores para cada experimento de 5 minutos.
nbvg = np.arange(25,201,25)
# Estimación del consumo de energía mediante el modelo del estado del arte
E = c1*nbvg + c2
plt.plot(nbvg,E,'-o',color = '#F39C12',label = 'Estimaciones del modelo del estado del arte')
plt.xlabel("Número de sensores dentro del grupo $nb_{V_{G_{0}}}$")
plt.ylabel("Energía [J]")
plt.ylim([970, 1015]);plt.grid();plt.legend();plt.show()
```

La ejecución del script 8 produce la gráfica 5.7, en esta se puede observar que las estimaciones son idénticas a las presentadas en la gráfica 5.6.

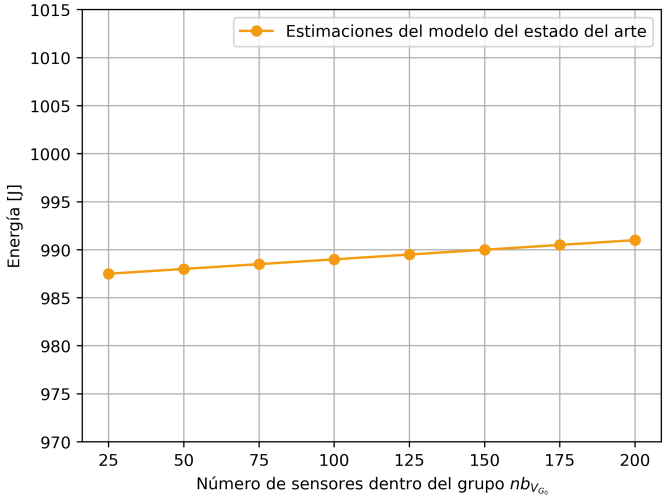

Figura 5.7: Replicación de estimaciones usando el modelo del estado del arte

También se pueden comparar los valores numéricos obtenidos con los valores presentados en el estudio del estado del arte (tabla 5.3). Cabe mencionar que solo es posible replicar las estimaciones del modelo del estado del arte, debido a que no se cuenta con la habitación con temperatura controlada ni el hardware usado en los experimentos descritos en el estudio del estado del arte, los cuales son necesarios para replicar las mediciones.

Listing 9 Valores numéricos de las estimaciones replicadas

```
1 E.tolist()
```

987.5 988.0 988.5 989.0 989.5 990.0 990.5 991.0

5.2.2. Calibración de las constantes del modelo del estado del arte

Anteriormente, se vio que para hacer estimaciones con el modelo del estado del arte se necesitan los valores numéricos de las constantes del modelo. Los autores del estudio usaron las mediciones del vatímetro para encontrar tales valores numéricos, en esta sección se usarán las mediciones tomadas usando el prototipo desarrollado para encontrar tales constantes. Además en el estudio del estado del arte se presentaron resultados para unas condiciones específicas; con propósitos de comparación, se compararán ambos modelos en el instante $t = 300[seg]$.

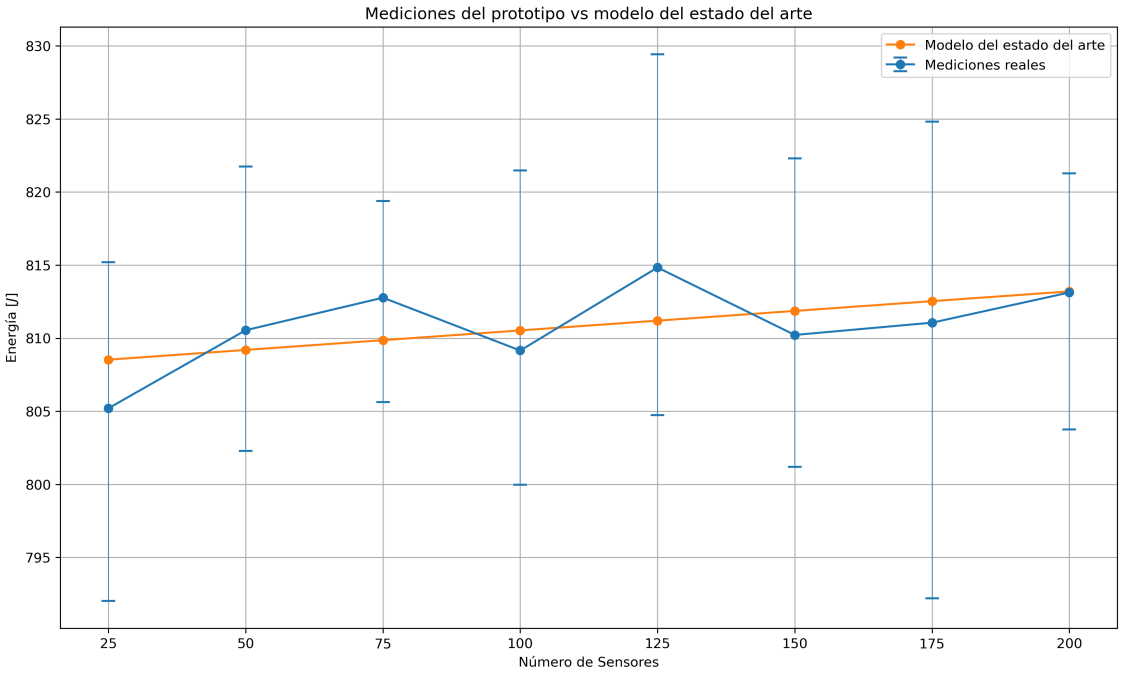

Figura 5.8: Calibración de las constantes del modelo del estado del arte.

Como se vio en la ecuación 5.3, para los casos de comparación el modelo del estado del arte toma la forma de una ecuación lineal, por lo que la calibración consistirá en encontrar los valores de C_1 y C_2. Debido a la simpleza del modelo el método de mínimos cuadrados será suficiente para encontrar tales constantes.

El proceso de calibración dio los valores para las constantes C_1 y C_2 (ecuación 5.3) que se muestran en la tabla 5.4.

Tabla 5.4: Valores numéricos de las contantes del modelo del estado del arte para el conjunto de mediciones hechas.

Constante	Valor
C_1	0.02669863510182939
C_2	807.8627964246551

Usando ambas contantes el modelo del estado del arte toma la siguiente forma para las mediciones recolectadas usando el prototipo.

$$E(nb_{V_{G_0}}, R_{G_i}) = \frac{0{,}02669863510182939 * nb_{V_{G_0}} + 807{,}8627964246551}{R_{G_i}} \tag{5.6}$$

5.2.3. Generalizando el modelo del estado del arte.

Ahora se procederá a comparar el modelo del estado del arte con el modelo desarrollado en esta tesis.

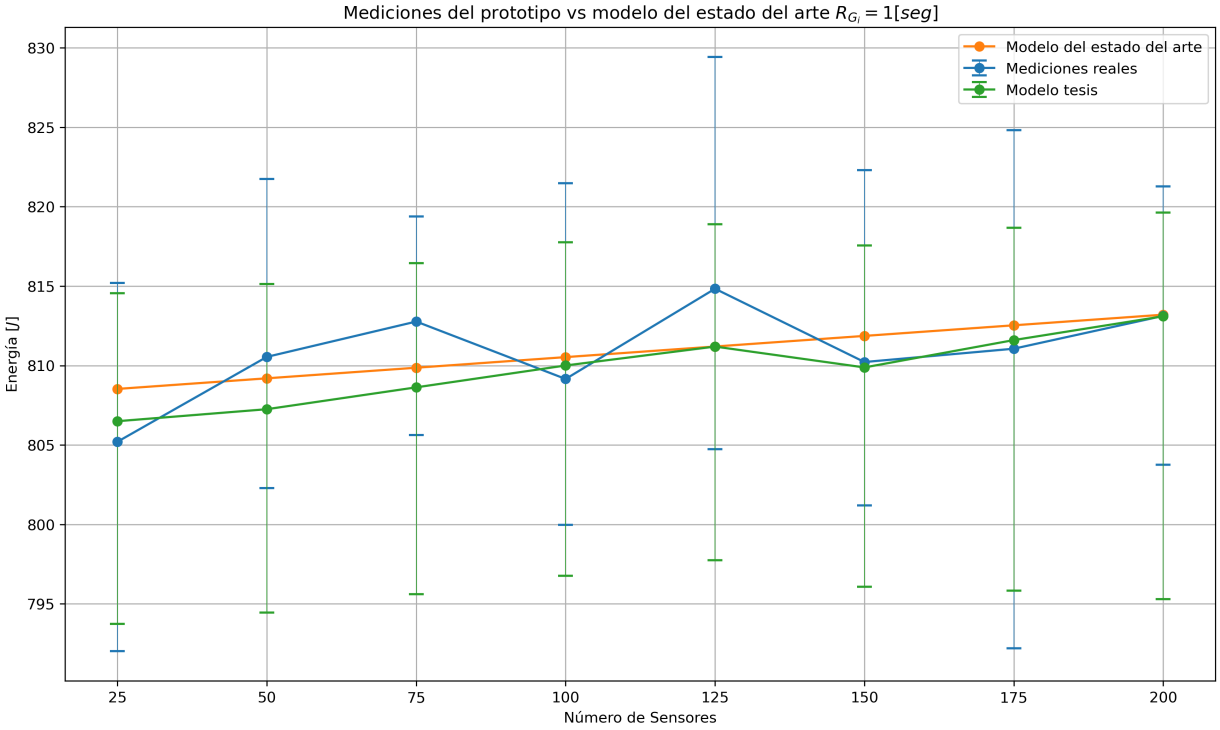

Figura 5.9: Mediciones del prototipo vs modelo del estado del arte vs modelo desarrollado ($R_{G_i} = 1[seg]$)

Con el fin de comparar de forma cuantitativa ambos modelos se calculó el error cuadrático medio entre las estimaciones de ambos modelos y las mediciones reales.

Tabla 5.5: Error cuadrático medio para el modelo del estado del arte y para el modelo desarrollado en esta tesis ($R_{G_i} = 1[seg]$).

Modelo	RMSE
Estado del arte (Borges et al. [10])	2.27289223970008
Tesis	2.346818068057912

La tabla 5.5 muestra que la diferencia entre el error cuadrático medio entre el modelo del estado del arte y el modelo desarrollado en esta tesis es de 0.07392582835783212. Esta diferencia es baja por lo que se puede afirmar que el modelo desarrollado es competitivo con respecto al estado del arte. Sin embargo, el modelo desarrollado en esta tesis muestra su valor cuando se usan periodos de actualización distintos a un segundo.

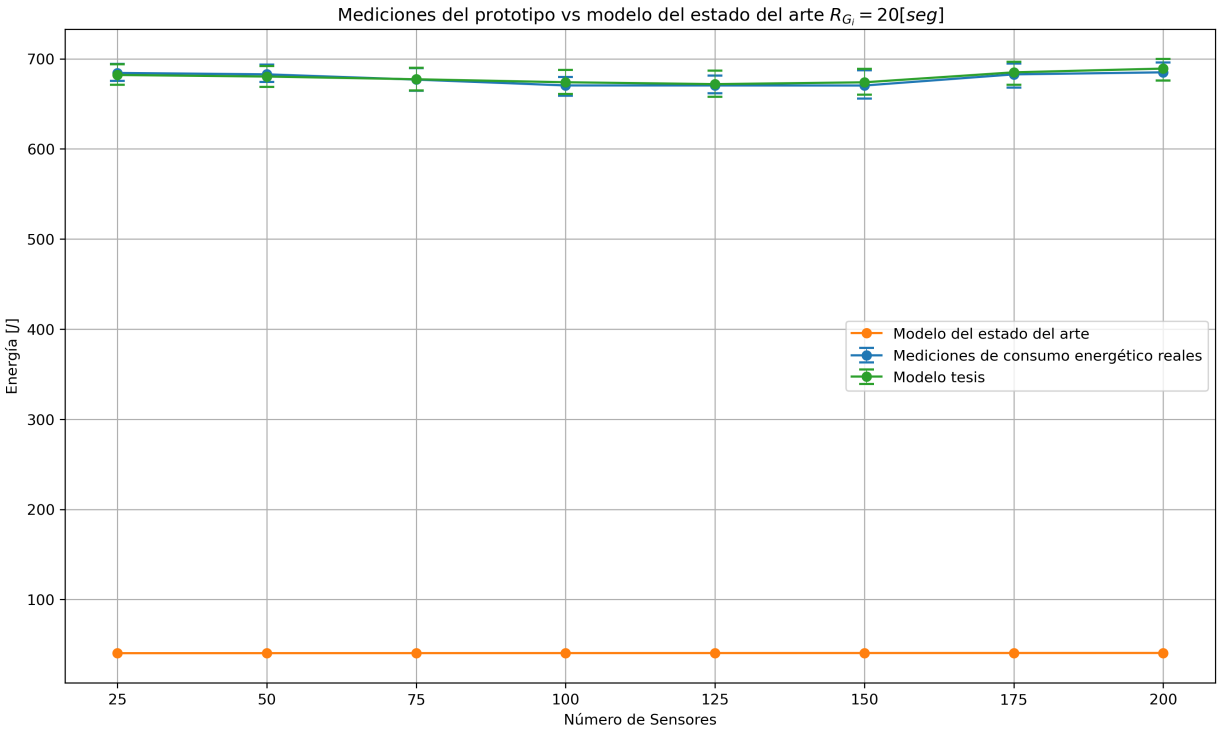

Figura 5.10: Mediciones de consumo energético reales vs modelo del estado del arte vs modelo desarrollado ($R_{G_i} = 20[seg]$).

Este último caso ($R_{G_i} = 20[seg]$) al igual que el caso ($R_{G_i} = 1[seg]$) se evaluó de forma cuantitativa usando el valor del error cuadrático medio como métrica.

Tabla 5.6: Error cuadrático medio para el modelo del estado del arte y para el modelo desarrollado en esta tesis ($R_{G_i} = 20[seg]$).

Modelo	RMSE
Estado del arte (Borges et al. [10])	637.3381020863869
Tesis	2.760910100099958

La tabla 5.6 muestra que al aumentar el periodo de actualización de los sensores, el error cuadrático medio de las estimaciones del modelo del estado del arte frente a las mediciones reales se eleva hasta 637.33. Este valor es considerablemente más alto en comparación con el error previo de 2.27, obtenido cuando el periodo de actualización era de un segundo. En cambio, el error cuadrático medio de las estimaciones del modelo desarrollado en esta tesis se mantiene cercano al valor registrado con un segundo de actualización. Esto sugiere que el modelo propuesto en esta tesis generaliza mejor para diferentes periodos de actualización.

La razón detrás de este alto error cuadrático medio es la forma del modelo del estado del arte (ecuación 5.6). Una vez este modelo se calibra lo único que modifica su salida frente a periodos de

actualización diferentes es una división siendo está incapaz de capturar la dinámica del consumo energético que sucede en la vida real.

Finalmente, el desarrollo de este modelo energético termina de manera satisfactoria habiéndose comparado con el método del estado del arte y habiéndolo expandido, abre las puertas a un desarrollo de una capa de energía del middleware IoTVar que pueda tomar estrategias de eficiencia energética en tiempo real.

Conclusiones y Recomendaciones

Conclusiones

1. Se implementó un sistema de estimación de consumo energético para el middleware IoTVar en una Raspberry Pi 4B, junto con un vatímetro especializado que permitió recopilar datos precisos de potencia instantánea.

2. El middleware IoTVar fue implementado exitosamente a un entorno Linux embebido, validando su capacidad para operar en plataformas con recursos limitados.

3. Se midió con precisión el consumo energético del sistema mediante un prototipo detallado en la tesis, lo que permitió evaluar el comportamiento bajo distintas condiciones operativas.

4. Se recopiló un dataset de consumo energético y variables del middleware, que fue clave para modelar y analizar el consumo energético de la Raspberry pi 4b.

5. Se propuso un modelo basado en redes neuronales para estimar el consumo energético, capturando relaciones no lineales y mejorando la precisión de las predicciones.

6. El modelo fue evaluado en una Raspberry Pi 4B y comparado con el método del estado del arte, obteniendo RMSE similar para $R_{G_i} = 1$ (2.35 vs 2.27) y mucho menor para $R_{G_i} = 20$ (2.76 vs 637.34); un RMSE menor indica mejor ajuste, evidenciando mayor generalización.

Recomendaciones

1. Se recomienda mejorar la calibración del vatímetro y explorar la compatibilidad con otros middleware y plataformas de hardware para incrementar la precisión.

2. Es recomendable evaluar la adaptabilidad del middleware en distintos entornos embebidos y desarrollar módulos que faciliten la integración con otras plataformas IoT.

3. Se sugiere explorar sensores de alta precisión y sistemas de adquisición avanzados, y ampliar la validación del prototipo en diferentes sistemas embebidos.

4. Es aconsejable automatizar el etiquetado y sincronización de datos, y ampliar el dataset para incluir más escenarios operativos y condiciones de prueba.

5. Es recomendable refinar el modelo con diferentes arquitecturas de redes neuronales y combinarlo con otros métodos estadísticos para mejorar la robustez.

Bibliografía

[1] P. V. Borges, C. Taconet, S. Chabridon, D. Conan, T. Batista, E. Cavalcante, and C. Batista, "Mastering interactions with internet of things platforms through the IoTVar middleware," in *13th International Conference on Ubiquitous Computing and Ambient Intelligence UCAmI 2019*. MDPI, p. 78. [Online]. Available: https://www.mdpi.com/2504-3900/31/1/78 IX, 11

[2] O. Uviase and G. Kotonya, "IoT architectural framework: Connection and integration framework for IoT systems," *Electronic Proceedings in Theoretical Computer Science*, vol. 264, pp. 1–17, 2018, [Online; accessed 27-Sep-2023]. [Online]. Available: http://arxiv.org/abs/1803.04780v1 IX, 13

[3] J. Flinn and M. Satyanarayanan, "PowerScope: a tool for profiling the energy usage of mobile applications," in *Proceedings WMCSA'99. Second IEEE Workshop on Mobile Computing Systems and Applications*. IEEE, pp. 2–10. [Online]. Available: http://ieeexplore.ieee.org/document/749272/ IX, 20

[4] T. Instruments, "INA219 zerø-drift, bi-directional current/power monitor with i2c interface." [Online]. Available: https://www.ti.com/lit/ds/symlink/ina219.pdf IX, XIII, 21, 31, 39, 40

[5] B. Carter and R. Mancini, *Op amps for everyone*, 5th ed. Oxford, United Kingdom Cambridge, MA: Newnes. IX, 22, 23, 32, 34

[6] I. Takeuchi, Q. V. Le, T. D. Sears, and A. J. Smola, "Nonparametric quantile regression," *Journal of Machine Learning Research*, vol. 7, pp. 1231–1264, 2006. IX, 26, 27

[7] J. Lambert, R. Monahan, and K. Casey, "Power consumption profiling of a lightweight development board: Sensing with the INA219 and teensy 4.0 microcontroller," *Electronics*,

vol. 10, no. 7, p. 775, 2021. [Online]. Available: https://www.mdpi.com/2079-9292/10/7/775 IX, 29, 30, 31, 39, 43

[8] R. P. Foundation, *Raspberry Pi Pico Schematic*, 2021. [Online]. Available: https://proto-pic.co.uk/content/RPI-PICO-R3-PUBLIC-SCHEMATIC.pdf IX, 33

[9] R. P. Ltd, "RP2040 datasheet," 2024. [Online]. Available: https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf X, 37, 38

[10] P. V. Borges, C. Taconet, S. Chabridon, D. Conan, and E. Cavalcante, "A middleware architecture for mastering energy consumption in internet of things applications," in *Proceedings of the 5th International Conference on ICT for Sustainability (ICT4S)*, Rennes, France, 2023, pp. 66–75. X, XI, XIII, 3, 5, 15, 49, 60, 81, 82, 83, 88, 89

[11] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Anchorage, AK, USA, 2019, pp. 2623–2631. [Online]. Available: http://arxiv.org/abs/1907.10902 XIII, 73, 74

[12] L. Cozzi, T. Gould, T. Gül, K. McNamara, C. McGlade, J. Trüby, and E. Bellevrat, "World energy outlook 2017," Paris, p. 782, 2017. [Online]. Available: https://www.iea.org/reports/world-energy-outlook-2017 2

[13] C. Quillama, "Operación de la subestación de quencoro cusco ante el crecimiento de la demanda en el periodo 2018 al 2024," Ph.D. dissertation, Universidad Nacional de San Antonio Abad del Cusco, Cusco, 2022. [Online]. Available: https://repositorio.unsaac.edu.pe/bitstream/handle/20.500.12918/6581/253T20220153_TC.pdf?sequence=1 2

[14] O. Alšauskas, L. A. Sarazola, Y. Arsalane, B. Barreau, and S. Bennett, "World energy outlook 2022," International Energy Agency (IEA), Paris, Tech. Rep., 2022. [Online]. Available: https://www.iea.org/reports/world-energy-outlook-2022 2

[15] B. C. de Reserva del Perú, "Reporte de inflación: Panorama actual y proyecciones macroeconómicas 2021–2022," Banco Central de Reserva del Perú, Lima, Tech. Rep. No.

2005-6985, Jun. 2021. [Online]. Available: https://www.bcrp.gob.pe/docs/Publicaciones/Reporte-Inflacion/2021/junio/reporte-de-inflacion-junio-2021.pdf 2

[16] T. Okrasinski, T. Fleming, A. Moore, G. Gines, S. Kelly, S. Moore, and M. Shackleton, "Smarter2030: Ict solutions for 21st century challenges," Global eSustainability Initiative (GeSI), Brussels, Belgium, Tech. Rep., Oct. 2015. [Online]. Available: https://smarter2030.gesi.org/downloads/Full_report.pdf 2

[17] T. M. Attia, "Challenges and opportunities in the future applications of IoT technology," in *2nd Europe–Middle East–North Africa Regional Conference of the International Telecommunications Society (ITS)*, Aswan, Egypt, Feb. 2019, pp. 1–15. [Online]. Available: https://hdl.handle.net/10419/201752 2

[18] M. Friedli, L. Kaufmann, F. Paganini, and R. Kyburz, "Energy efficiency of the internet of things," Lucerne University of Applied Sciences (prepared for IEA 4E EDNA), Lucerne, Switzerland, Tech. Rep., Apr. 2016. [Online]. Available: https://www.iea-4e.org/wp-content/uploads/publications/2016/04/Energy_Efficiency_of_the_Internet_of_Things_-_Technical_Report_FINAL.pdf 2

[19] F. A. Almalki, S. H. Alsamhi, R. Sahal, J. Hassan, A. Hawbani, N. S. Rajput, A. Saif, J. Morgan, and J. Breslin, "Green iot for eco-friendly and sustainable smart cities: Future directions and opportunities," *Mobile Networks and Applications*, vol. 28, no. 1, pp. 178–202, Feb. 2023. [Online]. Available: https://link.springer.com/10.1007/s11036-021-01790-w 2

[20] W. Mao, Z. Zhao, Z. Chang, G. Min, and W. Gao, "Energy-efficient industrial internet of things: Overview and open issues," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 11, pp. 7225–7237, Nov. 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9381665/ 2

[21] C. K. Metallidou, K. E. Psannis, and E. A. Egyptiadou, "Energy efficiency in smart buildings: IoT approaches," *IEEE Access*, vol. 8, pp. 63 679–63 699, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9050775/ 2

[22] S. Pasricha, R. Ayoub, M. Kishinevsky, S. K. Mandal, and U. Y. Ogras, "A survey on energy management for mobile and IoT devices," *IEEE Design & Test*, vol. 37, no. 5, pp. 7–24, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9017997/ 3

[23] A. Noureddine, R. Rouvoy, and L. Seinturier, "A review of energy measurement approaches," *ACM SIGOPS Operating Systems Review*, vol. 47, no. 3, pp. 42–49, 2013. [Online]. Available: https://dl.acm.org/doi/10.1145/2553070.2553077 3

[24] P. V. B. C. D. Silva, C. Taconet, S. Chabridon, D. Conan, E. Cavalcante, and T. Batista, "Energy awareness and energy efficiency in internet of things middleware: a systematic literature review," vol. 78, no. 1, pp. 115–131. [Online]. Available: https://link.springer.com/10.1007/s12243-022-00936-5 3

[25] S. A. Sarswatula, T. Pugh, and V. Prabhu, "Modeling energy consumption using machine learning," *Frontiers in Manufacturing Technology*, vol. 2, p. 855208, 2022. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fmtec.2022.855208/full 3

[26] N. N. Alajlan and D. M. Ibrahim, "Tinyml: Enabling of inference deep learning models on ultra-low-power iot edge devices for ai applications," *Micromachines*, vol. 13, no. 6, p. 851, 2022. [Online]. Available: https://www.mdpi.com/2072-666X/13/6/851 3

[27] Z. Meng, H. Sun, and X. Wang, "Forecasting energy consumption based on svr and markov model: A case study of china," *Frontiers in Environmental Science*, vol. 10, p. 883711, 2022. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fenvs.2022.883711/full 4

[28] A. Sabovic, C. Delgado, J. Bauwens, E. D. Poorter, and J. Famaey, "Accurate online energy consumption estimation of iot devices using energest," in *Proceedings of the 28th International Conference on Computer Communications and Networks (ICCCN)*. Springer, 2019, pp. 373–383. [Online]. Available: https://doi.org/10.1007/978-3-030-33506-9_32 4

[29] K. Kesrouani, H. Kanso, and A. Noureddine, "A preliminary study of the energy impact of software in raspberry pi devices," in *2020 IEEE 29th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*. IEEE, pp. 231–234. [Online]. Available: https://ieeexplore.ieee.org/document/9338608/ 4, 81

[30] S. Bhattacharya, P. Kumar, I. Meenakshisundaram, T. Gadekallu, S. Sharma, M. Alkahtani, and M. Abidi, "Deep neural networks based approach for battery life prediction," vol. 69, no. 2, pp. 2599–2615, publisher: Tech Science Press. [Online]. Available: https://www.techscience.com/cmc/v69n2/43846 4

[31] H. Kanso, A. Noureddine, and E. Exposito, "Automated power modeling of computing devices: Implementation and use case for raspberry pis," *Sustainable Computing: Informatics and Systems*, vol. 37, p. 100837, 2023. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S2210537922001688 5, 81

[32] P. V. Borges, C. Taconet, S. Chabridon, D. Conan, E. Cavalcante, and T. Batista, "Taming internet of things application development with the IoTvar middleware," *ACM Transactions on Internet Technology*, vol. 23, no. 2, pp. 1–21, 2023. [Online]. Available: https://dl.acm.org/doi/10.1145/3586010 5, 15

[33] J. Sengupta, S. Ruj, and S. D. Bit, "A comprehensive survey on attacks, security issues and blockchain solutions for iot and iiot," *Journal of Network and Computer Applications*, vol. 149, p. 102481, 2020. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1084804519303418 9

[34] S. Siboni, V. Sachidananda, Y. Meidan, M. Bohadana, Y. Mathov, S. Bhairav, A. Shabtai, and Y. Elovici, "Security testbed for internet-of-things devices," *IEEE Transactions on Reliability*, vol. 68, no. 1, pp. 23–44, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8565917 9

[35] J. H. Nord, A. Koohang, and J. Paliszkiewicz, "The internet of things: Review and theoretical framework," *Expert Systems with Applications*, vol. 133, pp. 97–108, 2019. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0957417419303331 9

[36] S. Luthra, D. Garg, S. K. Mangla, and Y. P. S. Berwal, "Analyzing challenges to internet of things (IoT) adoption and diffusion: An indian context," *Procedia Computer Science*, vol. 125, pp. 733–739, 2018. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1877050917328624 9

[37] M. Asplund and S. Nadjm-Tehrani, "Attitudes and perceptions of IoT security in critical societal services," *IEEE Access*, vol. 4, pp. 2130–2138, 2016. [Online]. Available: https://ieeexplore.ieee.org/document/7463021/ 10

[38] L. Chen, S. Thombre, K. Jarvinen, E. S. Lohan, A. Alen-Savikko, H. Leppakoski, M. Z. H. Bhuiyan, S. Bu-Pasha, G. N. Ferrara, S. Honkala, J. Lindqvist, L. Ruotsalainen, P. Korpisaari, and H. Kuusniemi, "Robustness, security and privacy in location-based services for future IoT: A survey," *IEEE Access*, vol. 5, pp. 8956–8977, 2017. [Online]. Available: http://ieeexplore.ieee.org/document/7903611/ 10

[39] H. M. Al-Kadhim and H. S. Al-Raweshidy, "Energy efficient and reliable transport of data in cloud-based IoT," *IEEE Access*, vol. 7, pp. 64 641–64 650, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8716653/ 10

[40] G. Blair, D. Schmidt, and C. Taconet, "Middleware for internet distribution in the context of cloud computing and the internet of things: Editorial introduction," *Annals of Telecommunications*, vol. 71, no. 3, pp. 87–92, Apr. 2016. [Online]. Available: http://link.springer.com/10.1007/s12243-016-0493-z 12

[41] Q. H. Mahmoud, *Middleware for Communications*, 1st ed. Chichester, UK: John Wiley & Sons, 2004. [Online]. Available: https://onlinelibrary.wiley.com/doi/book/10.1002/0470862084 12

[42] A. T. Campbell, G. Coulson, and M. E. Kounavis, "Managing complexity: Middleware explained," *IT Professional*, vol. 1, no. 5, pp. 22–28, oct 1999. [Online]. Available: http://ieeexplore.ieee.org/document/793667/ 12

[43] I. Lee and K. Lee, "The internet of things (IoT): Applications, investments, and challenges for enterprises," *Business Horizons*, vol. 58, no. 4, pp. 431–440, jul 2015. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0007681315000373 12

[44] A. H. H. Ngu, M. Gutierrez, V. Metsis, S. Nepal, and M. Z. Sheng, "IoT middleware: A survey on issues and enabling technologies," *IEEE Internet of Things Journal*, vol. 4, no. 1, pp. 1–20, feb 2017. [Online]. Available: http://ieeexplore.ieee.org/document/7582463/ 12

[45] G. Marques, N. Garcia, and N. Pombo, "A survey on IoT: Architectures, elements, applications, QoS, platforms and security concepts," in *Advances in Mobile Cloud Computing and Big Data in the 5G Era*, C. X. Mavromoustakis, G. Mastorakis, and C. Dobre, Eds. Springer International Publishing, vol. 22, pp. 115–130, series Title: Studies in Big Data. [Online]. Available: http://link.springer.com/10.1007/978-3-319-45145-9_5 12

[46] S. M. A. Razzaque, I. B. M. M., A. Alhammadi, and A. V. Vasilakos, "Middleware for internet of things: A survey," in *IEEE Internet of Things Journal*, vol. 3, no. 1. IEEE, 2016, pp. 70–95. [Online]. Available: https://ieeexplore.ieee.org/document/7359442 12, 13

[47] B. Nakhuva and T. Champaneria, "Study of various internet of things platforms," *International Journal of Computer Science & Engineering Survey*, vol. 6, no. 6, pp. 61–74, 2015. [Online]. Available: http://www.aircconline.com/ijcses/V6N6/6615ijcses05.pdf 13

[48] K. Yaghmour, *Building Embedded Linux Systems*. O'Reilly Media, 2008. 16

[49] R. Love, *Linux Kernel Development*, 3rd ed. Addison-Wesley Professional, 2010. 16

[50] J. Corbet, A. Rubini, and G. Kroah-Hartman, *Linux Device Drivers*, 3rd ed. O'Reilly Media, 2005. 16

[51] C. Hallinan, *Embedded Linux Primer: A Practical, Real-World Approach*, ser. Prentice Hall Open Source Software Development Series. Prentice Hall, 2007, OCLC: ocm67361321. 16

[52] R. Streif, *Embedded Linux systems with the Yocto Project*, ser. Prentice Hall open source software development series. Prentice Hall. 17

[53] B. Dezfouli, I. Amirtharaj, and C.-C. C. Li, "EMPIOT: An energy measurement platform for wireless IoT devices," *Journal of Network and Computer Applications*, vol. 121, pp. 135–148, Nov. 2018. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1084804518302479 18

[54] S. Hill, D. Hudgins, K. Eckles, M. Toa, and Z. Tang, *An Engineer's Guide to Current Sensing*. Texas Instruments Incorporated, 2023. [Online]. Available: https://rb.gy/7e9bkj 18

[55] T. P. Trappenberg, *Fundamentals of Machine Learning*, 1st ed. Oxford University Press, 2019. [Online]. Available: https://academic.oup.com/book/36775 25

[56] J. Roeder, S. Altmeyer, and C. Grelck, "Can we trust our energy measurements? a study on the odroid-xu4," in *Proceedings of the 14th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT '22)*, Jun. 2022. [Online]. Available: https://arxiv.org/abs/2206.10377 30, 47

[57] Texas Instruments, "LMx24, LMx24x, LMx24xx, LM2902, LM2902x, LM2902xx, LM2902xxx quadruple operational amplifiers," p. 62, Mar. 2024. [Online]. Available: https://acortar.link/CzAmFv 33

[58] Raspberry Pi Foundation, "Raspberry pi 15w USB-c power supply," 2024. [Online]. Available: https://datasheets.raspberrypi.com/power-supply/15w-usb-c-power-supply-product-brief.pdf 55, 56

[59] H. Quispe, Rodolfo y Pedrini, "Top-db-net: Top dropblock for activation enhancement in person re-identification," in *2020 25th International conference on pattern recognition (ICPR)*, no. arXiv:2010.05435, IEEE. arXiv, 2021. [Online]. Available: http://arxiv.org/abs/2010.05435 75

Apéndice A

Firmware del Prototipo

A.1. Script principal

```
#include "hardware/adc.h"
#include "hardware/dma.h"
#include "hardware/gpio.h"
#include "hardware/i2c.h"
#include "hardware/timer.h"
#include "ina219.h"
#include "pico/stdlib.h"
#include "math.h"
#include "stdio.h"

#define SERIAL_BAUD 2560000
#define SDA_LINE 16
#define SCL_LINE 17

#define ADC_INPUT_CH 0
#define ADC_INPUT_PIN 26
#define OFFSET_VOLT 4.36
#define AMP_GAIN 2.99
#define SUBS_GAIN 1.107
#define DIVIDER_GAIN 0.31577256020406114

#define SLOPE_M 9.33613445
#define OFFSET_B 13.25273109

#define STATE_HIGH 1
```

```
26  #define STATE_LOW 0
27
28  i2c_inst_t *i2c_if = i2c0;
29  ina219 powerSensor(0x40, i2c_if);
30
31  uint16_t adc_raw_value;
32  const float adc_to_volt_factor = 0.012603482716692722;
33
34  #define ADC_CLK_DIV 1
35  #define SAMPLE_COUNT 128
36  dma_channel_config dma_cfg;
37  uint dma_channel_id;
38  float freq_buffer[SAMPLE_COUNT];
39
40  void adc_dma_capture(uint16_t *buffer) {
41      adc_fifo_drain();
42      adc_run(false);
43      dma_channel_configure(dma_channel_id, &dma_cfg,
44                            buffer,
45                            &adc_hw->fifo,
46                            SAMPLE_COUNT,
47                            true
48      );
49      adc_run(true);
50      dma_channel_wait_for_finish_blocking(dma_channel_id);
51  }
52
53  void setup_adc_dma() {
54      adc_gpio_init(ADC_INPUT_PIN);
55      adc_init();
56      adc_select_input(ADC_INPUT_CH);
57      gpio_set_dir(23, GPIO_OUT);
58      gpio_put(23, STATE_HIGH);
59      adc_fifo_setup(
60          true,
61          true,
62          1,
63          false,
64          true
65      );
66      adc_set_clkdiv(ADC_CLK_DIV);
```

```
67      sleep_ms(1000);
68      dma_channel_id = dma_claim_unused_channel(true);
69      dma_cfg = dma_channel_get_default_config(dma_channel_id);
70      channel_config_set_transfer_data_size(&dma_cfg, DMA_SIZE_16);
71      channel_config_set_read_increment(&dma_cfg, false);
72      channel_config_set_write_increment(&dma_cfg, true);
73      channel_config_set_dreq(&dma_cfg, DREQ_ADC);
74  }
75
76  void setup_ina219() {
77      i2c_init(i2c_if, SERIAL_BAUD);
78      gpio_set_function(SDA_LINE, GPIO_FUNC_I2C);
79      gpio_set_function(SCL_LINE, GPIO_FUNC_I2C);
80      gpio_pull_up(SDA_LINE);
81      gpio_pull_up(SCL_LINE);
82      powerSensor.setCalibration(true);
83  }
84
85  float adc_to_bus_voltage(double adc_val) {
86      float voltage_adc = adc_val * adc_to_volt_factor;
87      return (voltage_adc + OFFSET_B) / (SLOPE_M * DIVIDER_GAIN);
88  }
89
90  bool timer_measurement_callback(struct repeating_timer *t) {
91      uint64_t start_time = time_us_64();
92      uint16_t capture_buf[SAMPLE_COUNT];
93      powerSensor.triggerMeasurement();
94      uint64_t time_after_sensor = time_us_64();
95      adc_dma_capture(capture_buf);
96      double avg_adc = 0.0;
97      for (int i = 0; i < SAMPLE_COUNT; i++) {
98          avg_adc += capture_buf[i];
99      }
100     avg_adc = round(avg_adc / SAMPLE_COUNT);
101     uint64_t time_after_adc = time_us_64();
102     float current_mA = powerSensor.getCurrent_mA();
103     float bus_voltage = adc_to_bus_voltage(avg_adc);
104     float power_W = current_mA * bus_voltage;
105     float sensor_time = time_after_sensor - start_time;
106     float adc_time = time_after_adc - time_after_sensor;
107     printf("%lld,%.1f,%.4f,%.4f\n", time_after_sensor, current_mA, bus_voltage, power_W);
```

```
108     return true;
109 }
110 
111 int main() {
112     stdio_init_all();
113     setup_ina219();
114     setup_adc_dma();
115     struct repeating_timer timer_instance;
116     add_repeating_timer_us(-3000, timer_measurement_callback, NULL, &timer_instance);
117     while (true) {}
118     return 0;
119 }
```

A.2. Libreria INA219

A.2.1. Header

```
1  #ifndef INA219_H_
2  #define INA219_H_
3  
4  #include "hardware/i2c.h"
5  #include "pico/stdlib.h"
6  
7  #define INA219_ADDRESS 0x40
8  #define INA219_REG_BUSVOLTAGE 0x02
9  #define INA219_REG_SHUNTVOLTAGE 0x01
10 #define INA219_REG_CURRENT 0x04
11 #define INA219_REG_POWER 0x03
12 #define INA219_REG_CALIBRATION 0x05
13 #define INA219_REG_CONFIG 0x00
14 
15 #define I2C_CHAN i2c0
16 
17 class ina219 {
18 public:
19   ina219(uint8_t addr = INA219_ADDRESS, i2c_inst_t *i2c = I2C_CHAN);
20 
21   ~ina219();
22 
23   void setCalibration(bool triggered = false);
```

```
24    int16_t getBusVoltage_raw();
25    int16_t getShuntVoltage_raw();
26    int16_t getCurrent_raw();
27    int16_t getPower_raw();
28    float getShuntVoltage_mV();
29    float getBusVoltage_V();
30    float getCurrent_mA();
31    float getPower_mW();
32    void triggerMeasurement();
33    uint64_t triggerMeasurement(uint64_t call_timestamp);
34    void wireWriteRegister(uint8_t reg, uint16_t value);
35    uint16_t wireReadRegister(uint8_t reg);
36
37  private:
38    uint8_t ina219_i2caddr;
39    float ina219_currentDivider_mA;
40    float ina219_powerMultiplier_mW;
41    float ina219_shuntVoltageMultiplier_mV;
42    float ina219_busVoltageMultiplier_V;
43    i2c_inst_t *ina219_i2cChan;
44  };
45
46  #endif
```

A.2.2. Implementación

```
1   #include "ina219.h"
2
3   #include "hardware/i2c.h"
4   #include "hardware/timer.h"
5   #include "pico/stdlib.h"
6   #include "stdio.h"
7
8   ina219::ina219(uint8_t addr, i2c_inst_t *i2c) {
9     ina219_i2caddr = addr;
10    ina219_i2cChan = i2c;
11    ina219_currentDivider_mA = 0;
12    ina219_powerMultiplier_mW = 0;
13    ina219_shuntVoltageMultiplier_mV = 0;
14    ina219_busVoltageMultiplier_V = 0;
15  }
```

```
16  ina219::~ina219() {}
17
18  void ina219::setCalibration(bool triggered) {
19
20    // Calibration value
21    // uint16_t ina219_calValue = 10240;
22    uint16_t ina219_calValue = 8192;
23    wireWriteRegister(INA219_REG_CALIBRATION, ina219_calValue);
24
25    // Configuration register
26    if (triggered) {
27      // uint16_t config = 0x0000 | 0x1000 | 0x0480 | 0x0070 | 0x0001; // 25Hz
28      //  uint16_t config = 0x0000 | 0x1000 | 0x0480 | 0x0050 | 0x0001; // 250Hz
29      //  uint16_t config = 0x0000 | 0x1000 | 0x0480 | 0x0068 | 0x0001; // 50Hz
30      uint16_t config = 0x0000 | 0x1000 | 0x0480 | 0x0050 | 0x0001; // 300Hz
31      // uint16_t config = 0x0000 | 0x1000 | 0x0480 | 0x0048 | 0x0001; // 500Hz
32      // uint16_t config = 0x0000 | 0x1000 | 0x0480 | 0x0018 | 0x0001; // 1kHz
33      // uint16_t config = 0x0000 | 0x1000 | 0x0480 | 0x0060 | 0x0001; // 100Hz
34      // uint16_t config = 0x0000 | 0x1000 | 0x0780 | 0x0058 | 0x0001; //200Hz
35      // uint16_t config = 0x0000 | 0x0000 | 0x0780 | 0x0058 | 0x0001;
36      wireWriteRegister(INA219_REG_CONFIG, config);
37
38    } else {
39      uint16_t config = 0x2000 | 0x1000 | 0x0780 | 0x01F8 | 0x0007;
40      wireWriteRegister(INA219_REG_CONFIG, config);
41    }
42
43    i2c_write_blocking(ina219_i2cChan, INA219_ADDRESS << 1, NULL, 0,
44                       true); // Start condition
45
46    // Write high-speed master code (00001XXX)
47    i2c_write_blocking(ina219_i2cChan, 0b00001000, NULL, 0, true);
48
49    // Repeated start condition
50    i2c_write_blocking(ina219_i2cChan, INA219_ADDRESS << 1, NULL, 0,
51                       true); // Repeated start
52
53    // Multipliers
54    // ina219_currentDivider_mA = 25;
55    ina219_currentDivider_mA = 20;
56    ina219_powerMultiplier_mW = 0.8f;
```

```
57    ina219_shuntVoltageMultiplier_mV = 0.01;
58    ina219_busVoltageMultiplier_V = 0.003986;
59  }
60
61  int16_t ina219::getBusVoltage_raw() {
62    return wireReadRegister(INA219_REG_BUSVOLTAGE) >> 3;
63  }
64
65  int16_t ina219::getShuntVoltage_raw() {
66    return wireReadRegister(INA219_REG_SHUNTVOLTAGE);
67  }
68
69  int16_t ina219::getCurrent_raw() {
70    return wireReadRegister(INA219_REG_CURRENT);
71  }
72
73  int16_t ina219::getPower_raw() { return wireReadRegister(INA219_REG_POWER); }
74
75  float ina219::getShuntVoltage_mV() {
76    return getShuntVoltage_raw() * ina219_shuntVoltageMultiplier_mV;
77  }
78
79  float ina219::getBusVoltage_V() {
80    return getBusVoltage_raw() * ina219_busVoltageMultiplier_V;
81  }
82
83  float ina219::getCurrent_mA() {
84    return getCurrent_raw() / ina219_currentDivider_mA;
85  }
86
87  float ina219::getPower_mW() {
88    return getPower_raw() * ina219_powerMultiplier_mW;
89  }
90
91  void ina219::triggerMeasurement() {
92    uint16_t val = wireReadRegister(INA219_REG_CONFIG);
93    wireWriteRegister(INA219_REG_CONFIG, val);
94    uint16_t convReady = 0x0000;
95    while (!convReady) {
96      convReady = ((wireReadRegister(INA219_REG_BUSVOLTAGE)) &
97                   0x0002); // checks if sampling is completed
```

```
98    }
99  }
100
101 uint64_t ina219::triggerMeasurement(uint64_t call_timestamp) {
102   uint16_t val = wireReadRegister(INA219_REG_CONFIG);
103   wireWriteRegister(INA219_REG_CONFIG, val);
104   uint64_t convertion_timestamp = time_us_64();
105   uint16_t convReady = 0x0000;
106   while (!convReady) {
107     convReady = ((wireReadRegister(INA219_REG_BUSVOLTAGE)) &
108                  0x0002); // checks if sampling is completed
109   }
110   return convertion_timestamp;
111 }
112
113 void ina219::wireWriteRegister(uint8_t reg, uint16_t value) {
114   uint8_t data[] = {reg, (value >> 8) & 0xFF, value & 0xFF};
115   i2c_write_blocking(ina219_i2cChan, ina219_i2caddr, data, sizeof(data), false);
116 }
117
118 uint16_t ina219::wireReadRegister(uint8_t reg) {
119   uint16_t value = 0;
120   i2c_write_blocking(ina219_i2cChan, ina219_i2caddr, &reg, 1, true);
121   i2c_read_blocking(ina219_i2cChan, ina219_i2caddr,
122                     reinterpret_cast<uint8_t *>(&value), sizeof(value), false);
123   return (value >> 8) | ((value & 0xFF) << 8); // Swap bytes for little-endian
124 }
```

A.3. Configuración CMake

A.3.1. Configuración del directorio

```
1  add_executable(${NAME}
2          main.cpp
3          ina219.cpp
4          )
5
6  target_link_libraries(${NAME}
7          hardware_i2c
8          hardware_timer
```

```
        hardware_adc
        hardware_dma
        hardware_pio
        pico_time
        pico_stdlib
        )
target_include_directories(${NAME} PRIVATE
        ${CMAKE_CURRENT_LIST_DIR}
        )

pico_add_extra_outputs(${NAME})

# enable usb output, disable uart output
pico_enable_stdio_usb(${NAME} 1)
pico_enable_stdio_uart(${NAME} 0)
```

A.3.2. Configuración del proyecto

```
cmake_minimum_required(VERSION 3.12)
set(NAME RPiPico_wattmeter)

set(CMAKE_EXPORT_COMPILE_COMMANDS ON)
include(pico_sdk_import.cmake)

project(${NAME} C CXX ASM)
set(CMAKE_C_STANDARD 11)
set(CMAKE_CXX_STANDARD 17)

pico_sdk_init()

add_subdirectory(src)
install(CODE "execute_process(COMMAND $ENV{HOME}/bin/picoDeploy.sh
↪  ${CMAKE_CURRENT_BINARY_DIR}/src/${NAME}.elf)")
install(FILES
    ${CMAKE_CURRENT_BINARY_DIR}/src/${NAME}.uf2
    DESTINATION ${CMAKE_CURRENT_BINARY_DIR}
)

set(CPACK_INCLUDE_TOPLEVEL_DIRECTORY OFF)
set(CPACK_GENERATOR "ZIP" "TGZ")
include(CPack)
```

Apéndice B

Script de Captura de Mediciones del Prototipo

B.1. Script principal

```
from classes.RPi_Pico_Wattmeter_serial import RPi_Pico_serial
from classes.RPi_ssh_agent import RaspberryPiSSH

from multiprocessing import Process


def take_measurements(experiment_name):
    recorder = RPi_Pico_serial()
    seconds = 90000000
    num_samples = seconds * 200
    save_path = "./data/new_data/"

    if recorder.connect_to_port():
        recorder.open_file(save_path, experiment_name)
        recorder.record_data(num_samples)
        recorder.close()
    else:
        print("Could not find RPi Pico")


def make_ssh_command(
    sensor_number="1",
```

```
    test_time="1",
    file_name="perf_fiware",
    platform_url="localhost",
    strategy="green",
    freshness_frequency="1",
    notification_url="1",
):
    exec_path = "cd 2023-iotvar-hardware/Code/Phase_1/iothandler/application"

    mvn_call = (
        "mvn exec:java -Dexec.mainClass=iotvarPerformance.fiware.IotvarFiwareExample "
        + '-Dexec.args="'
        + f"{sensor_number} {test_time} {file_name} {platform_url} {strategy} "
        + f'{freshness_frequency} {notification_url}"'
    )
    command = f"{exec_path} && {mvn_call}"
    return command


def get_startTimestamp(exec_finish_message):
    timestampIndex = exec_finish_message.find("StartTime:")
    Timestamp = exec_finish_message[timestampIndex + 11 : timestampIndex + 11 + 13]
    return Timestamp


if __name__ == "__main__":
    experiment_name = "dynamic_refresh_time_60seconds"
    freshness = "60"
    testTime = "300"

    power_measure = Process(target=take_measurements, args=(experiment_name,))
    # ssh
    ssh_client = RaspberryPiSSH()
    # ssh_client_iotvar_sender = RaspberryPiSSH()

    ssh_client.connect()

    log_file = open(
        "./data/new_data/" + experiment_name + "_experiment_metadata.csv", "w"
    )
    log_file.write("%s" % "sensorNumber,testTime,freshness,startTimestamp\n")
```

```
    power_measure.start()
    cpuLogname = experiment_name + "_CPU_usage.csv"
    cpuLogpath = (
        "/home/han4n/2023-iotvar-hardware/Code/Phase_3/IoTVar_PowerProfiler/extras/"
    )
    ssh_client.triggerCPUmeasurements(cpuLogname, cpuLogpath)

    for i in [25, 50, 75, 100, 125, 150, 175, 200]:
        for j in range(25):
            print("number of sensors: " + str(i) + " experiment: " + str(j + 1))
            command = make_ssh_command(
                sensor_number=str(i), test_time=testTime, freshness_frequency=freshness
            )
            # ssh_client_iotvar_sender.connect()
            ssh_client.execute_command(command)
            startTimestamp = get_startTimestamp(ssh_client.exec_finish_message)
            log_file.write(
                "%s" % str(i)
                + ","
                + testTime
                + ","
                + freshness
                + ","
                + startTimestamp
                + "\n"
            )
            # ssh_client_iotvar_sender.close()
    power_measure.terminate()
    log_file.close()
    ssh_client.finishCPUmeasurements()
    print("Finished experiments")
    print("Transfering CPU log file")

    scp = ssh_client.get_scp_agent()
    destination = "/home/han4n/2023-iotvar-hardware/Code/Phase_3/IoTVar_PowerProfiler/data/new_data"

    scp.get(cpuLogpath + cpuLogname, destination)

    ssh_client.close()
    print("Transfer complete")
```

B.2. Clase usada en la conexión con el prototipo

```
from serial.tools import list_ports
import serial
import time
import csv
from datetime import datetime, timedelta


class RPi_Pico_serial:
    def __init__(self, baud_rate=115200):
        self.baud_rate = baud_rate
        self.serial_port = None
        self.file = None

    def connect_to_port(self):
        pico_ports = list(list_ports.grep("2E8A"))
        if not pico_ports:
            print("No Raspberry Pi Pico found")
            return False
        else:
            pico_serial_port = pico_ports[0].device
            print("Raspberry Pi Pico found at {}".format(pico_serial_port))
            self.serial_port = serial.Serial(pico_serial_port, self.baud_rate)
            return True

    def open_file(self, path, experiment_name):
        current_timestamp = time.time()
        current_datetime_utc = datetime.utcfromtimestamp(current_timestamp)
        current_datetime_local = current_datetime_utc - timedelta(hours=5)
        name = current_datetime_local.strftime("%Y-%m-%d_%H-%M-%S")
        filename = path + name + "_" + experiment_name + ".csv"
        self.file = open(filename, "w", newline="")
        self.file.truncate()

    def record_data(self, num_samples):
        if not self.serial_port or not self.file:
            print("Serial port or file not opened.")
            return
        writer = csv.writer(self.file, delimiter=",")
```

```
        writer.writerow(
            ["unixTimestamp", "ucTimestamp", "current_ma", "voltage_v", "power_mw"]
        )
        for _ in range(num_samples):
            try:
                s_bytes = self.serial_port.readline()
                pc_timestamp = time.time()
                decoded_bytes = s_bytes.decode("utf-8").strip("\r\n")
                values = [float(x) for x in decoded_bytes.split(",")]
                values.insert(0, pc_timestamp)

                writer = csv.writer(self.file, delimiter=",")
                writer.writerow(values)
            except Exception as e:
                print(f"Error encountered serial: {e}")

    def close(self):
        if self.file:
            self.file.close()
```

B.3. Clase usada en la conexión con la Raspberry Pi 4B

```
import paramiko
import os
import time
from scp import SCPClient


class RPiSSHHandler:
    def __init__(self, host="raspberrypi.local"):
        self.host = host
        self.username = os.getenv("RPI_user")
        self.password = os.getenv("RPI_pass")
        self.ssh_client = paramiko.SSHClient()
        self.ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        self.exec_result = ""
        self.is_connected = False

    def connect(self):
        print("Connecting to Raspberry Pi 4B...")
```

```
        while not self.is_connected:
            self.ssh_client.connect(
                hostname=self.host,
                username=self.username,
                password=self.password,
                allow_agent=False,
                look_for_keys=False,
            )
            self.is_connected = True
        self.is_connected = False
        print("Connection established!")

    def run_command(self, cmd):
        try:
            _, stdout, _ = self.ssh_client.exec_command(cmd)
            self.exec_result = stdout.read().decode()
            time.sleep(5)
        except Exception as err:
            print(f"Command execution failed: {err}")

    def start_cpu_logging(self, log_file, log_path):
        sample_time = "250000"
        command = f"cd {log_path} && ./cpu_monitor {sample_time} {log_file}"
        self.ssh_client.exec_command(command)

    def stop_cpu_logging(self):
        _, stdout, _ = self.ssh_client.exec_command("pidof ./cpu_monitor")
        pid = stdout.read().decode().strip()
        if pid:
            self.ssh_client.exec_command(f"kill -2 {pid}")
            print("CPU logging stopped")

    def scp_client(self):
        return SCPClient(self.ssh_client.get_transport())

    def close(self):
        self.ssh_client.close()
```

B.4. Script de medición de uso de CPU

```c
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <unistd.h>

#define BUFFER_SIZE 1024

FILE *outputFile = NULL;

void sigint_handler(int signum) {
  printf("Finished measuring CPU usage\n");
  if (outputFile != NULL) {
    fclose(outputFile);
  }
  exit(signum);
}

int main(int argc, char *argv[]) {
  // Registering SIGINT handler
  if (signal(SIGINT, sigint_handler) == SIG_ERR) {
    printf("Failed to register SIGINT handler\n");
    return 1;
  }

  // Check if sleep duration provided as command-line argument
  if (argc != 3) {
    fprintf(stderr, "Usage: %s <sleep_duration_seconds> <output_file>\n",
            argv[0]);
    return 1;
  }

  int sleepDurationSeconds = atoi(argv[1]);
  char *outputFileName = argv[2];

  // Open output file for writing
  FILE *outputFile = fopen(outputFileName, "w");
```

```
39    if (outputFile == NULL) {
40      fprintf(stderr, "Error opening output file\n");
41      return 1;
42    }
43
44    // Variables to store Unix timestamps
45    time_t start_time, end_time;
46    struct timespec current_time;
47
48    fprintf(outputFile, "Timestamp,CPU_usage\n");
49
50    fflush(outputFile);
51    printf("Started measuring CPU usage\n");
52
53    // Infinite loop to continuously monitor CPU usage
54    while (1) {
55
56      // Open /proc/stat file before sleep
57      FILE *stat_file_before = fopen("/proc/stat", "r");
58      if (stat_file_before == NULL) {
59        fprintf(stderr, "Error opening /proc/stat\n");
60        return 1;
61      }
62
63      // Variables for reading file and parsing CPU statistics before sleep
64      char buffer_before[BUFFER_SIZE];
65      long long user_before = 0, nice_before = 0, system_before = 0,
66                idle_before = 0, iowait_before = 0, irq_before = 0,
67                softirq_before = 0, steal_before = 0, guest_before = 0,
68                guest_nice_before = 0;
69
70      // Read and parse CPU statistics before sleep
71      if (fgets(buffer_before, BUFFER_SIZE, stat_file_before) == NULL) {
72        fprintf(stderr, "Error reading /proc/stat\n");
73        fclose(stat_file_before);
74        return 1;
75      }
76
77      sscanf(buffer_before,
78             "cpu %lld %lld %lld %lld %lld %lld %lld %lld %lld %lld",
79             &user_before, &nice_before, &system_before, &idle_before,
```

```
80              &iowait_before, &irq_before, &softirq_before, &steal_before,
81              &guest_before, &guest_nice_before);
82
83      // Close /proc/stat file
84      fclose(stat_file_before);
85
86      // Sleep for specified duration
87      usleep(sleepDurationSeconds);
88
89      // Capture end time
90      // end_time = time(NULL);
91      clock_gettime(CLOCK_REALTIME, &current_time);
92
93      // printf("End Time: %ld\n", end_time);
94
95      // Open /proc/stat file after sleep
96      FILE *stat_file_after = fopen("/proc/stat", "r");
97      if (stat_file_after == NULL) {
98        fprintf(stderr, "Error opening /proc/stat\n");
99        return 1;
100     }
101
102     // Variables for reading file and parsing CPU statistics after sleep
103     char buffer_after[BUFFER_SIZE];
104     long long user_after = 0, nice_after = 0, system_after = 0, idle_after = 0,
105               iowait_after = 0, irq_after = 0, softirq_after = 0,
106               steal_after = 0, guest_after = 0, guest_nice_after = 0;
107
108     // Read and parse CPU statistics after sleep
109     if (fgets(buffer_after, BUFFER_SIZE, stat_file_after) == NULL) {
110       fprintf(stderr, "Error reading /proc/stat\n");
111       fclose(stat_file_after);
112       return 1;
113     }
114
115     sscanf(buffer_after,
116            "cpu %lld %lld %lld %lld %lld %lld %lld %lld %lld %lld", &user_after,
117            &nice_after, &system_after, &idle_after, &iowait_after, &irq_after,
118            &softirq_after, &steal_after, &guest_after, &guest_nice_after);
119
120     // Close /proc/stat file
```

```
121       fclose(stat_file_after);
122
123       // Calculate CPU usage percentage
124       long long total_before = user_before + nice_before + system_before +
125                                idle_before + iowait_before + irq_before +
126                                softirq_before + steal_before;
127       long long total_after = user_after + nice_after + system_after +
128                               idle_after + iowait_after + irq_after +
129                               softirq_after + steal_after;
130
131       long long Previdle = idle_before + iowait_before;
132       long long Postidle = idle_after + iowait_after;
133
134       long long total_diff = total_after - total_before;
135       long long idle_diff = Postidle - Previdle;
136
137       // printf("%lli,%lli\n", total_diff, idle_diff);
138
139       double cpu_usage = 100.0 * (1.0 - (double)idle_diff / total_diff);
140
141       long long end_time =
142           current_time.tv_sec * 1000LL + current_time.tv_nsec / 1000000LL;
143
144
145       fprintf(outputFile, "%lli,%.4f\n", end_time, cpu_usage);
146
147       // Flush the output buffer to ensure data is written to file
148       fflush(outputFile);
149     }
150
151     return 0;
152   }
```

Apéndice C

Procesamiento de Datos y Entrenamiento de red

```python
import polars as pl
import numpy as np

def encontrar_indices_mas_cercanos(marcas_de_tiempo, arreglo_busqueda):
    objetivos_normalizados = np.array(marcas_de_tiempo) / 1000.0
    fuente_de_datos = np.array(arreglo_busqueda)

    puntos_cercanos = []
    for ts_objetivo in objetivos_normalizados:
        diferencias = np.abs(fuente_de_datos - ts_objetivo)
        indice_minimo = diferencias.argmin()
        puntos_cercanos.append((indice_minimo, fuente_de_datos[indice_minimo]))

    return puntos_cercanos

ruta_mediciones = "../data/new_data/2024-06-19_21-37-07_dynamic_refresh_time_60seconds.csv"
ruta_metadatos = "../data/new_data/dynamic_refresh_time_60seconds_experiment_metadata.csv"

datos_potencia_brutos = pl.read_csv(ruta_mediciones)
metadatos_experimento = pl.read_csv(ruta_metadatos)

matriz_mediciones = datos_potencia_brutos.to_numpy()
timestamps_pc = matriz_mediciones[:, 0]
timestamps_uc = matriz_mediciones[:, 1]
corriente_medida = matriz_mediciones[:, 2]
```

```
voltaje_medido = matriz_mediciones[:, 3]
potencia_calculada = matriz_mediciones[:, 4]

duracion_experimento_ms = 300 * 1e3
timestamp_final = metadatos_experimento['startTimestamp'] + duracion_experimento_ms
metadatos_experimento = metadatos_experimento.with_columns(
    endTimestamp = timestamp_final
)

indices_inicio = encontrar_indices_mas_cercanos(
    metadatos_experimento['startTimestamp'], datos_potencia_brutos['unixTimestamp']
)
indices_fin = encontrar_indices_mas_cercanos(
    metadatos_experimento['endTimestamp'], datos_potencia_brutos['unixTimestamp']
)

df_segmentacion = pl.DataFrame({
    "numero_sensor": metadatos_experimento['sensorNumber'],
    "info_inicio": indices_inicio,
    "info_fin": indices_fin
}, strict=False)

contador_experimento = 1
ruta_salida = "../data/new_data/dynamic_refresh_time_60_refresh_time/"

for sensor_id, info_inicio, info_fin in zip(df_segmentacion['numero_sensor'],
↪  df_segmentacion['info_inicio'], df_segmentacion['info_fin']):

    idx_start = int(info_inicio[0])
    idx_end = int(info_fin[0])

    segmento_potencia = potencia_calculada[idx_start:idx_end]
    segmento_corriente = corriente_medida[idx_start:idx_end]
    segmento_voltaje = voltaje_medido[idx_start:idx_end]
    segmento_ts_uc = timestamps_uc[idx_start:idx_end]
    segmento_ts_pc = timestamps_pc[idx_start:idx_end]

    df_experimento = pl.from_dict({
        "unixTimestamp": segmento_ts_pc,
        "ucTimestamp": segmento_ts_uc,
        "current_ma": segmento_corriente,
```

```
            "voltage_v": segmento_voltaje,
            "power_mw": segmento_potencia
        })

        nombre_archivo = f"numSensor_{sensor_id}_numExperiment_{contador_experimento}.csv"
        df_experimento.write_csv(f"{ruta_salida}{nombre_archivo}", separator=",")

        contador_experimento += 1
        if contador_experimento > 25:
            contador_experimento = 1

import os
import polars as pl
import numpy as np
from scipy.optimize import curve_fit
from sklearn.metrics import r2_score

def calcular_energia_acumulada(potencia_en_watts, frecuencia_muestreo):
    intervalo_tiempo = 1.0 / frecuencia_muestreo
    energia = np.cumsum(potencia_en_watts) * intervalo_tiempo
    return energia[1:]

def modelo_lineal(t, pendiente, intercepto):
    return pendiente * t + intercepto

directorio_datos = "../data/final_data/dynamic_refresh_time_60_refresh_time/"
frecuencia_hz = 333.0
tolerancia_muestras = frecuencia_hz * 2

archivos_csv = [f for f in os.listdir(directorio_datos) if f.endswith(".csv")]

vectores_energia_validos = []
ids_sensores_validos = []
indices_filtrados = []

for idx, nombre_archivo in enumerate(archivos_csv):
    try:
        partes = nombre_archivo.split('_')
        id_sensor = int(partes[1])
    except (IndexError, ValueError):
        indices_filtrados.append(idx)
```

```
107             continue
108
109         ruta_completa = os.path.join(directorio_datos, nombre_archivo)
110         df = pl.read_csv(ruta_completa)
111
112         potencia_watts = df["power_mw"].to_numpy() * 1e-3
113
114         longitud_esperada = 100000
115         if not (longitud_esperada - tolerancia_muestras < len(potencia_watts) < longitud_esperada +
            ↪  tolerancia_muestras):
116             indices_filtrados.append(idx)
117             continue
118
119         energia_joules = calcular_energia_acumulada(potencia_watts, frecuencia_hz)
120         vectores_energia_validos.append(energia_joules)
121         ids_sensores_validos.append(id_sensor)
122
123     pendientes = []
124     interceptos = []
125     valores_r_cuadrado = []
126
127     intervalo_muestreo = 1.0 / frecuencia_hz
128
129     for vector_energia in vectores_energia_validos:
130         eje_tiempo = np.arange(intervalo_muestreo, (len(vector_energia) + 1) * intervalo_muestreo,
            ↪  intervalo_muestreo) - intervalo_muestreo
131         intercepto_fijo = vector_energia[0]
132
133         try:
134             params, _ = curve_fit(lambda t, m: modelo_lineal(t, m, intercepto_fijo), eje_tiempo,
                ↪  vector_energia)
135             pendiente_optima = params[0]
136
137             energia_predicha = modelo_lineal(eje_tiempo, pendiente_optima, intercepto_fijo)
138             r2 = r2_score(vector_energia, energia_predicha)
139
140             pendientes.append(pendiente_optima)
141             interceptos.append(intercepto_fijo)
142             valores_r_cuadrado.append(r2)
143         except RuntimeError:
144             pass
```

```
145
146 periodo_refresco = 60
147 df_coeficientes = pl.DataFrame({
148     "refresh_period": [periodo_refresco] * len(ids_sensores_validos),
149     "number_sensors": np.array(ids_sensores_validos, dtype=np.int32),
150     "m": pendientes,
151     "b": interceptos,
152     "r_squared": valores_r_cuadrado
153 })
154
155 ruta_salida_coef = '../data/final_data/coefficients_fixed_b/60sec.csv'
156 df_coeficientes.write_csv(ruta_salida_coef, separator=",")
157
158 df_curvas_energia = pl.DataFrame({
159     "refresh_period": [periodo_refresco] * len(ids_sensores_validos),
160     "number_sensors": np.array(ids_sensores_validos, dtype=np.int32),
161     "energy_curve": vectores_energia_validos
162 })
163
164 ruta_salida_curvas = '../data/final_data/coefficients_fixed_b/curves/60sec_curves.json'
165 df_curvas_energia.write_ndjson(ruta_salida_curvas)
166
167 import os
168 import numpy as np
169 import polars as pl
170 import tensorflow as tf
171 from tensorflow.keras.models import Model, load_model
172 from tensorflow.keras.layers import Input, Dense
173 from tensorflow.keras import backend as K
174 from sklearn.model_selection import train_test_split
175 from sklearn.preprocessing import StandardScaler
176 from scipy.stats import iqr
177 from joblib import dump, load
178 import matplotlib.pyplot as plt
179
180 def perdida_pinball(cuantil_tau):
181     def loss(y_verdadero, y_predicho):
182         error = y_verdadero - y_predicho
183         return K.mean(K.maximum(cuantil_tau * error, (cuantil_tau - 1) * error), axis=-1)
184     return loss
185
```

```python
def construir_modelo_cuantilico(n_entradas, cuantiles, neuronas_capa1=52, neuronas_capa2=102,
    tasa_aprendizaje=0.005):
    entradas = Input(shape=(n_entradas,))
    x = Dense(neuronas_capa1, activation='relu')(entradas)
    x = Dense(neuronas_capa2, activation='relu')(x)

    salidas = [Dense(1, name=f"cuantil_{int(q*100)}")(x) for q in cuantiles]

    modelo = Model(inputs=entradas, outputs=salidas)

    optimizador = tf.keras.optimizers.Adam(learning_rate=tasa_aprendizaje)
    losses = {f'cuantil_{int(q*100)}': perdida_pinball(q) for q in cuantiles}

    modelo.compile(optimizer=optimizador, loss=losses)
    return modelo

directorio_coeficientes = "../data/final_data/coefficients_fixed_b/"
archivos_coeficientes = [f for f in os.listdir(directorio_coeficientes) if f.endswith(".csv")]

lista_de_dfs = []
for nombre_archivo in archivos_coeficientes:
    ruta_completa = os.path.join(directorio_coeficientes, nombre_archivo)
    df = pl.read_csv(ruta_completa)

    q25, q75 = np.percentile(df['m'], [25, 75])
    rango_iqr = q75 - q25
    limite_superior = q75 + 1.5 * rango_iqr
    limite_inferior = q25 - 1.5 * rango_iqr

    df_limpio = df.filter((pl.col('m') > limite_inferior) & (pl.col('m') < limite_superior))
    lista_de_dfs.append(df_limpio)

datos_completos = pl.concat(lista_de_dfs)

features = np.column_stack([
    datos_completos['refresh_period'].to_numpy(),
    datos_completos['number_sensors'].to_numpy()
])
target = datos_completos['m'].to_numpy().reshape(-1, 1)

escalador_features = StandardScaler()
```

```
escalador_target = StandardScaler()

features_scaled = escalador_features.fit_transform(features)
target_scaled = escalador_target.fit_transform(target)

dump(escalador_features, 'escalador_entrada.bin', compress=True)
dump(escalador_target, 'escalador_salida.bin', compress=True)

X_ent, X_pru, y_ent, y_pru = train_test_split(features_scaled, target_scaled, test_size=0.2,
↪  random_state=42)

CUANTILES_OBJETIVO = [0.05, 0.5, 0.95]

modelo = construir_modelo_cuantilico(
    n_entradas=2,
    cuantiles=CUANTILES_OBJETIVO,
    neuronas_capa1=52,
    neuronas_capa2=102,
    tasa_aprendizaje=0.0051,
)

historial = modelo.fit(
    X_ent,
    [y_ent for _ in CUANTILES_OBJETIVO],
    batch_size=18,
    epochs=200,
    verbose=1,
    validation_split=0.2
)

modelo.save('./modelo_energia_refactorizado.h5')

predicciones_escaladas = modelo.predict(X_pru)

predicciones_desescaladas = [escalador_target.inverse_transform(p) for p in predicciones_escaladas]
y_pru_desescalado = escalador_target.inverse_transform(y_pru)
X_pru_desescalado = escalador_features.inverse_transform(X_pru)

plt.style.use('seaborn-v0_8-whitegrid')
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(18, 7))

```

```
266 ax1.scatter(X_pru_desescalado[:, 0], y_pru_desescalado, color='blue', alpha=0.5, label='Valores Reales
    ↪ (m)')
267 ax1.scatter(X_pru_desescalado[:, 0], predicciones_desescaladas[1], color='black', s=20,
    ↪ label='Predicción Mediana (Cuantil 50)')
268 ax1.scatter(X_pru_desescalado[:, 0], predicciones_desescaladas[0], color='red', s=20, alpha=0.7,
    ↪ label='Límite Inferior (Cuantil 5)')
269 ax1.scatter(X_pru_desescalado[:, 0], predicciones_desescaladas[2], color='red', s=20, alpha=0.7,
    ↪ label='Límite Superior (Cuantil 95)')
270 ax1.set_xlabel('Periodo de Refresco (s)', fontsize=12)
271 ax1.set_ylabel('Coeficiente "m"', fontsize=12)
272 ax1.set_title('Predicciones vs Periodo de Refresco', fontsize=14)
273 ax1.legend()
274
275 ax2.scatter(X_pru_desescalado[:, 1], y_pru_desescalado, color='blue', alpha=0.5, label='Valores Reales
    ↪ (m)')
276 ax2.scatter(X_pru_desescalado[:, 1], predicciones_desescaladas[1], color='black', s=20,
    ↪ label='Predicción Mediana (Cuantil 50)')
277 ax2.scatter(X_pru_desescalado[:, 1], predicciones_desescaladas[0], color='red', s=20, alpha=0.7,
    ↪ label='Límite Inferior (Cuantil 5)')
278 ax2.scatter(X_pru_desescalado[:, 1], predicciones_desescaladas[2], color='red', s=20, alpha=0.7,
    ↪ label='Límite Superior (Cuantil 95)')
279 ax2.set_xlabel('Número de Sensores', fontsize=12)
280 ax2.set_ylabel('Coeficiente "m"', fontsize=12)
281 ax2.set_title('Predicciones vs Número de Sensores', fontsize=14)
282 ax2.legend()
283
284 plt.tight_layout()
285 plt.savefig('../images/model_benchmark/fixed_b/evaluacion_modelo_refactorizado.png', dpi=300)
286 plt.show()
287
288 num_ejecuciones = 10
289 resultados_metricas = {'cobertura_90': [], 'perdida_pinball_promedio': [], 'ancho_intervalo_promedio':
    ↪ []}
290
291 for i in range(num_ejecuciones):
292     X_ent_val, X_pru_val, y_ent_val, y_pru_val = train_test_split(features_scaled, target_scaled,
        ↪ test_size=0.2, random_state=i)
293
294     modelo_val = construir_modelo_cuantilico(2, CUANTILES_OBJETIVO)
295     modelo_val.fit(X_ent_val, [y_ent_val]*3, epochs=200, batch_size=18, verbose=0)
296
```

```
297         y_pred_val = modelo_val.predict(X_pru_val)
298         y_pru_inv = escalador_target.inverse_transform(y_pru_val)
299         y_pred_inv = [escalador_target.inverse_transform(p) for p in y_pred_val]
300 
301         cobertura = np.mean((y_pru_inv >= y_pred_inv[0]) & (y_pru_inv <= y_pred_inv[2]))
302         ancho_intervalo = np.mean(y_pred_inv[2] - y_pred_inv[0])
303 
304         resultados_metricas['cobertura_90'].append(cobertura)
305         resultados_metricas['ancho_intervalo_promedio'].append(ancho_intervalo)
306 
307     for metrica, valores in resultados_metricas.items():
308         if metrica != 'perdida_pinball_promedio':
309             media = np.mean(valores)
310             desv_est = np.std(valores)
311             print(f"{metrica.replace('_', ' ').title()}: {media:.4f} ± {desv_est:.4f}")
```

C.1. Extracto del dataset recopilado


```
{"refresh_period":1,"number_sensors":125,"energy":[0.01602792732732733,0.030658864564564563,0.04402472
↪ 072072072,0.0553
↪ 5786576576577,0.0661898033033033,0.07681952852852852,0.08767587117117118,0.09853677927927929,0.10
↪ 926038138138139,0.117
↪ 64483093093094,0.12565364414414415,0.1336450033033033,0.14177353333333334,0.14979005945945947,0.1
↪ 5805801801801803,0.16
↪ 613316066066067,0.17415380330330332,0.18218985825825826,0.19021666546546545,0.19826967417417418,0
↪ .2062671981981982,0.2
↪ 142755099099099,0.22226995105105102,0.2302803069069069,0.23834256276276275,0.24633700390390392,0.
↪ 25434531561561563,0.2
↪ 623397567567568,0.27035165585585585,0.2783661333333333,0.2863775279279279,0.2943889225225225,0.30
↪ 24403900900901,0.3104
↪ 5074594594596,0.31846572942942947,0.3264683723723724,0.3344736027027027,0.34247883303303306,0.350
↪ 4732741741742,0.36014
↪ 82672672673,0.3713602186186186,0.38347512462462463,0.3960714627627627,0.40751323693693686,0.41916
↪ 366426426427,0.428624
↪ 15165165163,0.4366349993993994,0.4446468984984985,0.45268656306306304,0.4607688561561561,0.469231
↪ 4513513513,0.47769326
↪ 93693693,0.48947257357357354,0.5021436039039039,0.5147402132132132,0.5285854426426426,0.541519920
↪ 7207207,0.55424119849
↪ 84985,0.5670527936936937,0.5799611918918919,0.59290976996997,0.606081821021021,0.618980724924925,
↪ 0.6315609072072073,0.
↪ 6413799960960961,0.6512205003003003,0.661052745045045,0.6708488882882883,0.6807224000000001,0.690
↪ 5974402402403,0.70047
↪ 09519519519,0.7100969981981982,0.7203999465465465,0.7307375117117118,0.7402145330330331,0.7496271
↪ 996996998,0.759044462
↪ 4624626,0.7684413558558559,0.778584552852853,0.7895166552552554,0.8006901423423425,0.811762391891
↪ 892,0.821922601801801
↪ 9,0.8320526525525526,0.8424516480480482}
```