Universidad de la Defensa Nacional - Centro Regional Universitario Córdoba - Instituto Universitario Aeronáutico (CRUC-IUA) Facultad de Ingeniería - Departamento de Electrónica y Telecomunicaciones

Sistema Dinámico de Adquisición y Alerta de Velocidad para la Concientización de Conductores Vehiculares

Trabajo Final de Grado

Ipharraguerre, Facundo

Tutor:

Ing. Esp. José María Ducloux

Palabras clave

Armbian

Autocorrelación de señales

Ciudades inteligentes

Comunicación inalámbrica

InfluxDB

Linux

Medición de velocidad

Microcontrolador

Procesamiento digital de señales

Puerta de enlace IoT

Python

Radar Doppler

Redis

STM32

Índice general

Dedicatoria y agradecimientos

A mi familia, amigos y compañeros, gracias por estar siempre cuando se los necesita.

A mis profesores, gracias por su dedicación y por compartir sus conocimientos. Agradezco a la Educación Pública y al staff docente y no docente del Instituto Universitario Aeronáutico.

Dedicado a mí mismo, que, entre perseverancia, obstinación y determinación, logré comenzar, transitar y cerrar un ciclo de relativa importancia, aun cuando las circunstancias no acompañaron.

También dedicado a los que no lo creyeron posible (me incluyo).

Resumen

En este Trabajo Final de Grado se ha desarrollado una plataforma de telemetría la cual ha sido aplicada para medir la velocidad de los vehículos en un sitio determinado. El propósito principal es proporcionar a los conductores información instantánea y comprensible sobre su velocidad de conducción, al tiempo que se almacenan los datos para su posterior análisis. Esta iniciativa busca asistir a los conductores en la adaptación de su estilo de conducción y promover la responsabilidad individual.

El trabajo realizado se enfoca en el análisis, diseño e implementación de un sistema de adquisición y procesamiento de señales de frecuencia intermedia de un radar Doppler de microondas permitiendo visualizar la velocidad de los vehículos en una pantalla LED ubicada en el área de cobertura del radar. En el proyecto se utilizan sistemas embebidos que se integran con soluciones de bases de datos de series temporales en un servidor de Internet. Además, se ha considerado la escalabilidad del proyecto, permitiendo la adición de nuevas estaciones o puntos de control sin afectar el funcionamiento de las estaciones existentes. Los resultados obtenidos en términos de escalabilidad y estabilidad son consistentes con las expectativas previas.

Abstract

In this Final Degree Project, a telemetry platform has been developed, which has been applied to measure vehicle speed at a specific location. The primary purpose is to provide drivers with instant and understandable information about their driving speed while storing data for subsequent analysis. This initiative aims to assist drivers in adapting their driving style and promoting individual responsibility.

The work focuses on the analysis, design, and implementation of a system for acquiring and processing intermediate frequency signals from a microwave Doppler radar, allowing the visualization of vehicle speed on an LED display located within the radar's coverage area. The project employs embedded systems integrated with time-series database solutions on an Internet server. Scalability has been considered, allowing the addition of new stations or checkpoints without affecting the operation of existing ones. The results obtained in terms of scalability and stability are consistent with prior expectations.

Índice de Figuras

Índice de Tablas

1. Introducción

1.1 Contexto temático

En Argentina, la siniestralidad vial es una preocupación constante. Según el Informe de Siniestralidad Vial Fatal del Año 2021, se registraron 6.637 víctimas fatales en todo el país. Buenos Aires, Santa Fe y Córdoba fueron las provincias con mayor cantidad de fallecidos (Argentina.gob.ar, 2021). En particular, Córdoba reportó 254 víctimas fatales en ese período (Argentina.gob.ar, 2022).

La velocidad de circulación vehicular está directamente relacionada con la seguridad vial. Estudios demuestran que por cada 1 km/h de aumento en la velocidad promedio, aumenta aproximadamente un 2% la frecuencia de lesiones en accidentes viales. Además, un incremento en la velocidad también se traduce en una mayor gravedad de los accidentes (OCDE, 2018; Argentina.gob.ar, 2021; tuteorica, s.f.). La falta de conciencia sobre la velocidad es un problema, y muchos conductores no son conscientes de la velocidad a la que circulan.

Para abordar esta problemática, se utilizan sistemas de medición de velocidad, como radares y velocímetros. Estos sistemas ayudan a controlar y hacer cumplir los límites de velocidad. Los radares Doppler, por ejemplo, miden la velocidad de los vehículos y son ampliamente utilizados en aplicaciones de tráfico y seguridad vial (Físicas, 2017). Es fundamental promover la conciencia sobre la importancia de respetar los límites de velocidad para reducir la siniestralidad y garantizar la seguridad en nuestras comunidades (Argentina.gob.ar, 2021).

1.2 Definición del problema

La falta de conciencia sobre la velocidad de circulación vehicular es una problemática muy relacionada a la insonorización del habitáculo de los vehículos y las mejoras en la suspensión de estos. Si bien está probado que la velocidad es un factor determinante en la cantidad y gravedad de los accidentes de tránsito, muchas veces los conductores no son conscientes de la velocidad a la que circulan. Con el aumento del parque automotor y el crecimiento de las ciudades, esta problemática se hace cada vez más evidente y es necesario tomar conciencia sobre la importancia de respetar los límites de velocidad y promover la seguridad vial general en nuestras comunidades.

1.3 Motivación

Para la concepción de este sistema se parte de la premisa de que "lo que no se mide no se puede llegar a conocer y lo que no se conoce no se puede cambiar o mejorar". Al intentar promover un cambio en el comportamiento de los conductores apoyándose en la responsabilidad individual y al realizar el muestreo y mostrar a los conductores la velocidad actual de su vehículo, este sistema busca obtener resultados a corto y largo plazo, mejorando la seguridad en las áreas donde es desplegado. A la vez se intenta enfocar el proyecto a dar una solución versátil y fácilmente adaptable a la obtención de otros datos de telemetría.

1.4 Objetivos

1.4.1 Objetivo general

Desarrollar una plataforma orientada a obtener datos de telemetría, aplicándola particularmente a obtener la velocidad de vehículos registrada en un sitio, con la finalidad de mostrar a los conductores la velocidad a la cual circulan y al mismo tiempo almacenar los datos para posibilitar su análisis.

1.4.2 Objetivos específicos

- Desarrollar un sistema embebido capaz de medir la velocidad de vehículos mediante el procesamiento de la señal proporcionada por un radar de microondas.
- Desarrollar un dispositivo concentrador o puerta de enlace que se encargará de recolectar los datos desde los sistemas embebidos (*endpoints*) y enviarlos a un servidor de Internet.
- Implementar un servicio de Internet, que permita la recopilación y la visualización de los datos adquiridos.
- Posibilitar el despliegue de una red de dispositivos de medición de velocidad vehicular en una zona o región, los cuales se comunicarán con la puerta de enlace mediante una interfaz inalámbrica.
- Diseñar un protocolo de comunicación bidireccional eficiente para la comunicación entre los dispositivos de medición de velocidad vehicular y la puerta de enlace.
- Desarrollar esta plataforma de manera iterativa, realizando pruebas en todas sus fases para incrementar gradualmente y por etapas la funcionalidad del sistema.

1.5 Contribución esperada y alcance

Para asistir a los usuarios en la adecuación de su modo de conducción y promover la responsabilidad individual, los conductores necesitan contar con acceso a la información de manera instantánea y comprensible.

El proyecto se limita a desarrollar un *stack* para habilitar la medición de velocidad, mostrar el resultado y almacenarlo en una base de datos en la 'nube'. No se tienen en cuenta los sensores adicionales que podrían conectarse a los *Endpoints*, ni el desarrollo de las fuentes de alimentación. Tampoco se encuentra dentro del alcance el cómo se utiliza la información obtenida y almacenada, por ejemplo, queda fuera del alcance del proyecto compatibilizar los datos para poder ser utilizados por un controlador de semáforos o un sistema de fotomulta. Cabe destacar que, para esta primera versión, se encuentra fuera del alcance del proyecto el control de colisiones entre estaciones para la comunicación contra el dispositivo concentrador o *gateway*, sin embargo, está concebido de forma que sea relativamente directa la implementación del control de flujo y paridad en las comunicaciones en futuras mejoras.

1.6 Metodología

De forma general, existen al menos dos enfoques para abordar la implementación de sistemas: 'top down' y 'bottom up'. Cada uno de estos enfoques posee ciertas características que determinan su proceso de desarrollo. En el contexto del diseño e implementación del sistema propuesto en este proyecto, se exploraron estas dos metodologías.

De forma general, la metodología del proyecto incluye los siguientes pasos:

1. **Revisión** bibliográfica: se realiza una revisión de la literatura sobre la medición de velocidad de vehículos y el procesamiento de señales de radar de microondas de efecto Doppler, así como sobre el diseño de sistemas embebidos, de protocolos de comunicación y las bases de datos que mejor se ajustan para el propósito del proyecto.
2. **Diseño** del sistema embebido: diseñar un sistema embebido capaz de medir la velocidad de vehículos mediante el procesamiento de la señal proporcionada por un radar de microondas de efecto Doppler. Esto podría incluir la selección de los componentes, determinar si es necesario acondicionar la señal, la programación del microcontrolador y la realización de pruebas preliminares.
3. **Diseño** y prueba de las comunicaciones: diseñar un protocolo de comunicación bidireccional eficiente para la comunicación entre los *Endpoints* y cada puerta de enlace. Esto podría incluir la realización de pruebas preliminares para determinar la eficacia del protocolo y realizar ajustes según sea necesario.
4. **Desarrollo** del dispositivo concentrador o puerta de enlace: desarrollar un dispositivo concentrador o puerta de enlace utilizando Python y Linux que se encargará de recolectar los datos desde los sistemas embebidos y enviarlos a un servidor de Internet. Esto podría incluir el diseño de la interfaz de usuario y la implementación del protocolo de comunicación bidireccional que se diseñó en la etapa previa.
5. **Implementación** del servidor de Internet: implementar un servicio en la nube o en un servidor privado que sea alcanzable a través de Internet, habilitando un sitio web para la visualización y almacenamiento de los datos adquiridos. Por ejemplo, puede utilizarse InfluxDB, que proporciona una interfaz para visualizar los datos a través de tableros de Grafana. Esta etapa podría incluir la selección del servicio, la configuración del software de la aplicación y la integración de la interfaz de usuario.
6. **Despliegue** de los dispositivos de medición de velocidad vehicular: desplegar los dispositivos de medición de velocidad vehicular en una zona, los cuales se comunicarán con la puerta de enlace usando su interfaz inalámbrica. Este punto podría incluir la selección de los lugares adecuados para la instalación de los dispositivos, la configuración de la red y la realización de pruebas preliminares.
7. **Desarrollo** iterativo: desarrollar la plataforma de manera iterativa, realizando pruebas en todas sus fases para incrementar gradualmente y por etapas la funcionalidad del sistema. Esto podría incluir la identificación y solución de problemas, la integración de nuevas características y la mejora de la eficiencia y la escalabilidad del sistema.

2. Marco teórico

2.1 Internet de las cosas (IoT)

2.1.1 ¿Qué es IoT o 'Internet de las cosas'?

Internet de las Cosas (IoT) se refiere a la conexión de elementos físicos cotidianos a Internet, desde objetos domésticos comunes hasta dispositivos médicos y sistemas de ciudades inteligentes. IoT básicamente se apoya en la conectividad de dispositivos que actúan como actuadores o sensores, permitiendo la transferencia de información a través de redes de datos con mínima intervención humana. A través de la integración de dispositivos informáticos en una variedad de objetos, IoT posibilita el funcionamiento eficiente de sistemas que analizan y utilizan datos.

2.1.2 IoT Industrial

La interconexión de dispositivos a través del Internet de las Cosas (IoT) ha transformado radicalmente diversas industrias, introduciendo niveles inigualables de eficiencia y control. Este avance tecnológico se ha infiltrado en múltiples verticales, desde la fabricación hasta el comercio minorista, demostrando su capacidad para transformar y optimizar procesos. IoT Industrial (IIoT) emerge como una fuerza capacitadora para empresas, fomentando eficiencias en áreas que van desde la fabricación hasta la logística y la salud.

Por ejemplo, en el ámbito de las máquinas de construcción, diversos sensores se integran en partes propensas a daños, permitiendo un mantenimiento predictivo y mejorando el rendimiento del personal. Este enfoque resalta la inmediatez de la recopilación y el análisis de datos, junto con la capacidad de realizar análisis a largo plazo para optimizar modelos futuros. IoT Industrial (IIoT) se entrelaza con las operaciones de diversos sectores industriales, desde fábricas hasta empresas de energía y transporte, al ofrecer soluciones prácticas para la toma de decisiones informadas.

En el ámbito de la logística y el transporte, IoT presenta una transformación en la cadena de suministro, desde el etiquetado de contenedores con dispositivos de identificación por radiofrecuencia (RFID) hasta sistemas de seguimiento que transmiten datos de telemetría en todo momento, prescindiendo de lectores en las instalaciones. Esto posibilita un análisis continuo y detallado de los envíos, optimizando la eficiencia y la visibilidad en cada etapa. En esta era de IoT aplicado, la logística y el transporte se destacan como áreas de oportunidad, donde la conectividad y la recopilación de datos en tiempo real mejoran la eficiencia, la seguridad y la experiencia del cliente, transformando la forma en que se gestionan y ejecutan los procesos logísticos y de transporte.

Finalmente, en el sector agrícola, IoT despliega sensores de humedad en campos, proporcionando a los agricultores datos precisos para la planificación y la activación automatizada del riego basada en datos de sensores.

2.2 Bases de Datos

2.2.1 Redis

Redis, "*Remote Dictionary Server*" (Servidor de Diccionario Remoto), es un motor de base de datos en memoria y modelo clave/valor, de alto rendimiento, ampliamente adoptado en la industria. Va más allá de ser un simple almacén clave/valor, ya que también actúa como un servidor de estructuras de datos, lo que significa que admite diferentes tipos de valores.

En Redis, la flexibilidad radica en la capacidad de asignar distintos tipos de datos como valores, tales como cadenas de texto, números enteros, listas, conjuntos, hashes y conjuntos ordenados. Esta versatilidad facilita diversas aplicaciones, como el almacenamiento de resultados de consultas, el conteo y clasificación de elementos, la implementación de colas y mensajería, y la gestión de sesiones de usuario.

La eficiencia de Redis es particularmente apreciada en escenarios que demandan respuestas rápidas y alto rendimiento, como en servicios de juegos, publicidad, servicios financieros, atención médica e IoT. Su diseño basado en memoria proporciona acceso de baja latencia, ya que opera completamente en memoria, eliminando la latencia asociada al acceso a disco presente en bases de datos tradicionales.

Además de su rendimiento, Redis se destaca como un proyecto de código abierto respaldado por una activa comunidad. Al adherirse a estándares abiertos, evita el '*vendor-lock*', y ofrece una variedad de clientes y librerías de software para su integración en distintos entornos.

2.2.2 InfluxDB

InfluxDB se posiciona como un motor de base de datos especializado en la gestión de series temporales, aplicando principios de diseño que favorecen el rendimiento en operaciones de telemetría. Su estructura se fundamenta en la premisa de que los datos temporales son predominantemente nuevos, minimizando operaciones de sobreescritura y registrándose en orden ascendente.

Este enfoque conlleva ciertas restricciones, como limitaciones en actualizaciones y eliminaciones, pero impulsa la eficiencia en escritura y consulta. Asimismo, el diseño sin esquema de InfluxDB ofrece flexibilidad para manejar datos discontinuos, a pesar de ciertas limitaciones en funcionalidades como las uniones entre tablas.

La infraestructura de InfluxDB se destaca por su capacidad de ingestión, admitiendo varios millones de puntos de datos por segundo con cardinalidad ilimitada. Además, el soporte SQL acelera el desarrollo de consultas y aplicaciones. InfluxDB ofrece un servicio gestionado con un modelo serverless, basando su plan de pago en los recursos utilizados. Para facilitar la adopción, proporciona un plan gratuito que permite realizar pruebas, escalando el servicio conforme evoluciona la aplicación desde el desarrollo hasta la puesta en producción.

2.3 STM32

STM32 representa una familia de microcontroladores que se basa en diversos núcleos ARM Cortex-M de 32 bits. STMicroelectronics, al licenciar la tecnología del procesador ARM, brinda una amplia variedad de opciones configurables en sus diseños de núcleos ARM. Antes de convertir el diseño en un chip de silicio, ST incorpora sus propios periféricos al núcleo ARM.

Comparados con soluciones como ESP, PIC y Arduino, los microcontroladores STM32 ofrecen un rendimiento superior, capacidades en tiempo real, procesamiento eficiente de señales digitales y operación optimizada en términos de consumo y voltaje, además de proporcionar una conectividad robusta. Apreciablemente, estos beneficios no vienen a expensas del presupuesto, ya que los IC de las familias de STM32 se destacan por su asequibilidad y la abundancia de recursos disponibles gracias a su amplia adopción.

La flexibilidad de STM32 se extiende a su compatibilidad con el estándar ARM, la capacidad de manejar interrupciones de manera rápida, independencia de un compilador específico y la ausencia de middleware innecesario. Además, su compatibilidad con Arduino, respaldada por un *bootloader* USB integrado y un sólido soporte de librerías, lo convierte en una opción atractiva. Su costo competitivo, la superioridad tecnológica respecto a alternativas como AVR 328p, y la posibilidad de cambiar el modelo de IC dentro de la misma (o incluso diferente) familia, mientras se mantienen los diseños de *pinout*, contribuyen a su prominencia en la comunidad de desarrollo.

Un aspecto destacado son las opiniones de sus usuarios, que reflejan la percepción positiva de la comunidad respecto a los microcontroladores STM32, como por ejemplo,la percepción sobre la calidad de la documentación, reconocida por su claridad y utilidad, marcando una diferencia sustancial para los desarrolladores. Esto consolida a las familias de ICs de STM32 como una elección confiable y poderosa para proyectos de diversas magnitudes que involucren microcontroladores.

2.3.1 STM32F103

El STM32F103, miembro de la familia de microcontroladores STM32, basados en el procesador Arm Cortex-M, se posiciona como una solución versátil para la adquisición y procesamiento de señales. Este microcontrolador, operando a una frecuencia máxima de 72 MHz, ofrece, según el fabricante, “buen rendimiento, capacidades en tiempo real, procesamiento de señales digitales y operación eficiente en términos de consumo”.

Con una memoria flash de 128 KB y una RAM estática de 20 KB, el STM32F103 se destaca por su GPIO, así como dispositivos vinculados a través de APB (*Advanced Peripheral Bus*). Entre sus características principales se encuentran:

- Convertidores Analógico/Digital (ADC): Equipado con un convertidor ADC de 12 bits, permite la conversión de señales analógicas a digitales, facilitando la interfaz con sensores analógicos y otros dispositivos.

- Módulos de Comunicación: Soporta varios protocolos de comunicación, incluyendo 2 I2C, 2 SPI y 3 USART/UART, ampliando las opciones de conectividad.

- Temporizadores/Contadores: Incorpora timers de 16 bits que se prestan para diversas tareas de temporización, como generar señales PWM, medir duraciones de pulsos y generar interrupciones a intervalos específicos.

- Compatibilidad con Diversos Dispositivos: Admite una amplia variedad de periféricos, desde controladores de juegos hasta sistemas de posicionamiento global (GPS), máquinas industriales y controladores lógicos programables (PLC).

- Rango de Operación: Puede funcionar en un rango de voltaje de 2 a 3,6 volts y operar en temperaturas que van desde -40° C hasta 85° C o incluso hasta 105° C, dependiendo de la serie y el empaque seleccionados, que varían de 36 a 100 pines.

Además, el STM32F103 incluye capacidades que potencian su versatilidad:

- USART/UART: Facilita la comunicación serie síncrona/asíncrona, permitiendo protocolos como UART, USART y LIN.

- SPI: Interfaz de Comunicación Periférica Serial, posibilita la comunicación sincrónica con otros dispositivos en una arquitectura maestro-esclavo.

- I2C: Protocolo de Comunicación Integrada, habilita la comunicación con dispositivos mediante una interfaz de dos hilos conductores (2 *wire*).

- PWM: Modulación por Ancho de Pulso, una técnica para controlar la tensión o corriente suministrada a un dispositivo ajustando el ciclo de trabajo de una señal cuadrada.

- DMA: Acceso Directo a Memoria, permite la transferencia de datos entre periféricos y memoria sin intervención de la CPU, optimizando el rendimiento del sistema.

2.4 Arquitectura ARM

La arquitectura ARM, inicialmente concebida para ordenadores personales, ha evolucionado para convertirse en la arquitectura de 32 y 64 bits más ampliamente adoptada en la producción de unidades. Su diseño RISC (Reduced Instruction Set Computing) proporciona procesadores ARM más eficientes en términos de transistores, traduciéndose en soluciones económicas y energéticamente eficientes. Este perfil eficiente los convierte en la opción predilecta para dispositivos alimentados por batería, como teléfonos móviles y tabletas, consolidando su liderazgo en el ámbito de la electrónica móvil y embebida.

La arquitectura ARM se ha expandido gracias a licencias que permiten a diversas empresas, desde Apple hasta Samsung, desarrollar microcontroladores y CPUs basados en núcleos ARM. La versatilidad de estos procesadores los adapta a una amplia gama de aplicaciones, desde juegos hasta dispositivos médicos y electrodomésticos. La *licenciabilidad* de ARM ha catalizado su adopción en la industria, diversificando el panorama de la informática y la electrónica.

En el segmento de las supercomputadoras, la eficiencia energética de ARM se erige como un diferenciador clave. Aunque x86 puede realizar cálculos similares con menos instrucciones, la verdadera fuerza de ARM reside en su capacidad para la paralelización, ofreciendo más hilos de ejecución eficientes en términos de energía. Desde su origen en ordenadores personales hasta su actual posición dominante en dispositivos móviles y su incursión exitosa en supercomputadoras, la arquitectura ARM destaca por su versatilidad, eficiencia y adaptabilidad a una diversidad de aplicaciones y entornos de uso.

2.4.1 H3 Quad-core Cortex-A7

El procesador H3 Quad-core Cortex-A7 se posiciona como el sucesor del Cortex-A8, ofreciendo un rendimiento mejorado y una arquitectura más simple.

Este procesador forma parte de la arquitectura big.LITTLE, donde combina núcleos Cortex-A7 con núcleos Cortex-A15 en un sistema heterogéneo. Compatible con el Cortex-A15, el ARM Cortex-A7 del Orange Pi presenta características clave:

- **Microarquitectura de doble emisión parcial:** Este diseño permite que ciertas instrucciones se ejecuten más eficientemente al permitir la emisión de dos instrucciones por ciclo de reloj en determinadas circunstancias.
- **Pipeline de 8 etapas:** El pipeline es la secuencia de procesamiento de una instrucción. Con 8 etapas, se mejora la eficiencia del procesador al dividir la ejecución de instrucciones en pasos más pequeños.
- **Conjunto de instrucciones NEON SIMD:** es una tecnología que acelera operaciones en paralelo mediante instrucciones SIMD (Single Instruction, Multiple Data), beneficiando tareas como el procesamiento de gráficos y señales.
- **Unidad de punto flotante VFPv4:** VFP (Vector Floating Point) es una unidad dedicada al procesamiento de operaciones de punto flotante.
- **Soporte de virtualización por hardware:** permite ejecutar múltiples sistemas operativos o instancias de software de manera aislada.

En términos de rendimiento, el Cortex-A7 supera al Cortex-A5 en un 20% en el rendimiento de un solo hilo, heredando todas las características esenciales de los procesadores Cortex-A15 y Cortex-A17.

2.4.2 Linux sobre Plataformas ARM

En el campo de los sistemas embebidos, originalmente diseñados para el control en tiempo real de sistemas especializados, se ha producido una notable evolución gracias al desarrollo de chipsets, kits de desarrollo y a la comunidad *open source*. Este cambio ha transformado la versatilidad de los sistemas embebidos, permitiéndoles desempeñar diversas funciones en un solo dispositivo. La elección del sistema operativo embebido es fundamental, y las distribuciones de Linux embebido sobresalen como la opción preferida.

Es esencial explorar los beneficios y desventajas de cada sistema operativo. Un ejemplo destacado es ARMBIAN, una distribución basada en Debian optimizada para arquitecturas ARM de 32 bits (armhf). Esta distribución se destaca por ofrecer un entorno de consola estándar, ya sea BASH o ZSH, junto con utilidades provenientes de Debian/Ubuntu, permitiendo una configuración fácil a través de una interfaz guiada. Las imágenes distribuidas están optimizadas para medios de almacenamiento flash, y la distribución se presenta sin *bloatware* ni *spyware*, siendo apta tanto para principiantes como para profesionales.

La seguridad es un aspecto que puede ajustarse mediante un comando (‘armbian-config’) y las imágenes son supervisadas por profesionales dentro de la comunidad. Con conocimientos profundos sobre el hardware, los mantenedores y la comunidad de ARMBIAN aportan más de 30 años de experiencia en Linux y Linux embebido. Esta distribución proporciona un marco de construcción de código abierto para generar interfaces de hardware optimizadas para Linux, siendo una elección clara para proyectos de sistemas embebidos basados en arquitecturas ARM de 32 bits.

2.5 Lenguajes de programación

2.5.1 Lenguaje C

El lenguaje de programación C, concebido por Dennis Ritchie en 1972, figura entre los más antiguos y versátiles. Su aplicación se extiende desde el desarrollo de sistemas operativos, como Linux, hasta una amplia gama de programas. La clave de esta versatilidad radica en su naturaleza de bajo nivel, dando lugar a interactuar directamente con la memoria de la máquina.

En el ámbito de los sistemas embebidos, el lenguaje C desempeña un papel fundamental. Los sistemas embebidos, presentes en dispositivos electromecánicos, se centran normalmente en microcontroladores. En este punto las Capas de Abstracción de Hardware (HALs) resultan particularmente importantes. Las HALs actúan como interfaz entre el hardware y el software, posibilitando la utilización de diferentes modelos de hardware sin necesidad de modificar el software, facilitando así el desarrollo y la portabilidad de las aplicaciones.

Además, C se destaca como un lenguaje orientado a la implementación de sistemas operativos, en particular, Unix. Caracterizado por ser de tipos de datos estáticos y débilmente tipificados, C se sitúa en un punto intermedio entre lenguajes de alto y bajo nivel.

Cuando se trata de aplicaciones de telemetría, el lenguaje C sigue siendo esencial. En el contexto de sistemas de transmisión de información para diversas áreas de trabajo, el desarrollo de sistemas que recopilen datos de ubicaciones remotas y ofrezcan información en tiempo real se vuelve más accesible con la versatilidad del lenguaje C.

Nota: En términos de programación, "débilmente tipificado" significa que el lenguaje proporciona menos restricciones en la manipulación de tipos de datos. En el caso de C, esto implica que se pueden realizar operaciones entre diferentes tipos de datos con menos restricciones o comprobaciones automáticas que en lenguajes fuertemente tipificados. La debilidad en la tipificación brinda más flexibilidad pero requiere mayor atención por parte del desarrollador para evitar errores.

2.5.2 Python en Sistemas Embebidos

Python ha ganado reconocimiento en la comunidad de sistemas embebidos como una suerte de ‘navaja suiza’ capaz de solucionar una variedad de desafíos. Sus aplicaciones en el testing de dispositivos embebidos, especialmente en la integración, son notables. La facilidad de acceso al GPIO, rutinas numéricas de alta calidad como numpy/Scipy, y la sencillez para trabajar con HID y libusb/winusb hacen de Python una herramienta valiosa. Además, su versatilidad se extiende a la generación de tablas de búsqueda estáticas en C.

En el contexto de la integración continua, Python se convierte en una herramienta fundamental. Desde la generación de archivos de configuración binarios hasta la encriptación de firmware durante la construcción de actualizaciones, Python demuestra ser una caja de herramientas para diversas tareas en el desarrollo embebido. La creación de enlaces Python para bibliotecas C permite una interacción más dinámica en un intérprete Python.

Python encuentra su lugar en diversas industrias, destacando en análisis de datos como sustituto de MATLAB. Aunque no es la elección principal para programar directamente los

microcontroladores, es ideal para mostrar datos, entrenar modelos de visión por computadora y realizar pruebas rápidas de algoritmos. Además, su uso se extiende a la creación de software de control y GUIs.

En el ámbito del desarrollo y análisis de datos de sensores, Python, especialmente con numpy, se utiliza para visualizar y analizar datos crudos, así como para prototipar algoritmos de procesamiento de señales de manera eficiente. Este enfoque ahorra tiempo que sería consumido implementando prototipos en C dentro de las restricciones de un MCU. La facilidad de uso con IDEs como VSCode complementa esta versatilidad.

Desde comunicarse con sistemas de adquisición de datos hasta realizar comandos directamente sobre diversos dispositivos, Python facilita operaciones complejas en sistemas embebidos. Su aplicabilidad se destaca en ejemplos prácticos, como el control de fuentes de alimentación, la recolección de datos de equipos de medición y la interacción con interfaces de E/S, demostrando que Python se ha convertido en una herramienta indispensable en el desarrollo y producción de sistemas embebidos.

2.6 Radar Doppler Microondas

2.6.1 Generalidades

2.6.1.1 Introducción, Principios Básicos y Características Generales

La palabra radar es un acrónimo derivado de la frase "**RA**dio **D**etection **A**nd **R**anging" y se aplica a equipos electrónicos diseñados para detectar y rastrear objetos (objetivos) a distancias considerables. El principio básico detrás del radar es simple: se transmiten ráfagas cortas de ondas de radio, estas viajan a la velocidad de la luz, se reflejan en un objetivo y luego regresan como un eco, el cual aporta datos acerca de los objetos cercanos.

El radar aprovecha el principio del eco. Para ilustrar este principio, si se hiciera sonar el silbato de un barco en medio del océano, las ondas sonoras disiparían su energía a medida que viajan hacia afuera y, en algún momento, desaparecerían por completo. Sin embargo, si el silbato suena cerca de un objeto, como un acantilado, algunas ondas sonoras radiadas se reflejarían de vuelta al barco como un eco.

El radar es un dispositivo activo. Utiliza su propia energía de radio para detectar y rastrear el objetivo. No depende de la energía radiada por el objetivo mismo. La capacidad para detectar un objetivo a grandes distancias y ubicar su posición con alta precisión son dos de las principales características del radar.

La forma de la señal electromagnética radiada por el radar depende del tipo de información necesaria sobre el objetivo. Existen diferentes tipos de radar diseñados para aplicaciones específicas. Por ejemplo, el radar de navegación marítima utiliza modulación de pulsos, donde puede determinar la distancia a un objetivo midiendo el tiempo que tarda una ráfaga suficientemente corta de energía de radio en viajar al objetivo y regresar a su origen como un eco reflejado. Por otro lado, el radar Doppler, que utiliza ondas continuas (CW), opera de manera diferente. En el radar Doppler, la frecuencia de la señal reflejada desde un objetivo en movimiento cambia según la velocidad relativa del objetivo. Este cambio en la frecuencia, conocido como desplazamiento Doppler, se utiliza para extraer información sobre la velocidad y dirección del objetivo. Ambos tipos de radar, ya sea de modulación de pulsos o Doppler, hacen uso de antenas direccionales para transmitir la señal y recibir el eco reflejado, lo que permite determinar la dirección o rumbo del objetivo.

2.6.1.2 Radar Doppler

Los radares Doppler, a veces llamados radares de onda continua (CW) debido a su uso de ondas continuas para la transmisión, operan en frecuencias de microondas en el rango de varios gigahertz (GHz). Este tipo de radar se basa en un principio funcional clave que explora el efecto Doppler.

Cuando un radar Doppler transmite ondas continuas hacia un objetivo, la frecuencia de la señal reflejada permanece igual si el objetivo está fijo. Sin embargo, cuando el objetivo está en movimiento, la frecuencia de la señal reflejada experimenta un cambio conocido como desplazamiento Doppler. Este cambio es proporcional a la velocidad relativa del objetivo con respecto al radar.

Suponiendo que un objetivo se mueve hacia el radar. En este caso, la distancia entre el radar y el objetivo disminuye, generando un cambio aparente en la frecuencia de la onda reflejada, conocido como Efecto Doppler. Este cambio Doppler, de aproximadamente del orden de los kHz, es significativo cuando la onda transmitida tiene una frecuencia en el rango de GHz.

Para comprender mejor el efecto Doppler en el contexto del radar, es útil explorar cómo afecta la frecuencia de una onda. Según la teoría, cuando la fuente y el receptor de una onda están en movimiento relativo, se produce un cambio en la frecuencia de la onda, conocido como efecto Doppler.

Michael Parker, en su obra "Digital Signal Processing 101" (2017), proporciona una perspectiva interesante al explicar que este fenómeno no es exclusivo del sonido, sino que también se aplica a las ondas electromagnéticas, como las utilizadas en el radar. Como dato adicional, el efecto Doppler ha sido fundamental para determinar la expansión del universo mediante mediciones precisas de la luz de estrellas en el cielo nocturno (el fenómeno se conoce como *red-shift* o 'corrimiento al rojo').

El cambio de frecuencia Doppler (fd) se puede expresar mediante la siguiente ecuación:

$$fd = 2 * (f_0 / c) = (dR / dt) = 2 * (dR / dt) / \lambda_0 = 2 * (v * \cos(\theta)) / \lambda_0$$

Donde:

- v es la velocidad relativa del objetivo, con un ángulo θ respecto al vector posición R.
- $\lambda_0 = c / f_0$ es la longitud de onda de la señal transmitida.
- f_0 es la frecuencia de la señal de radar transmitida.
- c es la velocidad de la luz, aproximadamente 3 x 10^8 m/s.

Este cambio de frecuencia se utiliza para determinar la velocidad y dirección del objetivo en movimiento.

El número "2", que aparece como constante en la ecuación de la frecuencia Doppler, indica que en la práctica, el efecto Doppler ocurre dos veces en la medición: una vez en el camino en sentido sentido desde el radar hacia el objetivo y luego para la energía reflejada (ya afectada por un cambio Doppler) en el camino de regreso.

2.6.1.3 Bandas de Radiofrecuencia y Aplicaciones

Hay dos grupos de frecuencias de radio asignadas por estándares internacionales para el uso de sistemas de radar. El primer grupo se encuentra en la banda X, que corresponde a una longitud de onda de 3 cm, con un rango de frecuencia entre 9300 y 9500 MHz. El segundo grupo se encuentra en la banda S, con una longitud de onda de 10 cm y un rango de frecuencia de 2900 a 3100 MHz. A veces es más conveniente hablar en términos de longitud de onda en lugar de frecuencia debido a los altos valores asociados con esta última.

Un requisito fundamental del radar es la transmisión y recepción direccionales, que se logra produciendo un haz horizontal estrecho. Para enfocar la energía en un haz estrecho horizontal, la longitud de onda debe estar dentro del rango de unos pocos centímetros.

Si se desea, con un radar convencional, determinar la distancia a un objetivo midiendo el tiempo necesario para que un pulso viaje al objetivo y regrese como un eco reflejado, es necesario que este ciclo se complete antes de que se transmita el pulso inmediatamente siguiente. Esta es la razón por la que los pulsos transmitidos deben estar separados por períodos de tiempo de no-transmisión relativamente largos. De lo contrario, la transmisión ocurriría durante la recepción del eco reflejado del pulso precedente. Utilizando la misma antena tanto para transmitir como para recibir, el eco reflejado relativamente débil sería bloqueado por el pulso transmitido relativamente fuerte.

2.6.1.4 *Bandas I/J (Radares de Banda X y Ku)*

Entre 8 y 12 GHz, la relación de longitud de onda con el tamaño de la antena tiene un valor más favorable. Con antenas relativamente pequeñas, se puede lograr suficiente precisión angular, lo que favorece el uso militar como radar aéreo. Por otro lado, las antenas de sistemas de radar de control de misiles, que son muy grandes en comparación con la longitud de onda, siguen siendo lo suficientemente prácticas como para considerarse desplegables.

Esta banda de frecuencia se utiliza principalmente en aplicaciones civiles y militares para sistemas de radar de navegación marítima. Antenas pequeñas, económicas y de rotación rápida ofrecen rangos suficientes con muy buena precisión. Las antenas se pueden construir como radiadores de ranuras simples o antenas de parche.

Esta banda de frecuencia también es popular para radares de imágenes basados en el espacio o en el aire mediante Radar de Apertura Sintética (SAR), tanto para inteligencia electrónica militar como para cartografía geográfica civil. Una aplicación especial del Radar de Apertura Sintética Inversa (ISAR) es la vigilancia de los océanos para prevenir la contaminación ambiental.

2.6.1.5 *Implementación del radar Doppler*

La implementación del radar Doppler de onda continua implica la utilización de varios componentes. Este tipo de radar utiliza dos antenas: una para transmitir la señal de onda continua y otra para recibir el eco reflejado. Una vez que la señal reflejada es recibida, es enviada a un mezclador, donde se combina con la señal de transmisión original, lo que resulta en la obtención de la frecuencia intermedia (IF) mediante la ecuación $IF = F_{TX} - F_{RX}$. Esta frecuencia intermedia contiene la información sobre el cambio en la frecuencia de la señal reflejada, que está directamente relacionado con la velocidad de los objetos en movimiento.

La Figura 2.6.1 muestra un esquema de la implementación.

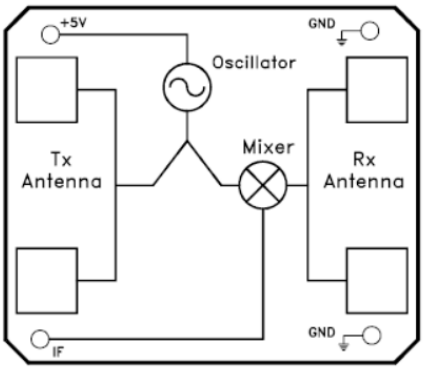

Figura 2.6.1. Esquema de un radar Doppler (HB100).

En la mayoría de los sistemas, las tensiones de salida deben registrarse o leerse de alguna manera tangible y esto se hace generalmente con algún tipo de microcontrolador. La forma más fácil para que un microcontrolador lea datos de un dispositivo analógico es si emite un nivel de tensión de corriente continua. Algunos módulos tienen esta característica incorporada. Para aquellos que no la tienen (como el HB100, véase la Figura 2.6.1), se emite sólo la señal de corriente alterna. Para estos módulos, debe implementarse un circuito de frecuencia a voltaje (FVC, lo inverso a un VCO). Un circuito integrado, como el LM2907N, se puede utilizar para este propósito específico. Este circuito se puede usar para calibrar los datos de salida para un conjunto específico de frecuencias esperadas provenientes del módulo y así contenerlo en el rango de voltaje de referencia. Se necesita el muestreo de estos datos y la manipulación a través de código para tomar estos datos crudos y hacerlos útiles para el sistema.

2.6.2 Uso en metrología legal

La Recomendación OIML R 91 (de la Organización Internacional de Metrología Legal) establece las condiciones que el radar debe cumplir cuando se van a utilizar los resultados de la medición en procedimientos legales. Esta Recomendación establece las condiciones que el radar debe cumplir cuando se van a utilizar los resultados de la medición en procedimientos legales. La interpretación legal de los resultados de las mediciones, la elección de los tipos de radar y las condiciones bajo las cuales se pueden aplicar estos instrumentos se deja en manos de las regulaciones nacionales.

2.6.3 Filtrado Analógico

El filtrado analógico desempeña un papel esencial en la etapa inicial de procesamiento de señales, ya que contribuye a optimizar la calidad de la señal, acorde a su posterior tratamiento digital. Este proceso se basa en la manipulación de componentes analógicos, combinados para lograr modificaciones específicas en la respuesta en frecuencia y la función de transferencia de un circuito.

La atenuación de frecuencias no deseadas mediante el uso de filtros pasivos permite suprimir componentes de la señal que dificultan la detección precisa de la información. Además, el filtrado activo puede realzar ciertas bandas de frecuencia de interés.

Existen diversos métodos de diseño de filtros (entre los más comunes se encuentran el Butterworth, Chebyshev, Cauer y Bessel), cada uno de los cuales ofrece ventajas específicas en términos de características de atenuación y respuesta en frecuencia. La implementación de estos filtros suele llevarse a cabo mediante redes LC, lo que permite ajustar con precisión las características del filtro para satisfacer las necesidades de la aplicación en cuestión.

Los filtros analógicos, además de su papel en la preparación de señales de radar, también se emplean como ecualizadores para corregir distorsiones en la señal y como filtros antialiasing para garantizar una adquisición precisa de datos en aplicaciones de muestreo.

2.6.4 Filtrado Digital

Según Bhaktavatsala (2002), los sistemas de radar modernos utilizan DSP extensivamente para generar y dar forma a los pulsos de transmisión, controlar el patrón del haz de la antena y realizar tareas complejas en el receptor. El proceso de filtrado digital desempeña un papel esencial en el procesamiento de señales, ya que se utiliza para mejorar la calidad de la señal de radar y permitir la extracción de información. Aunque en el ámbito de los radares se aplican diversas técnicas de procesamiento digital, como las utilizadas en SAR (Sintetizador de Apertura o Radar de Apertura Sintética), CFAR (Tasa Constante de Falsa Alarma) y SVS (Sistema de Visión Sintética), para este proyecto resultan de particular importancia resaltar las técnicas específicas empleadas en radares Doppler.

En radares Doppler, uno de los principales desafíos del procesamiento digital radica en abordar cuestiones como el ruido, la interferencia, el desvanecimiento y la ambigüedad. Estos son los factores que impactan la calidad de la señal y complican la detección de objetivos en movimiento. El ruido y la interferencia pueden generar lecturas no deseadas, mientras que el desvanecimiento puede causar fluctuaciones en la señal, lo que dificulta la medición de la velocidad.

En lo que respecta a la implementación de estas técnicas, se recurre a estrategias como el filtrado adaptativo, descrito en detalle por Pineda Márquez, J. L. (2009), que permite ajustar el filtro

en función de las condiciones de la señal. Además, se emplea la detección automática para identificar objetivos en la señal y el umbral adaptativo para mantener una tasa constante de falsas alarmas al descartar señales con amplitudes inferiores a un nivel predefinido.

Como describen Purdy et al. (2000), las técnicas de filtrado y umbrales adaptativos son cruciales para afrontar los desafíos inherentes a la detección de objetivos en movimiento en radares Doppler, ya que son cruciales para afrontar los desafíos inherentes a la detección de objetivos en movimiento. Esto convierte a estas técnicas en herramientas valiosas para aplicaciones como la medición de velocidad y la monitorización del tráfico aéreo y terrestre.

2.6.4.1 Filtrado por Autocorrelación

Se define como señal débil aquella que presenta una relación señal-ruido menor a 0 dB, indicando que el nivel de ruido supera al de la señal. Existe un profundo interés en la adquisición, análisis, procesamiento e identificación de señales catalogadas como débiles. A pesar de contener información valiosa, el procesamiento de estas señales enfrenta el desafío de que se encuentran inmersas en un ruido de fondo significativo.

La relevancia de estas investigaciones se manifiesta en diversas aplicaciones, tales como las comunicaciones a larga distancia o la detección de pequeñas señales periódicas ocultas en el ruido. Abarcan, además, la determinación de las características de transferencia de sistemas lineales en presencia de ruido generado internamente en dichos sistemas.

Fano, R. M. (1951), en su trabajo, demuestra diversas aplicaciones físicas de gran relevancia para las funciones de correlación. La efectividad de estos métodos depende de nuestra capacidad para aproximar experimentalmente la definición matemática de la función de correlación deseada mediante un diseño e implementación adecuados.

El método de correlación ha encontrado aplicaciones en diversos campos. Por ejemplo, se ha explorado su utilidad en la comunicación satelital (Jian-fei et al., 2009). Avramchuk et al. (2013) lo han aplicado con éxito en la detección de fugas en tuberías, mientras que Yuandong et al. (2021) lo emplean para el diagnóstico de fallas en sistemas mecánicos.

Tanto Blu et al. (2003) como Moulin (2006) han investigado métodos para el procesamiento de señales débiles, abordando aspectos como el uso de redes neuronales, análisis de wavelets y teoría de conjuntos difusos.

La autocorrelación se destaca como una técnica que no requiere conocimiento previo de los parámetros de la señal y, finalmente, permite la detección exitosa en condiciones de baja relación señal-ruido (Blu et al., 2003; Moulin, 2006).

La autocorrelación como método de filtrado, podría considerarse entonces, desempeña un papel crucial en el procesamiento de señales y tiene varias aplicaciones prácticas:

- Reconocimiento y detección de patrones periódicos en las señales.
- Procesamiento de Audio: En el audio, la autocorrelación ayuda en tareas como la detección de tonos. Puede determinar la frecuencia fundamental de una nota musical.

- Sincronización de comunicaciones: Ayuda a alinear las señales recibidas con las transmitidas, garantizando una recuperación precisa de los datos.
- Análisis de Series Temporales: La autocorrelación es esencial para analizar datos de series temporales en economía, finanzas y ciencias climáticas. Revela tendencias, estacionalidad y patrones cíclicos.

Salahdine, F. (2018) expresa la función Autocorrelación como

$$R_{xx}(\tau) = \int_{-\infty}^{\infty} f(t) * f(\tau + t)\, dt$$

Donde *Rxx* es la función de autocorrelación, τ es el índice de retardo (*lag*), f(t) es la señal detectada, * indica el complejo conjugado. Este método puede utilizarse en dominio del tiempo o la frecuencia. Al multiplicar por el complejo conjugado, se consideran tanto los valores de amplitud (módulo) como la fase de la señal. Esto es importante en casos donde la fase de la señal puede ser crítica para el análisis, como en la detección de señales moduladas.

Para señales de tiempo discreto, esta integral se convierte en una suma:

$$R_{xx}(\tau) = \sum_{-\infty}^{\infty} x(t) \cdot x(t-\tau)$$

En aplicaciones como la detección de radar, se podría utilizar la autocorrelación como una herramienta para validar la coherencia de las señales recibidas y así capturar las señales que son útiles.

3. Diseño

Es importante destacar que el proceso de diseño e implementación del actual proyecto no sigue un enfoque en cascada con un inicio y un final definidos. En cambio, se caracteriza por ser un proceso iterativo y flexible, donde el diseño y la implementación están estrechamente entrelazados.

En este enfoque, la concepción inicial del sistema proporciona una visión general y un rumbo a seguir. Sin embargo, a medida que se avanza en la implementación y se realizan las pruebas, surgen oportunidades para ajustar y optimizar el diseño. Se trata de un proceso en el que las funcionalidades se incorporan, modifican o eliminan en función de su relación costo/beneficio, pero siempre teniendo en cuenta el objetivo general del proyecto.

Este enfoque permite adaptar el proyecto a las dificultades y cambios que surgen durante el desarrollo del sistema. Al mantener el objetivo general del proyecto como un rumbo y no como una meta estricta, se pueden tomar decisiones informadas (a partir de datos obtenidos de las versiones anteriores) sobre qué ajustes son necesarios para lograr un sistema que cumpla con los estándares deseados.

A pesar de que los capítulos de este trabajo organizan la información de manera separada, el diseño y la implementación del sistema se desarrollaron en un ciclo continuo de evaluación y mejora, donde cada iteración conduce al sistema a cumplir con los requisitos y acercarse al objetivo final.

3.1 Diseño del Sistema

3.1.1 Diagrama General del Sistema Propuesto

En la Figura 3.1.1 se muestra un diagrama visual que presenta de manera general la arquitectura propuesta del sistema, identificando las principales interacciones entre los componentes.

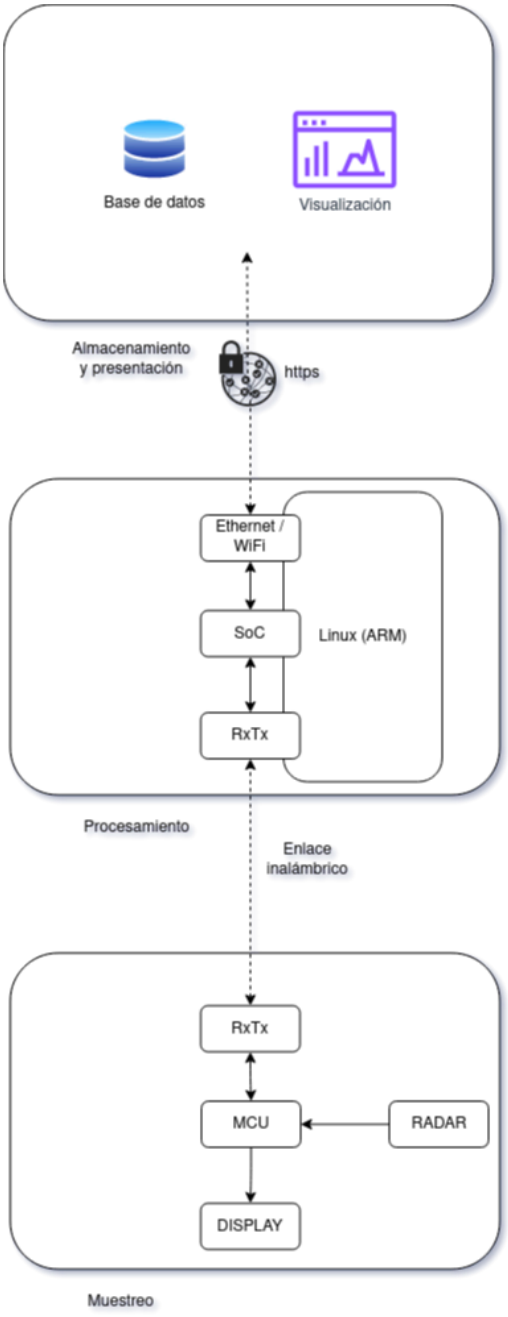

Figura 3.1.1. Mapa conceptual del sistema.

3.1.2 Selección de Dispositivos de Cómputo (MCUs y CPUs)

En el proceso de elegir los microcontroladores (MCUs) y unidades centrales de procesamiento (CPUs) para el sistema, se exploraron diversos criterios y consideraciones.

3.1.2.1 Selección de los microcontroladores

Para la elección de los microcontroladores STM32, se consideraron varias ventajas notables, como se detalla en el capítulo 2.3:

- **Economía y Amplia Adopción:** Los MCU STM32 son económicos y ampliamente adoptados, lo que garantiza la disponibilidad de recursos y la reutilización de código.
- **Compatibilidad ARM:** Utilizan instrucciones estándar de ARM, asegurando compatibilidad con un amplio conjunto de herramientas.
- **Eficiencia en Interrupciones:** Ofrecen interrupciones rápidas.
- **Compatibilidad con Arduino:** Son compatibles con Arduino y cuentan con librerías compartidas.
- **Tecnología Moderna:** Fabricados con tecnología relativamente nueva, garantizando un rendimiento acorde a los estándares actuales.
- **Flexibilidad de Pines:** Permite utilizar diferentes MCUs dentro de la misma familia o entre familias diferentes de STM32, manteniendo la compatibilidad de pines.
- **Documentación**: Los manuales de referencia están redactados de tal forma que facilitan significativamente el desarrollo.
- **Bootloader en la ROM:** Poseen el bootloader en la ROM, permitiendo restaurar el firmware en caso de errores graves sin necesidad de un depurador externo.

Con su conjunto de características, el STM32F103 se presenta como una elección ideal para proyectos que van desde capacidades básicas hasta plataformas integrales. En la Figura 3.1.2 se puede observar que se trata de una placa relativamente simple.

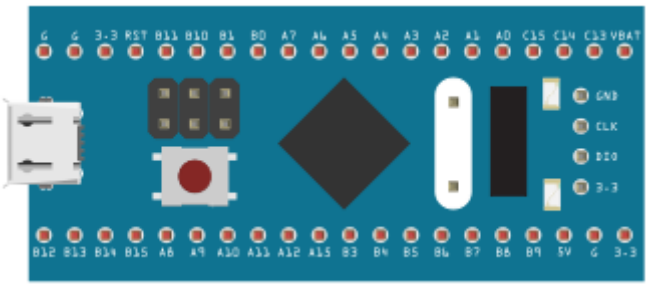

Figura 3.1.2: Diseño del PCB basado en STM32F103, “Blue Pill”.

3.1.2.2 Selección de los CPU para los gateways

Para los gateways, se optó por el Orange Pi de la Orange Pi Foundation. Esta elección se basa en varios factores que hacen del Orange Pi una alternativa atractiva.

En aplicaciones conectadas a una red local (LAN), el Orange Pi puede considerarse superior al Raspberry Pi Zero por diversas razones:

- **Eficiencia Energética y Rendimiento:** El Orange Pi usa el microprocesador H3 Quad-core Cortex-A7 de ARM, ofreciendo una plataforma de hardware acorde a los sistemas de telemetría, como se analizó en el capítulo 2.4. Además, este CPU es ligeramente más rápido que el del Raspberry Pi Zero.

- **Conectividad Confiable:** El uso de Ethernet por cable resulta más confiable que la conexión inalámbrica en un servidor que se encuentra siempre activo y necesita baja latencia en la comunicación de datos. El Orange Pi Zero ofrece un puerto RJ-45, representando una ventaja significativa en estos casos.

- **Puerto USB Estándar:** Brinda un puerto USB estándar, reduciendo la necesidad de adaptadores.

- **Consideraciones:** Aunque la documentación para Raspberry Pi es ampliamente reconocida, la del Orange Pi puede requerir un poco más de esfuerzo para entenderla. Se recomienda utilizar Armbian y seguir las instrucciones de instalación para Linux o Debian genérico para agregar el software adicional necesario.

Inicialmente, el concepto del gateway se pensó para implementarse sobre sistemas con especificaciones más básicas, puntualmente se buscó que corra sobre un sistema basado en un CPU Pentium (el P54C i586 socket 7 a 133 MHz) con Linux x86 y 32 MB de RAM. Sin embargo, debido al mejor soporte para ARM en comparación con microprocesadores sin extensiones SSE, se desarrolló el sistema sobre una plataforma ARM.

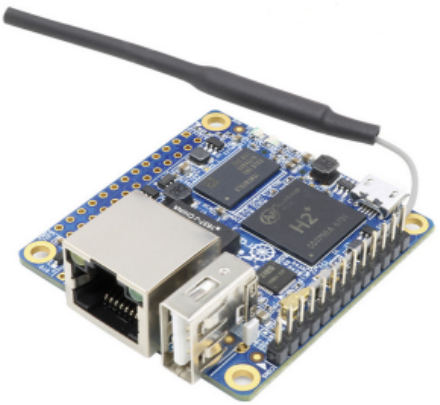

Figura 3.1.3. Orange Pi Zero.

3.1.3 Selección de Sensores y Periféricos

Para la medición de velocidad vehicular existen diversos métodos, los cuales pueden ser inicialmente diferenciados por su nivel de intrusividad. Siendo los métodos más intrusivos los que dependen de la modificación del camino para su instalación, mientras que los menos intrusivos pueden desplegarse a demanda y sin necesidad de modificar la calzada. La Tabla 1 compara algunas de las cualidades de interés de cada sistema.

Método \ Aspecto	Requiere modificar la calzada	Dificultad de despliegue	Durabilidad	Precisión	Costo de despliegue
Loop inductivo	Sí	Complejo	No sufre desgaste	Alta	Alto
Placas piezoeléctricas	Sí	Complejo	Se desgasta lentamente	Alta	Alto
Tubos neumáticos	Sí	Simple	Se desgasta rápidamente	Media	Bajo
Doppler ultrasónico	No	Simple	No sufre desgaste	Depende del estado del tiempo	Bajo
Doppler microondas	No	Simple	No sufre desgaste	Media	Medio
Infrarrojo/Óptico Láser	No	Simple	No sufre desgaste	Media	Medio
Manual (Δd/Δt)	No	Simple	N/A	Baja	Muy Alto

Tabla 3.1.3. Comparativa de métodos de medición del tráfico vehicular.

Considerando la no-intrusividad, la facilidad de despliegue y la resiliencia a las condiciones de lluvia y baja visibilidad de la solución de Radar Microondas de Efecto Doppler, es que se elige este método para la adquisición de muestras para este proyecto. Cabe destacar que el proyecto no contempla el diseño de los sistemas de soporte, como ser la alimentación de los equipos.

Se aborda la elección y adaptación de un radar comercial para el sistema de telemetría en lugar de optar por el radar HB100 con un circuito amplificador. Esta selección está basada en varios factores clave que hacen que el radar comercial sea la opción más adecuada.

El radar elegido (MPH K-15) se presenta en dos variantes: en la banda K y en la banda X, ambos certificados por la NHTSA (Administración Nacional de Seguridad del Tráfico en Carreteras). Estos dispositivos funcionan en modo manual y se alimentan mediante una conexión a la batería del automóvil. Se destacan por operar en modo estacionario y medir el tráfico entrante, además de contar con la capacidad de funcionar en el modo "Instant-On" para ser indetectable cuando está en modo de espera, ya que en modo '*hold*' no emite la señal portadora. Esto permite que el radar se active rápidamente y mida la velocidad de un objetivo en aproximadamente 0.1 segundos para evitar su detección.

Estos radares son del tipo Doppler de Onda Contínua (tratados en la sección 2.6) y ofrecen un tiempo entre muestra y muestra de aproximadamente 250 milisegundos. La direccionalidad está relacionada con el modelo, la versión de banda K tiene un ancho de haz de 15 grados, mientras que la versión de banda X (la cual se utiliza para la Prueba de Concepto) tiene un ancho de haz de 18 grados.

En cuanto al indicador, el radar muestra la velocidad del objetivo en un rango de 15 a 150 km/h en una pantalla de 3 dígitos.

Figura 3.1.4. Radar vehicular MPH K-15

La elección de este radar comercial se basó en su potencial adaptabilidad a las necesidades del proyecto. Es importante destacar que, si bien el radar comercial ofrece un rango de medición de velocidad de 15 a 150 km/h y un tiempo entre muestras de aproximadamente 250 milisegundos, el proyecto realiza la detección y análisis de la frecuencia intermedia de una manera distinta a la que realiza el equipo original. Por lo tanto, las limitaciones específicas del fabricante del dispositivo no se aplican en este proyecto. El radar seleccionado brinda la posibilidad de integrarse en la arquitectura general del sistema, permitiendo el muestreo de la frecuencia intermedia y proporcionando valores de referencia. Cabe mencionar que el aparato incluye un dispositivo calibrador como accesorio de fábrica, el cual se utiliza como referencia para su ajuste a 50 km/h.

Figura 3.1.5. Diapasón de Calibración.

De todos modos, cabe destacar que el sistema se concibe con la posibilidad de conectar otros dispositivos, como el HB100 cuando este es dotado de un amplificador y acondicionando su señal para que pueda ser procesada por el MCU.

Para la prueba de concepto, se eligió un display basado en el IC TM1637. Estos IC se encuentran en *displays* de 7 segmentos económicos. Se comunican con el procesador mediante un protocolo similar a I2C. La implementación es puramente emulación de software y no utiliza hardware especial (aparte de los pines del GPIO).

En el análisis de módulos inalámbricos, se evaluaron diversas opciones, y la elección recayó en el módulo HM-TRP, considerando que se trata de un transceiver FSK transparente, bastante popular y que combina bajo costo con un rendimiento que excede los requerimientos del proyecto. Basado en el IC Si1000 de Silicon Labs, este dispositivo ofrece cierta flexibilidad para configurar parámetros clave, como la capacidad de ajustar la velocidad del puerto serie, la frecuencia de operación (es compatible con frecuencias ISM de 433; 470; 868 y 915 MHz, ofreciendo comunicación half-duplex con modulación FSK), la potencia de salida (con un máximo de 100 mW (20 dBm) y una sensibilidad de -117 dBm), y la velocidad de transmisión (admite velocidades de comunicación de datos en serie desde 1.2 kbps hasta 115.2 kbps). Además, el dispositivo está equipado con una interfaz UART que simplifica la transmisión inalámbrica de datos mediante la provisión directa de información al puerto de entrada, así como una interfaz estándar TTL, que puede extenderse opcionalmente a RS232.

Figura 3.1.6. Módulo UART MH-TRP 915 MHz.

3.1.4 Selección de Software

En cuanto al software utilizado en el sistema, el firmware de los endpoint se desarrolla utilizando las librerías HAL del MCU STM32F103 en STM32CubeIDE en lenguaje C, por conveniencia. Para los gateways, se opta por Debian Linux y librerías de Python para gestionar las comunicaciones del stream binario a través de UART y el caché mediante Redis. Además, utilizando Python, se trabaja con los datos recibidos y se preparan en formato JSON para su envío a la instancia de InfluxDB en la nube.

En el ámbito de las comunicaciones, la adopción de estándares reconocidos, como MQTT sobre LoRaWAN o Zigbee, ofrecería beneficios significativos, como la compatibilidad, la facilidad de interoperabilidad entre dispositivos de distintos fabricantes, la consistencia en la comunicación y la confiabilidad probada en diversos escenarios.

No obstante, la decisión de optar por un protocolo personalizado puede tener justificaciones específicas. Las desventajas asociadas, como la posible incompatibilidad con algunas herramientas, el mayor esfuerzo de desarrollo y posibles problemas de interoperabilidad, deben evaluarse en relación con los requisitos particulares del proyecto. Si bien los estándares ofrecen una solución probada y es mantenida por terceros, un desarrollo personalizado puede ser una elección estratégica en situaciones donde la adaptabilidad y simplicidad son prioritarias, a pesar de la carga adicional de desarrollo y mantenimiento que conlleva. Un ejemplo de esto sería el caso donde la estructura de datos es lo suficientemente simple como para poder prescindir de la mayoría de las características de un protocolo completo, como sería el caso de tener que enviar una respuesta (un ACK que no sea crítico o un *heartbeat*) de 1 byte. En estas situaciones, es probable que no sea necesario adoptar un **protocolo** con control de flujo y paridad, preámbulos, seguridad y CRC, ya que podrían convertirse en un *overhead* sin sentido. También, podría darse el caso donde los datos no puedan formatearse fácilmente para adaptarlos a las tramas que se van a utilizar, o bien, que se requiera un esfuerzo adicional que no esté previsto realizar.

En cuanto a los gateways, se ha seleccionado Debian Linux como sistema operativo base, ofreciendo mayor flexibilidad y versatilidad en el entorno de desarrollo. Se ha diseñado el software para ser compatible con intérpretes de Python tanto en arquitecturas x86 como ARM, destacando la portabilidad del código.

El software de los gateways utiliza librerías de Python para facilitar la gestión de comunicaciones UART y la administración del caché Redis. Además, se realiza un procesamiento de los datos en formato JSON para su posterior envío y almacenamiento en la base de datos de InfluxDB en la nube. Esta elección se basa en la simplicidad de implementación y que el desarrollo sea ágil y efectivo. Cabe mencionar que se decide utilizar un software compatible con la nube debido a las ventajas en escalabilidad y disponibilidad que provee en comparación con soluciones *self-hosted*.

3.1.5 Diseño del Endpoint

Las decisiones de diseño del "endpoint" se realizan principalmente teniendo en cuenta las limitaciones del hardware. Ya que se trata del punto donde se censan los datos para su posterior procesamiento, se prefiere priorizar la eficiencia en el uso de recursos.

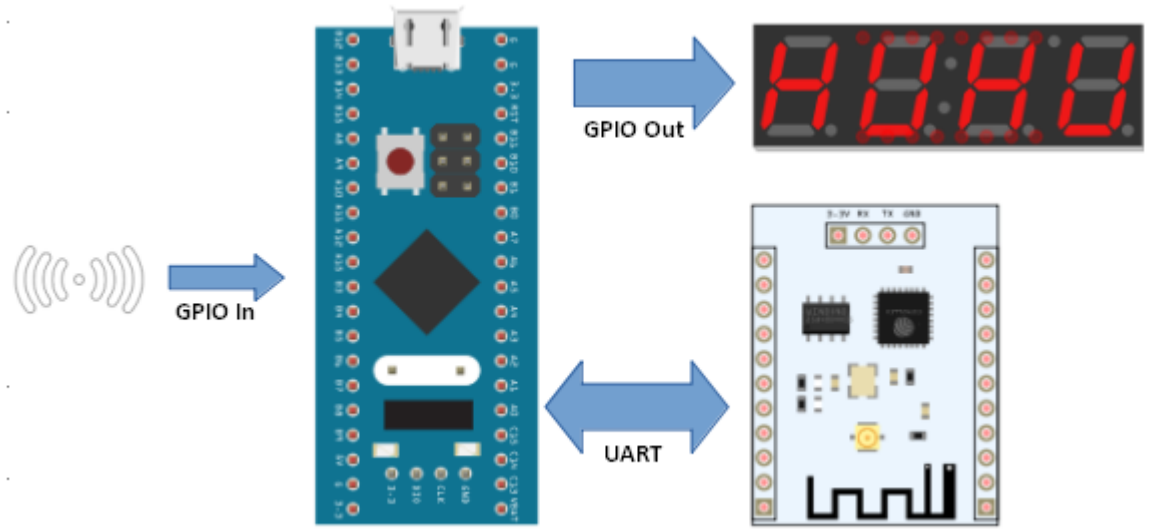

Figura 3.1.7: Elementos e interfaces del Endpoint.

Se pretende que el GPIO sea capaz de detectar la señal del radar para poder procesarla. Para esto se necesita adecuar los niveles de tensión de entrada y optimizar los tiempos en que el sistema se encuentra ocupado, ya que se dispone de un solo hilo de ejecución. Por ejemplo, sería óptimo no ocupar tiempo en comprobar el estado del buffer de recepción del puerto serie y utilizar interrupciones para procesar lo recibido. La recepción de tramas se utilizará para enviar los datos de velocidad (ya procesados por el gateway) al display LCD, ya que al gateway se envía un dato sobre un periodo de tiempo.

En esta primera versión del sistema, no es necesario identificar cada endpoint como un dispositivo único para su funcionamiento. Sin embargo, se contempla la posibilidad de que coexistan dos o más dispositivos por puerta de enlace y por *dashboard*, por lo que se prefiere obtener un identificador único para cada estación.

Además, se prevé la capacidad de incluir un bit de paridad en la trama, dentro del byte de control. También es posible utilizar los bytes no utilizados de la trama para incluir una suma de verificación como método adicional para comprobar la integridad de los datos recibidos.

3.1.6 Diseño del Gateway

El diseño del gateway se guía por la eficiencia en costos iniciales, la facilidad de despliegue y la optimización del rendimiento del hardware utilizado. Se enfoca en la portabilidad, utilizando estándares populares y probados.

Dado que se emplea un *transceiver* basado en Si-1000 para las comunicaciones de RF, el objetivo es adquirir datos desde el puerto serie, procesar las tramas recibidas y, por un lado, enviar acuses de recibo (ACK) al mismo puerto serie y, por otro lado, almacenar la información en la base de datos de InfluxDB.

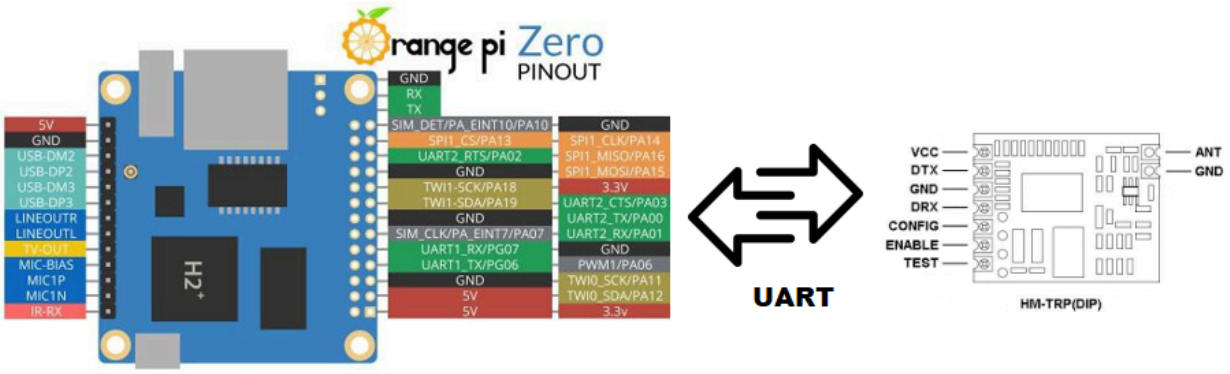

Figura 3.1.8: Pinout del Orange Pi Zero y el transceiver.

La elección definitiva recae en las placas Orange Pi, ejecutando un sistema operativo Debian compilado para arquitectura ARM, como se detalla en la sección 2.4. Estas placas se complementan con los transceivers antes mencionados para la comunicación inalámbrica serie. El flujo de trabajo implica la lectura de datos desde el buffer de recepción del puerto serie, la interpretación de la trama para determinar cómo manejar los datos del *payload* y, finalmente, la emisión de un ACK y el envío de la información a la base de datos definitiva.

La trama de acuse de recibo se utiliza para transmitir los datos ya procesados desde el *gateway* hacia cada estación. El dato está en un byte en particular y si el ID de dispositivo recibido en la trama coincide con el identificador propio del endpoint, entonces se presenta el entero en el display. Este método permite liberar al microcontrolador de pasos adicionales para convertir los valores de tiempo en la velocidad que se mostrará y permite aplicar correcciones directamente en el procesamiento de los datos sin tener que recurrir a flashear el firmware de cada estación.

3.1.7 Diseño de la Aplicación Web

La aplicación web se concibe como la principal interfaz para acceder a los datos históricos del sistema, posibilitando a los usuarios explorar y comprender la información generada. La prioridad en el diseño de esta aplicación radica en la visualización clara y comprensible de las series temporales.

Como se analizó en la sección 2.2.2, la elección de InfluxDB Cloud como plataforma para las visualizaciones se basa en su orientación de diseño para almacenar series temporales y la Web UI basada en una versión modificada de Grafana, lo que permite la interacción con los datos a través de una interfaz gráfica que corre en un navegador web, por lo tanto, es independiente del sistema operativo del dispositivo cliente. Este enfoque se alinea con la estrategia de utilizar estándares consolidados y probados en la industria. La interfaz gráfica desempeña un papel crítico al facilitar la interpretación de datos y al influir significativamente en la experiencia del usuario.

Esta decisión de diseño enfatiza la importancia de la interfaz gráfica en la capa de aplicación y su conexión con la experiencia del usuario.

4. Implementación

4.1 Enfoque y metodología

En la fase de implementación, se adopta un enfoque pragmático al combinar elementos de los métodos 'top down' y 'bottom up' (como se analizó en la sección 1.6) para materializar las metas establecidas. La concepción general del sistema se inició mediante una visión global, para luego seleccionar los componentes individuales necesarios. Posteriormente, se integran gradualmente estos componentes para construir un sistema completo. Esta combinación de estrategias permite adaptarse a los requerimientos y limitaciones de la implementación, garantizando una construcción coherente del sistema. Esta metodología se ajusta a la complejidad del proyecto, considerando factores como recursos disponibles y conocimientos técnicos.

4.2 Implementación del Endpoint

Los pines están habilitados de acuerdo con los requerimientos del diseño. El pin LED (PC13) se utiliza simplemente como indicador de actividad, proporcionando retroalimentación visual sobre el estado del dispositivo. El pin RADAR (PB8) se configura como una entrada digital para la recepción de señal del radar de onda contínua. Para las comunicaciones, se utiliza USART1_RX (PA10) y USART1_TX (PA9) a una velocidad de 9600 baudios. CLK_PIN (PB15) y DIO_PIN (PB14) se emplean para controlar el display. SYS_ITCK-SWCLK (PA14) y SYS_ITMS-SWDIO (PA13) están relacionados con el depurador STLINK. Se utilizan etiquetas personalizadas para simplificar la referencia a los pines en el código (por ejemplo,'LED' en lugar de 'GPIOC_13'). En la Figura 4.2.1 se muestra la configuración general de los pines MCU.

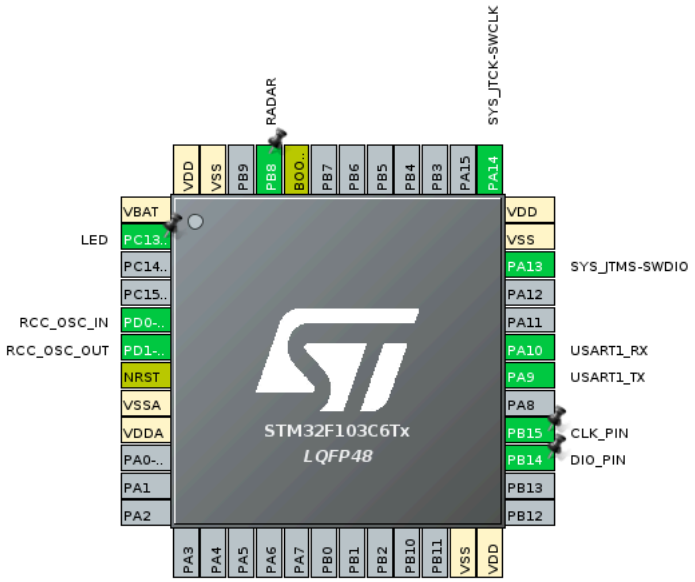

Figura 4.2.1. Configuración del microcontrolador.

4.2.1 Implementación del contador de frecuencia y filtrado

Se hizo uso de una metodología heurística* en una de las primeras etapas de procesamiento y filtrado de la señal digital. Tras evaluar diversas opciones para filtrar lecturas no deseadas, se optó por implementar el principio de autocorrelación de señales discretas, como se detalla en el capítulo 2.6.4. Este enfoque permite filtrar lecturas asociadas al ruido interno del radar cuando está en reposo, asegurando resultados adecuados. La elección de esta metodología no solo filtra las lecturas aleatorias, sino que también permite ajustar la sensibilidad del filtro al modificar la "profundidad" del autocorrelador.

** Cabe destacar que "metodología heurística" no es simplemente un eufemismo para "prueba y error", sino que se refiere a usar técnicas que si bien puede que no garanticen una solución óptima, al menos para estos casos son efectivas: se diseña, se implementa, se analizan los resultados y se rediseña hasta conseguir una solución satisfactoria.*

La información sobre la velocidad del objetivo está codificada en la frecuencia de salida del mezclador de RF, es la resultante de la mezcla entre la portadora microondas emitida por el aparato y el rebote Doppler que recibe. En esta instancia entonces se pretende obtener un valor entero asociado a la frecuencia de entrada. La figura 4.2.2 muestra el comportamiento de una FM en un gráfico v(t) cuando contiene algún tipo de información.

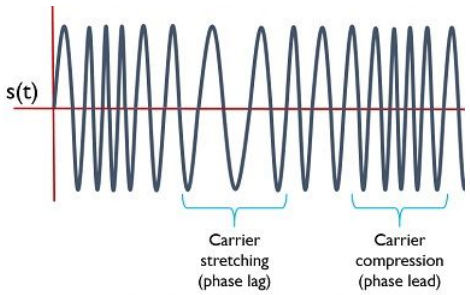

Figura 4.2.2. Modulación FM

Dado que el MPH K-15 (Figura 3.1.4) no cuenta con una salida ‘auxiliar’ para tomar la señal desde un dispositivo externo, para el actual proyecto se agregó, modificando ligeramente el circuito interno del radar, un pin de salida en la señal de frecuencia intermedia para así poder aprovechar la señal en la implementación.

La señal de entrada (o sea la salida del radar) tiene propiedades de una señal modulada en frecuencia (no estrictamente una FM) y procesada por un comparador (Figura 4.2.3). Se interpreta que el Δf del rebote Doppler se encuentra contenido en esta señal.

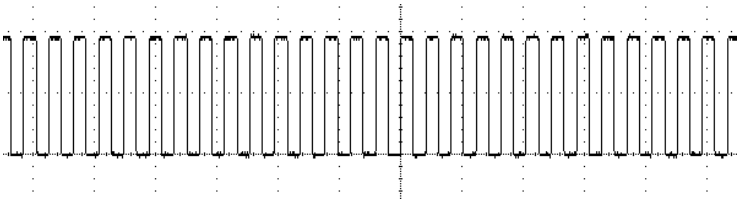

Figura 4.2.3. Salida del Radar usando el calibrador.

4.2.1.1 Contador de frecuencia

Para medir la frecuencia en el pin de entrada, se utiliza el método de contar los pulsos completos de la señal. En este enfoque, se emplea la lectura de un pin de entrada digital del STM32 para, con dos bucles *while()* dentro del bucle principal, incrementar un contador cada vez que el pin de entrada completa un ciclo. En la figura 4.2.4 se puede visualizar conceptualmente el muestreo de la señal. La función ReadPin() consulta constantemente el estado de la entrada y cada vez que sucede que pasa de un nivel alto a uno bajo y vuelve a aparecer un nivel alto, se incrementa un contador. Este concepto se puede ver implementado en C en el código Cod 4.2.1.0.

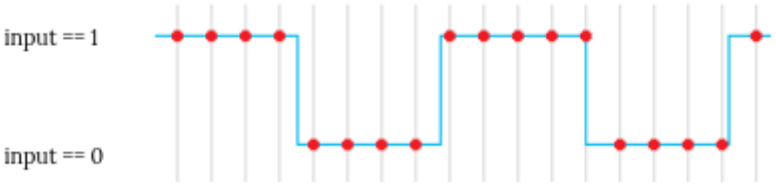

Figura 4.2.4. Muestreo de la señal de entrada.

```
01      while(1){
02              while (input == 1);
03              while (input == 0);
04              i++;
05      }
```

Cod. 4.2.1.0

Este bucle se repite para siempre, por lo cual el contador simplemente se incrementaría hasta desbordar la variable. Para un contador de frecuencia, resultan útiles únicamente los ciclos por unidad de tiempo (por ejemplo, por segundo). Se configura entonces el temporizador por hardware 'TIM1' para que se produzca una llamada cada 1 segundo e interrumpa este bucle.

Siguiendo el código Cod 4.2.1.1, se usa "i" como una variable global, la llamada (*callback*) de la interrupción que se genera cada segundo dispara la lectura esa variable y se reinicia el contador antes de salir de la interrupción y repetir el ciclo. De esta manera, la variable "i" cuenta la cantidad de flancos ascendentes por segundo (la frecuencia de la señal).

Un aspecto que debe considerarse para estas aplicaciones es la velocidad de muestreo: cada consulta al GPIO se lleva a cabo aproximadamente cada 1,65 µs. Esto se calculó usando un método muy simple, detallado en el fragmento de código Cod. 4.2.1.1, en el cual se toma una lectura del SysTick como referencia (línea 01) y se compara con la lectura (línea 05) que se obtiene luego de realizar un número arbitrario de consultas a un pin digital del GPIO (bucle de la línea 02).

```
01      refTick = HAL_GetTick();
02      for (i=0; i<32768; i++){
03              HAL_GPIO_ReadPin(GPIOB, RADAR_Pin);
04      }
05      benchmarkResult = (HAL_GetTick() - refTick);
```

Cod. 4.2.1.1

Esto dio como resultado que, por ejemplo, 32768 lecturas (este número es arbitrario) se llevan a cabo en 54 ms (de ahí se deducen los 1,65 µs por acceso). Dado que este tiempo de espera es relativamente corto frente a una señal de entrada del orden de los kilohertz, el uso de lecturas de pines del GPIO utilizando bucles "*while*" puede considerarse adecuado para efectuar la medición.

4.2.1.2 Filtrado

El método mencionado funciona bien para medir la frecuencia de una señal, pero no discrimina el ruido. Sucede que mientras el radar no está midiendo a un objetivo móvil, la señal de salida del radar (la FI) aparece modulada por ruido, tal como se muestra en la Figura 4.2.5. Se estima que probablemente exista un control de ganancia o una ganancia elevada, ya sea en la frecuencia intermedia a la salida del mezclador o al ser esta salida procesada por un comparador. Entonces sucede que el mismo ruido interno del aparato modula la salida y por ende el bucle contador de frecuencia del microcontrolador se incrementa aleatoriamente. En tal caso la variable “i” devuelve estos valores aleatorios (lo cual podría ser útil si uno buscara, por ejemplo, alimentar un generador de números aleatorios, pero no es el caso del proyecto).

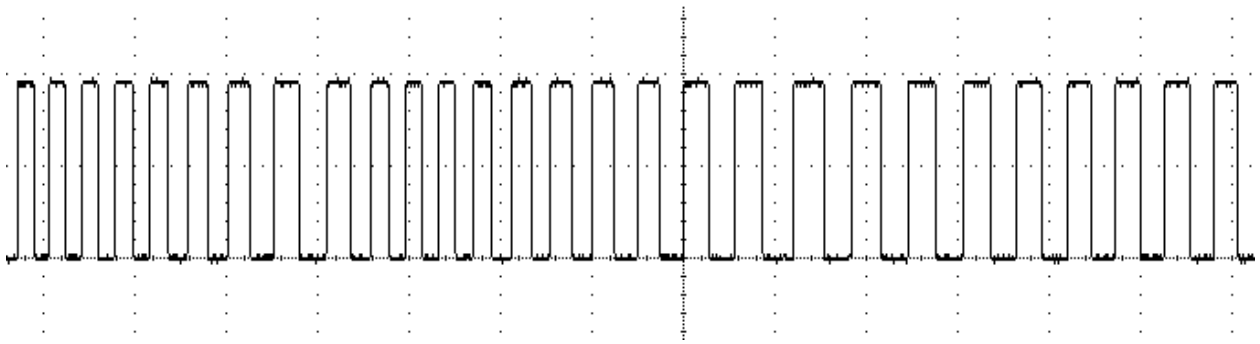

Figura 4.2.5. Señal del Radar en reposo.

En base a varias mediciones y análisis, se pudo observar una característica importante de la salida del radar que se captura en el pin de entrada: los ciclos de la señal cuando la misma se genera a partir de una lectura genuina del aparato, son relativamente uniformes o varían poco en el tiempo. Esta cualidad se puede visualizar comparando las figuras 4.2.3 contra la 4.2.5.

Entonces, teniendo en cuenta que:

- En ausencia de objetivos en movimiento, el radar devuelve una señal que es una FM modulada por ruido y por lo tanto la longitud de onda de la señal varía constantemente y de forma aleatoria.
- Mientras el radar recibe la señal de rebote Doppler con su Δf, la señal de frecuencia intermedia de salida del mezclador tiene una longitud de onda relativamente estable.

Se implementa un autocorrelador de medias-longitudes de onda de la siguiente manera:

- Se establece una “profundidad de autocorrelación”. Esto es, un valor entero mayor que 1, que viene a determinar la cantidad de semiciclos que se van a comparar.
- Se guardan tanto el valor máximo como el valor mínimo que se obtuvieron en las muestras, las cuales se obtienen integrando la media onda. Véase figura 4.2.6.
- Se establece además un retardo arbitrario para hacer saltos entre muestras (se estableció 20 ms, pero podría ser más, menos o incluso hacerlo adaptativo).
- Por último, se determina si la máxima diferencia entre el semiciclo muestreado más corto y el más largo se encuentra dentro de cierta tolerancia arbitraria.

El código Cod 4.2.1.2 realiza la medición del ancho de los pulsos del radar. En el bucle *for*, se espera un tiempo de 20 ms para el próximo pulso y luego se mide su ancho. Se utiliza un *array* nthPulseWidth para almacenar el ancho de cada pulso. Después de medir todos los pulsos, se calcula el *jitter*, que es la diferencia entre el ancho máximo y el ancho mínimo de los pulsos. Esto se calcula como maxPulseWidth - minPulseWidth.

```
1. for (i = 0; i < filteringDepth; i++) {
2.     HAL_Delay(20);
3.     while (HAL_GPIO_ReadPin(GPIOB, RADAR_Pin));
4.     while (!HAL_GPIO_ReadPin(GPIOB, RADAR_Pin));
5.     while (HAL_GPIO_ReadPin(GPIOB, RADAR_Pin)) {
6.             nthPulseWidth[i]++;
7.     }
8.     if (i == 0) {
9.             maxPulseWidth = nthPulseWidth[0];
10.      minPulseWidth = nthPulseWidth[0];
11. }
12. if (nthPulseWidth[i] > maxPulseWidth) {
13.      maxPulseWidth = nthPulseWidth[i];
14. }
15. if (nthPulseWidth[i] < minPulseWidth) {
16.      minPulseWidth = nthPulseWidth[i];
17. }
18. }
19. jitter = maxPulseWidth - minPulseWidth;
```

Cod. 4.2.1.2

La lógica de incrementar un contador mientras un nivel se mantiene es la del integrador digital. En la Figura 4.2.6 se visualiza conceptualmente cómo se incrementa un valor mientras el nivel de la señal muestreada se mantenga. Existen varias técnicas para realizar este procedimiento y esto da lugar a posibles optimizaciones. Muestreando y midiendo de esta manera se puede determinar si la señal es lo suficientemente estable para considerarse una lectura válida.

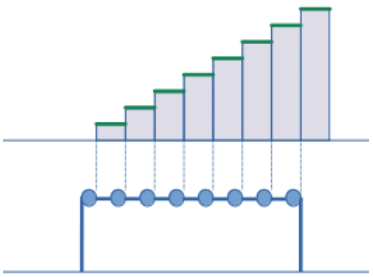

Figura 4.2.6. Muestreo y sumatoria de un semiciclo de la señal de entrada.

Entonces si `jitter < jitterCutoff`, se utiliza `HAL_GetTick()` para obtener una referencia y poder tomar el tiempo que lleva contar un número arbitrario de pulsos. En el fragmento de código Cod 4.2.1.3 se realiza el cálculo del período (o la frecuencia) de la onda completa del radar. Si el jitter es menor que un valor de corte definido por `jitterCutoff`, se realiza la medición del tiempo que demoran varios pulsos. Para esto, se utiliza un bucle que cuenta el tiempo entre un número específico de pulsos. Luego, se calcula el período normalizado, teniendo en cuenta que `filteringDepth` multiplica por dos porque se supone que la onda es simétrica. Si el jitter es mayor que el valor de corte, el período se establece en cero.

```
1.  if (jitter < jitterCutoff) {
2.      refTick = HAL_GetTick();
3.
4.      for (i = 0; i < (30 * filteringDepth); i++) {
5.        while (HAL_GPIO_ReadPin(GPIOB, RADAR_Pin));
6.        while (!HAL_GPIO_ReadPin(GPIOB, RADAR_Pin));
7.      }
8.
9.      deltaSystick = (HAL_GetTick() - refTick);
10.     normalizedInputPeriodTime = 2 * deltaSystick / filteringDepth;
11. }
12.
13. else {
14.      normalizedInputPeriodTime = 0;
15. }
```

Cod. 4.2.1.3

La utilización de la constante 30 en el bucle de conteo de pulsos se basa en la necesidad de optimizar la precisión de las mediciones, considerando las limitaciones de la trama de comunicaciones (la variable de 8 bits de precisión) y la resolución lograda en la medición. Esta elección se realiza con el fin de obtener un equilibrio entre la sensibilidad del sistema y la capacidad de procesar, almacenar y transmitir los resultados.

Inicialmente, se exploraron diferentes valores para esta constante, evaluando su impacto en la resolución de las mediciones y la estabilidad del bucle. Se observó que valores demasiado bajos resultaban en una menor sensibilidad a las variaciones de la señal de entrada, mientras que valores demasiado altos podían conducir al desbordamiento de la variable de almacenamiento o a la falta de tiempo de ejecución del bucle de conteo.

Al ajustar la constante a un valor de 30, se logró mantener un nivel aceptable de sensibilidad, permitiendo capturar las variaciones de la señal con suficiente precisión, mientras se evitaban problemas de desbordamiento y el bucle operaba dentro de los tiempos disponibles para realizar la medición.

Además de la estrategia actual de medir un período de tiempo para una cantidad fija de pulsos, existe la posibilidad de implementar un enfoque alternativo: medir los pulsos dentro de intervalos de tiempo predefinidos. Esta metodología tal vez permitiría obtener una mayor flexibilidad en la frecuencia de muestreo, ya que en lugar de esperar un tiempo indefinido para recopilar un número específico de pulsos, el sistema podría configurarse para realizar mediciones a intervalos regulares de tiempo. Por ejemplo, se podría establecer la detección y conteo de pulsos cada cierto periodo de tiempo y trabajar con el filtrado por autocorrelación sobre esta matriz de resultados. Esta modificación proporcionaría una forma más directa de controlar la frecuencia de muestreo y adaptarla a las necesidades específicas de la aplicación, lo que podría resultar en una mayor precisión y eficiencia en la adquisición de datos.

El valor de `normalizedInputPeriodTime` es almacenado en una variable global y cada segundo un temporizador por hardware dispara una llamada (*callback*) que envía, insertando en la trama de comunicaciones, el último tiempo `normalizedInputPeriodTime` obtenido.

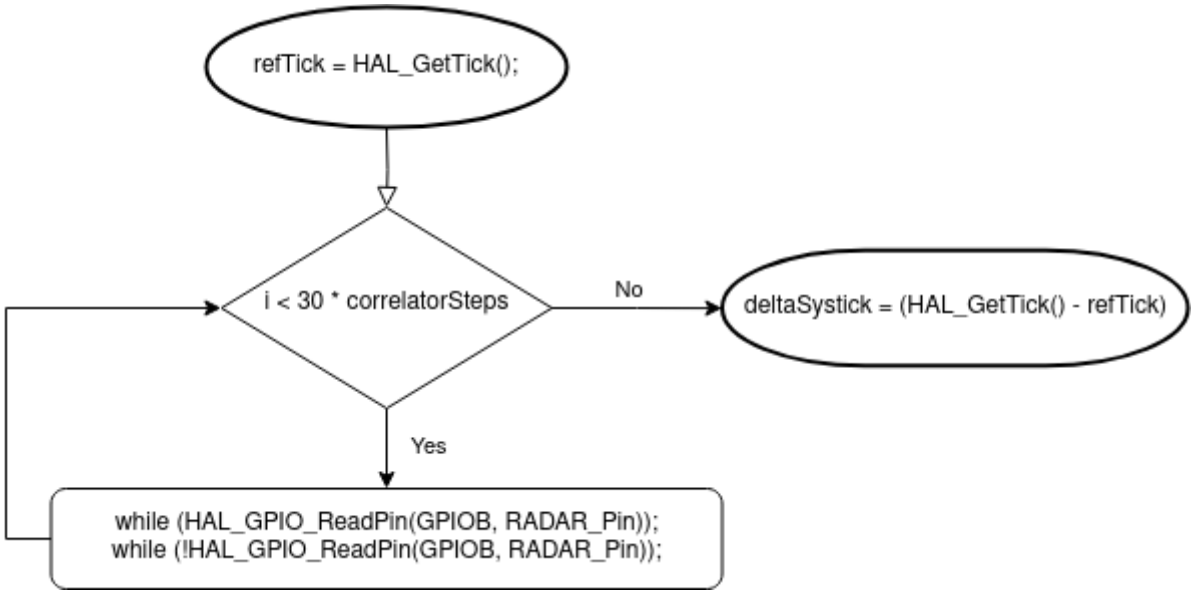

Figura 4.2.7: Determinación de periodos.

Como puede suponerse, existe una velocidad o frecuencia de Nyquist, como límite “duro” del sistema, ya que la velocidad de muestreo depende de la velocidad de ejecución del programa y está relacionada a los ciclos de CPU requeridos para sondear el pin de entrada. Otra limitación es que el sistema requiere, en la señal, que los valores altos y bajos en la onda cuadrada sean identificables por la entrada digital. Según la documentación, en la tabla 69 de la hoja de datos del STM32F103, se muestran los valores típicos y máximos de las tensiones de entrada baja (VI_L) y alta (VI_H) para los diferentes niveles de alimentación, excepto para el pin BOOT0. Por ejemplo, si el nivel de alimentación es VDD = 3.3 V, entonces la tensión de entrada baja máxima (para obtener un 0 lógico) es VI_L = 1.23 V y la tensión de entrada alta mínima (para obtener un 1 lógico) es VI_H = 1.88 V. Esto significa que si se aplica una tensión menor o igual a 1.23 V a un pin del GPIO, se leerá “0”, y si se aplica una tensión mayor o igual a 1.88 V, se leerá “1”. El fabricante no garantiza resultados para valores de tensión intermedios.

4.2.2 Implementación de las comunicaciones

Los endpoints inician la transmisión de tramas automáticamente, con una interrupción del timer interno del STM32F103, “TIM1”, cada 1 segundo. Para transmitir datos entre los dispositivos se usa una trama de 12 bytes (uint8_t frame[12]).

La trama se interpreta de esta manera:

```
   x    |   x    x    x    x   |   x    x    x    x   |   x    x    x
[Ctrl] | [pA1][pA2][pA3][pA4] | [pB1][pB2][pB3][pB4] | [pC1][pC2][pC3]
```

- 1 byte de control
- 4 bytes de payload A
- 4 bytes de payload B
- 3 bytes de payload C

En la implementación de las comunicaciones, se optó por utilizar DMA (Acceso Directo a Memoria) en la recepción de datos en lugar del método de preguntar constantemente (*polling*). Esto se logra mediante el uso de una llamada del búfer de recepción del UART, que se activa cuando el búfer se llena. En este *callback*, se realiza la lectura del búfer de recepción y se procesa la trama de datos entrante. Esta decisión de diseño permite una gestión más eficiente de los recursos de CPU al evitar la necesidad de ciclos continuos de consultas (*polling*) para verificar la llegada de datos. De esta manera, se optimiza el rendimiento del sistema al liberar la CPU para otras tareas mientras se espera la recepción de datos.

Si bien el payload es genérico, según el byte de control, los payload pA; pB y pC adquieren distintas funciones. En cada dispositivo hay un *switch-case* que procesa este byte de control y, dependiendo del contenido, decide cómo procesar la carga de la trama.

case 0x00 (trama de control o retorno):

Esta trama proviene del gateway, se utiliza para los ACK. Al recibirla, solamente hay que probar a qué dispositivo iba dirigida. Para esto, solamente es necesario comparar los 32 bits del payload B (pB) con el identificador único (devID) del dispositivo receptor.

```
 00 00 |  x    x    x    x  | id0  id1  id2  id3 |  x    x    x
[Ctrl] |[pA1][pA2][pA3][pA4]|[pB1][pB2][pB3][pB4]|[pC1][pC2][pC3]
```

case 0x10 (trama de datos):

Esta trama contiene datos de sensores. Por lo tanto, va dirigida a un gateway y se ignora en el resto de los endpoints.

```
 01 00 |  x    x    x    x  | id0  id1  id2  id3 |  x    x    x
[Ctrl] |[pA1][pA2][pA3][pA4]|[pB1][pB2][pB3][pB4]|[pC1][pC2][pC3]
```

Mediciones de velocidad: Para mostrar la velocidad en el display se decidió usar los acuses de recibo, en las tramas ACK se envía lo que el gateway recibió de algún dispositivo. Si el ID de la trama de ACK (payload B) coincide con el hwID del propio aparato que está recibiendo la trama, entonces significa que el gateway recibió y confirmó la lectura que se le envió momentos antes. El valor entero que se mostrará en el display corresponde al byte 0 del 'Payload A'.

Identificación: Se decidió hacer uso de un registro interno del MCU con el número de serie, para identificar cada dispositivo. De esta forma, cada trama posee un identificador único de 32 bits que indica a qué endpoint corresponde.

4.2.3 Display

La interfaz del TM1637 se basa en un protocolo simple, como se vio en la sección 3.1.3, que utiliza conmutación de GPIO. La librería está basada en la HAL de STM32, y puede integrarse con diferentes microcontroladores, con el fin de manejar únicamente el protocolo de bajo nivel del TM1637 modificando las funciones de configuración/reset de GPIO.

- La librería cuenta con una función de mapeo ASCII a 7 segmentos, facilitando la programación de este display.
- La librería, según los desarrolladores, se encuentra en una etapa 'beta'.

El TM1637 utiliza dos pines principales:

- CLK: Pin de entrada para la señal de reloj.
- DIO: Pin de Entrada/Salida de Datos.

El módulo se conecta a una fuente de alimentación de 3,3 a 5 V (VCC) y tiene un pin de masa (GND). Los pines CLK y DIO pueden conectarse a cualquier pin digital, lo que permite la conexión de múltiples módulos siempre que cada instancia tenga su propio par de pines. Al escribir el código, solo se necesita definir el par de pines en STM32CubeIDE y luego utilizarlos en el proyecto.

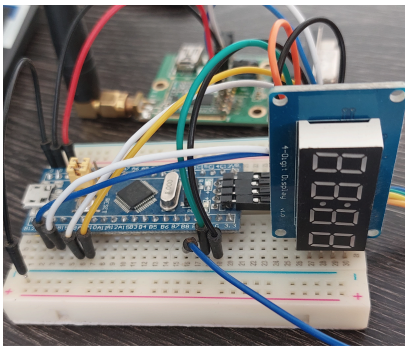

Figura 4.2.8: Implementación del *endpoint*.

4.2.4 Acondicionamiento de la entrada de señal del radar.

Se implementó un divisor resistivo con dos resistencias de 1kOhm, lo cual es una solución simple y a la vez efectiva para reducir la tensión de la señal que se obtiene del radar, la cual ronda los 9V durante el nivel alto de la señal. El uso de un pin tolerante a 5V del MCU asegura que la señal resultante del divisor resistivo se mantenga dentro de límites seguros, con una tensión máxima entre 2 y 5V como es requerido. Este proceso garantiza una adecuada interfaz entre el radar en sí y el microcontrolador, evitando daños y/o lecturas erróneas por incompatibilidad de niveles de tensión.

4.3 Implementación del Gateway

La aplicación en Python emplea el módulo `serial` para la comunicación UART, conectándose al puerto `/dev/ttyS1` con una velocidad de transmisión estándar de 9600 baudios y 8 bits por baudio. La elección del puerto y la velocidad depende de la configuración específica del sistema operativo y el hardware utilizado.

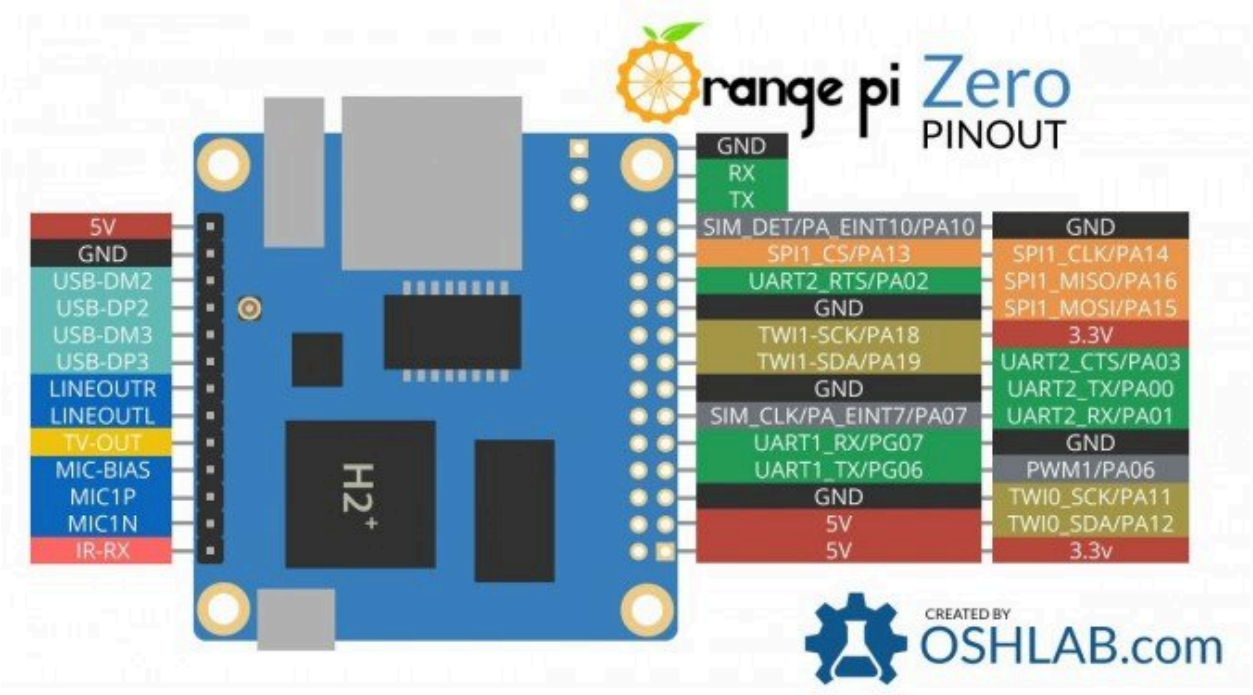

Figura 4.3.1. Pinout Orange Pi Zero.

Como se puede observar en el Anexo C, para realizar la lectura de datos del buffer de recepción del puerto serie, se espera la llegada de datos en paquetes de 12 bytes. Se interpretan y procesan estos datos byte a byte. El control de flujo se basa en el valor del primer byte del paquete. Siguiendo lo expuesto en la sección 4.2.2, los paquetes con un byte de control en `0x10` se consideran provenientes de estaciones, conteniendo datos de sensores. Estos datos se almacenan temporalmente en una base de datos Redis utilizando un *timestamp* como clave. La implementación originalmente utilizaba un contador incremental, pero utilizar una marca de tiempo como clave permite conservar el tiempo de cada lectura. Para guardar los datos en Redis, la velocidad y el ID del dispositivo se convierten en un string para unificar el tipo de dato.

La función `send` se utiliza para enviar un acuse de recibo (ACK) a la estación, llevando el mismo identificador de dispositivo que el paquete recibido.

A intervalos regulares definidos por `influxSleep` (actualmente configurado en 60), la aplicación consolida los resultados de velocidad e identificador de dispositivo almacenados en Redis y los envía como puntos de datos a InfluxDB. La biblioteca `influxdb_client` facilita este proceso, enviando datos en lotes de 60 lecturas para reducir la sobrecarga de la red que produce el intercambio HTTP y la autenticación. Cada lote de lecturas se transmite como una lista de puntos, donde cada punto representa una lectura individual, incluyendo la velocidad y el identificador del dispositivo.

Con una configuración de 9600 baudios, 8 bits por byte y sin paridad, la tasa máxima de transferencia teórica es de 9600 bits por segundo. Dado que cada estación transmite una vez por segundo y cada transmisión implica un ACK, el límite 'duro' de estaciones que podrían reportarse a un mismo gateway se determina por la suma de 'tiempo de transmisión + procesamiento + tiempo de ACK'. Despreciando el tiempo de procesamiento, el sistema podría atender un máximo de 400 estaciones.

El código incorpora además la biblioteca `logging` para registrar los errores que puedan surgir durante la ejecución en un archivo de texto 'debug.log'.

Figura 4.3.2. Implementación del Gateway.

4.4 Implementación de la aplicación web

La presentación de lecturas se realiza utilizando el ID de dispositivo correspondiente a cada endpoint, lo que garantiza la precisión de los datos visualizados. Actualmente, dado que se trata de un solo dispositivo, no se requieren ajustes adicionales en los datos antes de su representación. InfluxDB permite aplicar transformaciones y filtrado de los datos de series temporales directamente en la web. De todas formas, cualquier formato necesario en los datos puede aplicarse previamente a la escritura en la base de datos directamente desde la aplicación en Python que se ejecuta en el gateway.

Para el almacenamiento y visualización de datos, se emplea una cuenta free tier de InfluxDB Cloud. Junto con la librería de Python, se realiza el envío de datos en lotes de 60 lecturas mediante la función write_api.write() de la API de InfluxDB 2.0.

La interfaz web del tablero de InfluxDB se muestra en la Figura 4.4.1.

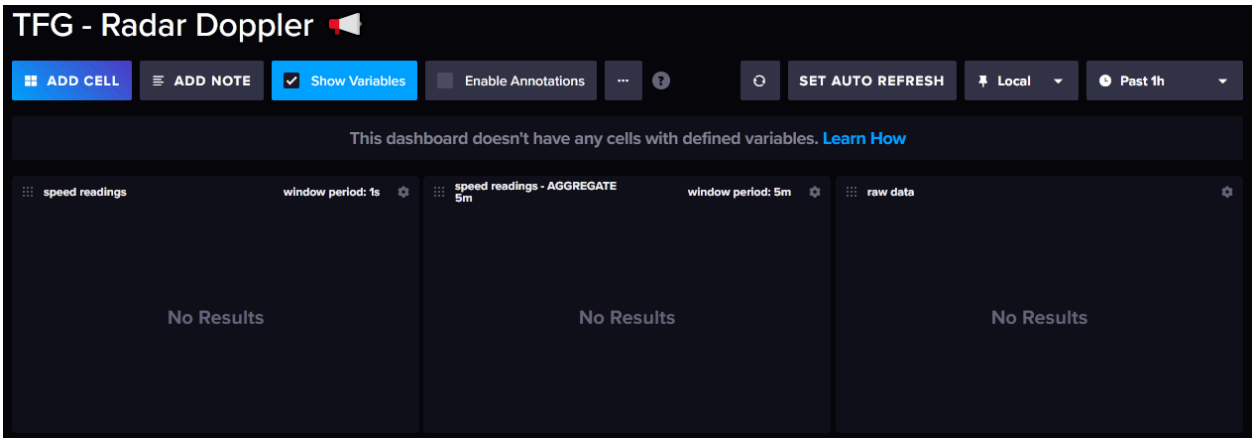

Figura 4.4.1: Interfaz gráfica web de InfluxDB.

5. Pruebas

5.1 Optimización del filtro autocorrelador

En esta etapa, utilizando dos puntos arbitrarios (reposo y señal de calibración), se busca determinar una configuración óptima y obtener datos para comprender el impacto de cada parámetro en el comportamiento general del sistema.

Los parámetros ajustables incluyen la profundidad de correlación (es decir, el número de veces que se comparan las duraciones de varios semiciclos de la señal de entrada para detectar coherencia) y el valor máximo de jitter permitido (la diferencia acumulada entre todas las muestras de la duración de los semiciclos). Cabe destacar que no se realizó la programación y desarrollo de pruebas automáticas ni simulaciones; en su lugar, las pruebas se llevaron a cabo modificando constantes en el código.

En este capítulo, se presentan los resultados de las pruebas posteriores a la implementación del sistema. Se utilizó el mismo programa que se ejecuta en el gateway para imprimir los valores recibidos en la consola.

Se eligieron varios valores para la profundidad de correlación y el jitter máximo, incluyendo 2, 4 y 6 para la profundidad, y 5, 10, 50 para el jitter máximo. Como valor extremo, se utilizó un valor de 8 para la profundidad y de 500 para el jitter máximo. Estos valores se seleccionaron para combinar entre sí y evaluar cómo incide cada parámetro en el comportamiento del filtrado.

5.1.1 Mediciones en reposo / solo ruido de fondo

En esta prueba, simplemente se enciende el sistema y permanece en reposo, apuntado hacia una pared. De esta forma se obtiene el comportamiento del aparato respecto al ruido interno.

Se realizan las pruebas modificando la profundidad del autocorrelador y el corte de Jitter máximo y se muestran en la Tabla 5.1.1.

Profundidad del autocorrelador	Corte MAX_JITTER	Resultados
2	5	samples: 9; 9; 9; 9; 9; 16; 0; 13; 12; 13 stdev: 4.25
2	10	samples: 13; 12; 13; 12; 21; 14; 0; 12; 13; 12 stdev: 5.07
2	50	samples: 13; 12; 13; 13; 14; 13; 12; 12; 12; 12 stdev: 0.70
4	5	samples: 0; 0; 0; 0; 0; 0; 0; 0; 0; 0 stdev: 0
4	10	samples: 0; 0; 0; 0; 0; 18; 0; 0; 0; 0 stdev: 5.69
4	50	samples: 19; 22; 21; 17; 20; 20; 18; 19; 20; 20 stdev: 1.43
6	5	samples: 0; 0; 0; 0; 0; 0; 0; 0; 0; 0 stdev: 0
6	200	samples: 20; 19; 19; 19; 18; 21; 21; 20; 20; 19 stdev: 0.97
8	500	samples: 15; 15; 14; 15; 15; 18; 18; 20; 18; 19 stdev: 2.11

Tabla 5.1.1: Muestreo en reposo.

5.1.2 Mediciones de la señal de calibración

Se utiliza el diapasón de calibración analizado en la sección 3.1.3 (del cual se muestra el análisis en frecuencia en la Figura 5.1.1) para generar una lectura de 50 Km/h y se muestran los resultados en la tabla 5.1.2.

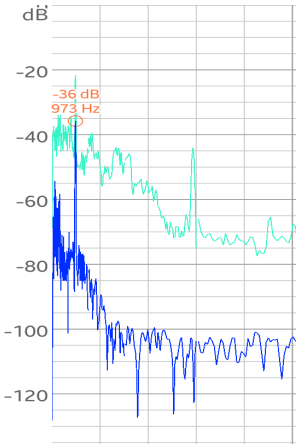

Figura 5.1.1: Espectro de la señal de calibración 973 Hz.

Profundidad del autocorrelador	Corte MAX_JITTER	Resultados
2	5	samples: 51; 50; 50; 50; 50; 50; 74; 50; 17; 50 stdev: 13.58
2	10	samples: 50; 34; 50; 50; 0; 50; 50; 50; 50; 51 stdev: 16.11
2	50	samples: 50; 50; 50; 50; 51; 51; 50; 50; 50; 50 stdev: 0.42
4	5	samples: 50; 51; 50; 50; 50; 50; 51; 50; 50; 51 stdev: 0.48
4	10	samples: 50; 50; 50; 50; 50; 50; 51; 50; 51; 51 stdev: 0.48
4	50	samples: 50; 50; 50; 50; 50; 50; 50; 50; 50; 50 stdev: 0
6	5	samples: 0; 0; 0; 0; 0; 0; 0; 0; 0; 0 stdev: 0
6	200	samples: 50; 0; 0; 0; 51; 0; 0; 50; 50; 0 stdev: 25.95
8	500	samples: 51; 50; 51; 51; 50; 51; 50; 51; 51; 50 stdev: 0.52

Tabla 5.1.2: Muestreo en presencia de la señal de calibración.

5.1.3 Costo computacional de leer el GPIO

Como se expone en la sección 4.2.1.1, el costo de leer el GPIO es de aproximadamente 1,65 µs. Si se tiene en cuenta que el clock del MPU es de 64 MHz, entonces cada ciclo toma 1.6 x10^{-8} s, entonces se estima que cada *poll* al GPIO consume 103 ciclos. De vuelta, esto se calculó usando un método muy simple, como se muestra en el código Cod. 5.1.3.1.

```
1. refTick = HAL_GetTick();
2. for (i=0; i<32768; i++){
3.      HAL_GPIO_ReadPin(GPIOB, RADAR_Pin);
4. }
5. benchmarkResult = (HAL_GetTick() - refTick);
```

Cod. 5.1.3.1

Esto da como resultado que las 32768 lecturas se llevan a cabo en 54 ms (de ahí se deducen los 1,65 µs por acceso). Dado que este tiempo de espera es relativamente corto frente a una señal de entrada del orden de los kilohertz, el uso de lecturas de pines del GPIO en bucles "while" puede considerarse adecuado para efectuar la medición.

5.2 Pruebas de interferencia

Se utiliza un tercer transceiver para insertar datos aleatorios en la banda de comunicaciones de 915 MHz

```
facundo@orangepizero:~$ tail debug.log
ERROR:root:At 2024-02-14 22:00:39, the Rx frame was (0, 0, 16, 0, 119, 0, 0, 85, 255, 108, 6, 0)
ERROR:root:At 2024-02-14 22:00:40, the Rx frame was (0, 0, 16, 0, 36, 0, 0, 85, 255, 108, 6, 0)
ERROR:root:At 2024-02-14 22:00:41, the Rx frame was (0, 0, 16, 0, 203, 0, 0, 85, 255, 108, 6, 0)
ERROR:root:At 2024-02-14 22:00:42, the Rx frame was (0, 0, 16, 0, 105, 0, 0, 85, 255, 108, 6, 0)
ERROR:root:At 2024-02-14 22:00:43, the Rx frame was (0, 0, 16, 0, 12, 0, 0, 85, 255, 108, 6, 0)
ERROR:root:At 2024-02-14 22:00:44, the Rx frame was (0, 0, 16, 0, 172, 0, 0, 85, 255, 108, 6, 0)
ERROR:root:At 2024-02-14 22:00:45, the Rx frame was (0, 0, 16, 0, 72, 0, 0, 85, 255, 108, 6, 0)
ERROR:root:At 2024-02-14 22:00:46, the Rx frame was (0, 0, 16, 0, 231, 0, 0, 85, 255, 108, 6, 0)
ERROR:root:At 2024-02-14 22:00:47, the Rx frame was (0, 0, 16, 0, 130, 0, 0, 85, 255, 108, 6, 0)
ERROR:root:At 2024-02-14 22:00:48, the Rx frame was (0, 0, 16, 0, 45, 0, 0, 85, 255, 108, 6, 0)
```

Figura 5.2.1: Registro de errores del gateway.

El sistema no acepta tramas que no contengan un 16 (0x10) en el primer byte y crea una entrada en un archivo local de registro de errores, acorde a lo establecido en el diseño.

5.3 Simulación con un generador de señales

Se utilizó un generador de onda cuadrada (una aplicación gratuita de la tienda de Android) y un amplificador operacional para eliminar la excursión negativa de la señal y así llevar el nivel a valores siempre mayores a 0 V para evitar dañar la entrada del microcontrolador. En la Figura 5.3.1 se puede observar el conjunto generador y display.

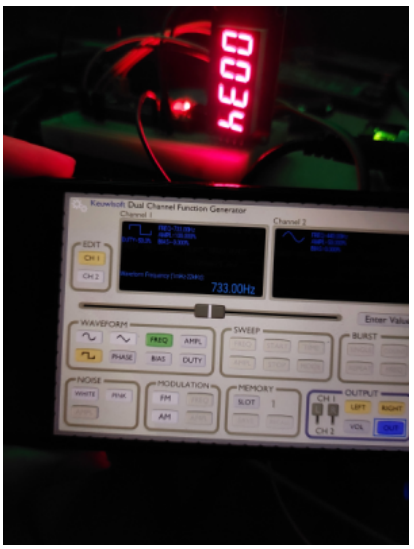

Figura 5.3.1. Conjunto generador y display.

Las pruebas muestran que el sistema es capaz de validar los pulsos de un generador de señal. En la Figura 5.3.2 se puede apreciar los valores de salida de la función utilizada en el código del Gateway (que recibe únicamente valores de tiempo desde el STM32 con precisión de 8 bits). A mayor frecuencia, menor es el tiempo que lleva contar los pulsos en el bucle final del código que se ejecuta en el MCU.

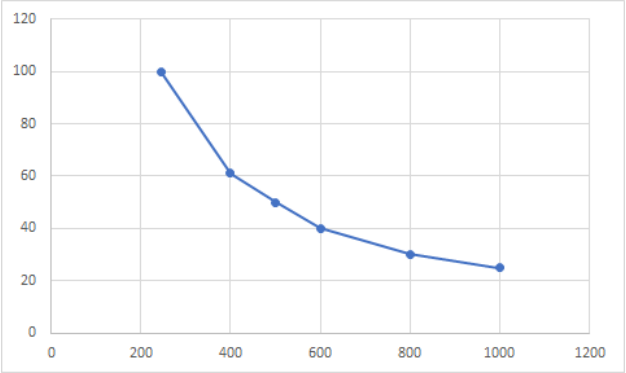

Figura 5.3.2. Comportamiento V(f).

5.4 Pruebas sobre vehículos

Se realiza una prueba de sensibilidad. Se instala sobre un trípode, enfocando la antena hacia una esquina. En la imagen de la figura 5.3.1 se verifica que la lectura de calibración devuelva la misma magnitud tanto en el display del aparato como en el display del endpoint.

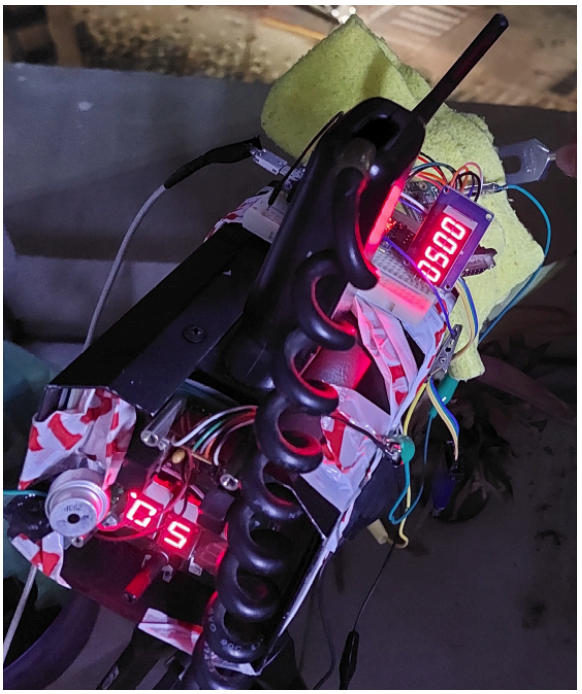

Figura 5.4.1: Prueba de lectura de calibración.

Una vez instalado, se toman mediciones durante unas horas. Se implementa temporalmente un control automático del valor de jitter máximo para encontrar valores que otorguen la suficiente sensibilidad sin llenar el registro de valores aleatorios. En la figura 5.4.2 se muestra el tablero web y, a la derecha, en la misma imagen, la ventana con la aplicación del gateway encontrando un valor óptimo para el valor de corte.

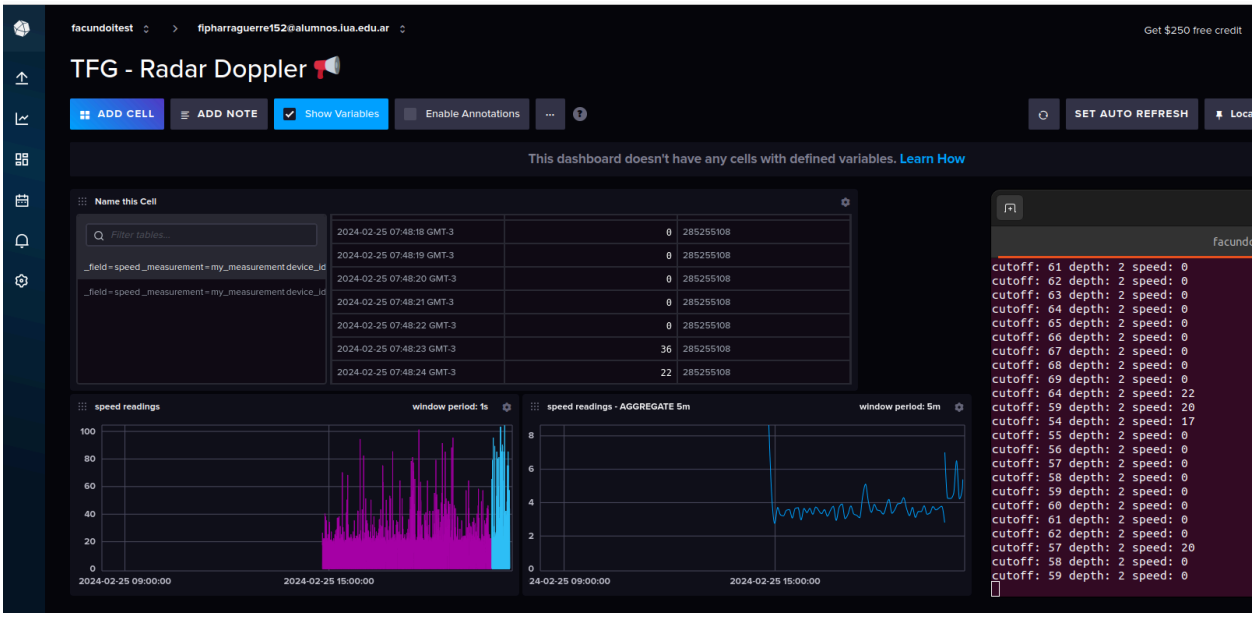

Figura 5.4.2: Tablero de InfluxDB y aplicación del gateway.

Solo para explorar la versatilidad del sistema, se realizó una prueba antes de una tormenta para tomar las diferencias en el volumen de tránsito vehicular.

En la Figura 5.3.2, el gráfico muestra la sumatoria de mediciones agregadas cada 5 minutos (suma todos los valores recibidos, incluso los valores en 0, por grupos de 5 minutos, y muestra el promedio), la línea vertical marca el momento donde comienza a llover. Se aprecia esta caída de la velocidad promedio debido a la diferencia del volumen de tráfico (al existir más mediciones en 0 Km/h).

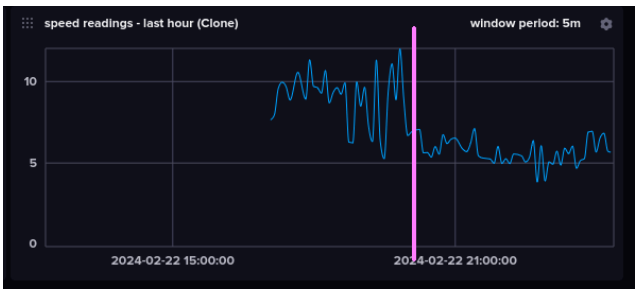

Figura 5.4.3: Lecturas de velocidad promedio cada 5 minutos.

6. Análisis de Resultados Experimentales y Discusión

El propósito de este proyecto ha sido el desarrollo de una plataforma para la adquisición de datos de telemetría. Se diseñó e implementó un sistema embebido para medir la velocidad de vehículos utilizando la señal de un radar Doppler, permitiendo así la obtención de datos de manera eficiente y económica.

Durante las pruebas realizadas, se observó que el sistema mostró sensibilidad a la distancia y ángulo de incidencia respecto a los objetivos. Para garantizar una mayor precisión en la recolección de datos, se recomienda instalar el sistema lo más cerca posible del lugar donde se requiere relevar los datos. Esta medida ayudará a minimizar las posibles variaciones en la medición y a mejorar la calidad de los resultados obtenidos. Además, es importante tener en cuenta que existe un retraso de aproximadamente 1 segundo entre el muestreo de los datos y su presentación en el display, lo que influye en la percepción de los usuarios y debería tomarse como un factor determinante al momento de elegir un lugar apropiado.

Ventajas y desafíos: La principal ventaja del sistema radica en su capacidad para operar con una variedad de dispositivos que proporcionen datos modulando la frecuencia de una señal, permitiendo adaptar su uso a distintos tipos de sensores. Sin embargo, su simplicidad puede presentar limitaciones en la precisión de la medición cuando la señal recibida es débil, en cuyo caso podrían explorarse métodos alternativos, por ejemplo, los basados en el cálculo de la Transformada Rápida de Fourier (FFT) de la señal de entrada.

El sistema ha demostrado una precisión aceptable en la captura de datos, tanto en entornos de pruebas reales como simulaciones utilizando generadores de frecuencias. El uso de InfluxDB Cloud ha permitido visualizar los datos con precisión de un segundo entre muestras, con una latencia mínima de aproximadamente 5 segundos para presentar los gráficos de la serie temporal.

7. Conclusiones

Teniendo en cuenta los objetivos del proyecto, se puede afirmar que se logró desarrollar la plataforma para obtener los datos de telemetría. Se implementó un sistema embebido para medir la velocidad de vehículos mediante el procesamiento de la señal de un radar Doppler.

Además de cumplir con el objetivo principal del proyecto, su desarrollo ha implicado la adquisición de conocimientos en diversas áreas relacionadas con el diseño e implementación de sistemas de radiocomunicación de mediana complejidad. Entre estas áreas se incluyen la organización de proyectos de investigación, la importancia del chasis y los planos de masa en dispositivos electrónicos, el uso de lenguajes de programación como C, Python y SQL, la implementación de bases de datos clave-valor, el manejo de interrupciones y temporizadores, el uso del acceso DMA, el desarrollo de algoritmos de filtrado y la optimización de antenas, entre otros aspectos relevantes.

Es importante destacar que el sistema es altamente escalable y sujeto a mejoras, habiéndose tenido en cuenta criterios de diseño que facilitan la incorporación de nuevas funcionalidades con la menor cantidad de obstáculos posibles.

7.1 Recomendaciones para versiones futuras

Con base en las observaciones realizadas y las lecciones aprendidas durante este proyecto, se ofrecen las siguientes recomendaciones para futuras investigaciones y mejoras:

Detección y comunicaciones:

Explorar la posibilidad de realizar correcciones por software para distintos tipos de sensores, permitiendo una mayor flexibilidad y adaptabilidad del sistema a diferentes condiciones de operación. Esto podría lograrse mediante la creación de una tabla de búsqueda de valores predeterminados (*lookup table* preconfigurada) o permitiendo al usuario cargar valores arbitrarios a modo de *prescaler*. Esta mejora aumentaría la flexibilidad y adaptabilidad del sistema, permitiendo su uso en una variedad más amplia de escenarios y aplicaciones. Además, podría implementarse algún tipo de filtrado utilizando la FFT en los *endpoints*, así como detectar movimiento con un sensor óptico para habilitar las lecturas de velocidad. Implementar controles automáticos de los coeficientes de profundidad y corte en el filtrado.

Implementar un esquema de acceso múltiple por división de tiempo (TDMA) para sincronizar las estaciones que se reporten a una misma puerta de enlace, mejorando así la eficiencia y reduciendo posibles colisiones de tramas.

Interfaz:

La interfaz web actual tiene un diseño bastante intuitivo; sin embargo, existe una oportunidad de mejora mediante la implementación de un diseño personalizado. Se sugiere explorar la posibilidad de integrar una API REST que permita recibir los datos. por ejemplo, en formato JSON, almacenarlos en una base de datos y visualizarlos de forma dinámica mediante herramientas como Grafana o una

visualización personalizada. Esta mejora no solo mejoraría la experiencia del usuario, sino que también ampliaría las capacidades de análisis y seguimiento del sistema.

Ya que el volumen de datos es contundente, se puede evaluar la aplicación de análisis predictivos para detectar y anticipar eventos indeseados (por ejemplo, congestión vehicular).

Uso de interrupciones para detectar cambios en la señal:

En lugar de realizar *polling* continuo al pin de entrada, se puede utilizar una interrupción para que el sistema solo se active cuando haya una señal presente. Esto detectaría los flancos ascendentes y mejoraría la eficiencia del sistema al reducir el consumo de energía y la carga del procesador mientras el nivel se mantenga. La implementación de interrupciones generalmente suele ser sencilla y se puede realizar utilizando el hardware y el software del microcontrolador que se esté utilizando.

Alternativas al sensor actual:

- Sensor HB100: HB100 es un radar Doppler microondas estándar de 10,525 GHz, con un voltaje de DC de 5 V y una corriente máxima de 45 mA. Tiene un alcance de hasta 10 m (se puede mejorar con una antena), dimensiones de 46,3 x 40,2 x 5,7 mm, con un peso de 6 g. Su costo es de alrededor de 5 USD.

- Sensor IVS-979-FMCW: El sensor IVS-979 de InnoSenT es una solución de radar de alta sensibilidad para aplicaciones de tráfico de largo alcance, ya que cuenta con un rango de detección típicamente de hasta 150 m, y un tamaño de 120 x 70,5 x 11,4 mm. Su campo de visión (de potencia media) es de 7° x 28° para concentrarse en la detección específica de carriles. Permite extraer información de velocidad y distancia de las señales recibidas y su costo ronda los 270 USD.

Bibliografía

Stallings, W. (2015). Wireless communication networks and systems - 2nd edition. *Pearson - Prentice Hall.*

Stallings, W. (2008). Comunicaciones y redes de computadores - 7ª edición. *Pearson - Prentice Hall.*

Kernighan & Ritchie. (1988). The C Programming Language - 2nd edition. *Bell Telephone Laboratories, Incorporated.*

Hambley, A. (2014). Electrical Engineering: Principles and Applications - 6th edition. *Pearson Education, Inc.*

A. B. P. Lathi (1998). Signal Processing and Linear Systems, *Berkeley-Cambridge.*

Asimakopoulos, D. N., & Bourbakis, N. G. (2011). Intelligent vehicle speed monitoring system based on radar technology. IEEE Transactions on Intelligent Transportation Systems, 12(3), 863-872.

Avramchuk, V., & Goncharov, A. (2013). Leak detection in pipelines using correlation and filtering. Journal of Physics: Conference Series, 450(1), 012013. https://doi.org/10.1088/1742-6596/450/1/012013

Bhaktavatsala (2002). DSP APPLICATIONS IN RADAR. Recuperado el 5 de noviembre de 2023, desde https://www.ee.iitb.ac.in/~esgroup/es_mtech02_sem/es02_sem_rep_bhakta.pdf

Blu, T., & Unser, M. (2003). Quantitative Fourier analysis of approximation techniques: Part I - Interpolators and projectors. IEEE Transactions on Signal Processing, 51(2), 433-446.

Burchianti, A., & Salvini, A. (2016). A new Doppler radar based sensor for vehicle speed measurement. Sensors, 16(12), 2125.

David Veneziano; Larry Hayden; Jared Ye. (2010). Effective Deployment of Radar Speed Signs.

Duona Z, et al. (2021). “Learning modulation filter networks for weak signal detection in noise”. Pattern Recognition 109, 107590.

Fano, R. M.: Signal-to-noise ratios in correlation detectors. MIT Research Lab. Electronics Tech. Rept. 186, 1951.

Gao, J., & Luo, Y. (2014). Vehicle speed detection using a Doppler radar sensor. IEEE Sensors Journal, 14(9), 3032-3040.

Jian-fei, L., Jian, L., & Wei, L. (2009). Research on the correlation and filtering method in satellite communication. 2009 International Conference on Measuring Technology and Mechatronics Automation, 2, 352-355.

Kay. S. M. (2003). “Fundamentals of Statistical Signal Processing, Volume I: Estimation Theory”. Beijing: China Machine Press.

Li, J., Li, H., Luo, Y., & Li, Z. (2017). Study on vehicle speed measurement based on a Doppler radar. Measurement, 104, 158-165.

Li, H., Li, J., Li, Z., & Luo, Y. (2016). A Doppler radar based on frequency-modulated continuous-wave for vehicle speed detection. Sensors, 16(10), 1731.

Lemos, M., & Costa, R. (2014). Vehicle speed measurement using a Doppler radar system. Journal of Sensors, 2014.

Moulin, P. (2006). Wavelet-based statistical signal processing using hidden Markov models. IEEE Transactions on Signal Processing, 54(11), 4182-4194.

Parker, M. (2017). Digital Signal Processing 101, Everything You Need to Know to Get Started. Newnes, an imprint of Elsevier.

Pineda Márquez, J. L. (2009). Filtrado digital de señales de radar. En J. L. Pineda Márquez, Procesamiento digital de señales aplicado a las comunicaciones (pp. 23-38). México: Alfaomega

Purdy, R. J., Blankenship, P. E., Muehe, C. E., Rader, C. M., Stern, E. A., & Williamson, R. C. (2000). Radar signal processing. Lincoln Laboratory Journal, 12(2), 41-60.

Salahdine, F. (2018). Compressive Spectrum Sensing for Cognitive Radio Networks (Tesis de doctorado, National Institute of Posts and Telecommunications).

Skrivervik, A. K., & Taflove, A. (2010). Modeling and simulation of a Doppler radar for measuring speed of moving vehicles. IEEE Transactions on Antennas and Propagation, 58(8), 2712-2722.

T. Moore and B. Sadler. (2006). “Maximum-Likelihood Estimation and Scoring under Parametric Constraints”. Army Research Lab, Aldelphi, MD, Tech. Rep. ARL-TR-3805.

Wang L, Yang Zh, Waters T.P. (2010). “Structural damage detection using cross correlation functions of vibration response”. Journal of Sound and Vibration 329, pp. 5070–5086.

Yuandong, W., Zhiqiang, W., & Xuefeng, C. (2021). Fault diagnosis of mechanical systems based on improved adaptive thresholding method. Journal of Mechanical Engineering, 57(5), 1-9.

Zhang, B., Fan, X., & Li, W. (2019). Doppler radar speed measurement and vehicle tracking algorithm based on improved dynamic threshold. IET Intelligent Transport Systems, 13(10), 1534-1541.

Zhang, H., Ma, W., Zhang, W., & Zhang, Y. (2019). A new signal processing method for vehicle speed measurement using a Doppler radar. Measurement, 136, 19-26.

STM32 Arm® Cortex®-based MCU user guide. [https://wiki.st.com/stm32mcu]

STM32F103 Datasheet [https://www.st.com/resource/en/datasheet/stm32f103ze.pdf]

Description of STM32F1 HAL and low-layer drivers. [https://www.st.com/content/ccc/resource/technical/document/user_manual/72/52/cc/53/05/e3/4c/98/DM00154093.pdf/files/DM00154093.pdf]

Armbian Documentation. [https://docs.armbian.com/]

Orange Pi Docs [http://www.orangepi.org/Docs/mainpage.html]

User Guide — influxdb_client documentation [https://influxdb-client.readthedocs.io/en/stable/usage.html]

Federal Highway Administration's (FHWA's) - Speed Management Digital Library [https://safety.fhwa.dot.gov/speedmgt/ref_mats/]

Resources - Eco-Counter [https://www.eco-counter.com/resources/]

RECOMENDACIÓN OIML R – 91: EQUIPOS DE RADAR PARA MEDIR LA VELOCIDAD DE VEHÍCULOS. Edición 1990 (E) https://www.oiml.org/en/publications/other-language-translations/spanish/r091-es90.pdf

Argentina.gob.ar. (2021). Informe de Siniestralidad Vial Fatal Año 2021. Recuperado de https://www.argentina.gob.ar/sites/default/files/2018/12/ansv_informe_siniestralidad-vial_fatal_2021_datos_preliminares.pdf

Argentina.gob.ar. (2022). Informe de siniestralidad vial fatal Año 2022. Recuperado de https://www.argentina.gob.ar/sites/default/files/2018/12/ansv_informe_siniestralidad_2022_datos_preliminares.pdf

OCDE. (2018). VELOCIDAD Y RIESGO DE SINIESTROS VIALES. Recuperado de https://stopaccidentes.org/publicaciones-asociacion-mejorar-seguridad-vial/velocidad-y-riesgo-de-siniestros-viales-ocde-irtad/gmx-niv203-con1313.htm

Físicas. (2017). Instrumentos de Medida de Velocidad. Recuperado de https://www.fisicas.info/2017/06/instrumentos-de-medida-de-velocidad.html

Argentina.gob.ar. (2021). Informe de Siniestralidad Vial Fatal Año 2021. Recuperado de https://www.argentina.gob.ar/sites/default/files/2018/12/ansv_informe_siniestralidad-vial_fatal_2021_datos_preliminares.pdf

Stop Accidentes. (s.f.). VELOCIDAD Y RIESGO DE SINIESTROS VIALES. OCDE. IRTAD 2018. Recuperado de https://stopaccidentes.org/publicaciones-asociacion-mejorar-seguridad-vial/velocidad-y-riesgo-de-siniestros-viales-ocde-irtad/gmx-niv203-con1313.htm

Tuteorica. (2021). Relación de la velocidad con la siniestralidad vial. Recuperado de https://tuteorica.com/material-complementario/relacion-de-la-velocidad-con-la-siniestralidad-vial/

Sandberg, W., Schoenecker, T., Sebastian, K., & Soler, D. (n.d.). Long-Term Effectiveness of Dynamic Speed Monitoring Displays (DSMD) for Speed Management at Speed Limit Transitions. Recuperado de https://highways.dot.gov/media/15211

PB Electronics. (s.f.). Police Radar - K15 Specifications. Recuperado de https://www.pbelectronics.com/K15_specifications.htm

CopRadar.com. (s.f.). M.P.H. K-15 police traffic radar. Recuperado de https://copradar.com/rdrspecs/k15.html

National Highway Traffic Safety Administration. Recuperado de https://www.nhtsa.gov/

Naylamp Mechatronics. (s.f.). Sensor de movimiento por Radar HB-100. Recuperado de https://naylampmechatronics.com/sensores-proximidad/303-sensor-de-movimiento-por-radar-hb-100.html

Arduino Reference. (s.f.). TM1637. Recuperado de https://www.arduino.cc/reference/en/libraries/tm1637/

Solectroshop. (s.f.). Sensor Movimiento Microondas Doppler Radar HB100 10.25GHz. Recuperado de https://solectroshop.com/es/sensores-de-distancia-proximidad-ir-y-ultrasonidos/897-sensor-movimiento-microondas-doppler-radar-hb-100-1025ghz.html

Silicon Laboratories. (s.f.). 240–960 MHz EZRadioPRO® transceiver. Recuperado de https://www.silabs.com/documents/public/data-sheets/Si1000.pdf

InnoSenT. (s.f.). ITR-3810. Recuperado de https://www.innosent.de/en/radar-systems/itr-3810/

Eco-Counter. (s.f.). Eco Display Compact 2. Recuperado de https://www.eco-counter.com/produits/real-time-displays/eco-display-compact-2/

Smart Prototyping. (s.f.). HB100 Doppler Module. Recuperado de https://www.smart-prototyping.com/HB100-Doppler-Module

InnoSenT. (s.f.). IVS-979 FMCW/FSK Radarsensor. Recuperado de https://www.innosent.de/en/sensors/ivs-979-fmcw/fsk-radarsensor/

Omnipresense. (s.f.). OPS7243-A Doppler Radar Sensor in All-Weather Enclosure. Recuperado de https://omnipresense.com/product/ops7243-a-doppler-radar-sensor-in-all-weather-enclosure/

Anexo A: Bucle infinito del endpoint

```
1.  while (1) {
2.      for (i = 0; i < filteringDepth; i++) {
3.              HAL_Delay(20);
4.              while (HAL_GPIO_ReadPin(GPIOB, RADAR_Pin));
5.              while (!HAL_GPIO_ReadPin(GPIOB, RADAR_Pin));
6.              while (HAL_GPIO_ReadPin(GPIOB, RADAR_Pin)) {
7.                      nthPulseWidth[i]++;
8.              }
9.              if (i == 0) {
10.                     maxPulseWidth = nthPulseWidth[0];
11.                     minPulseWidth = nthPulseWidth[0];
12.             }
13.             if (nthPulseWidth[i] > maxPulseWidth) {
14.                     maxPulseWidth = nthPulseWidth[i];
15.             }
16.             if (nthPulseWidth[i] < minPulseWidth) {
17.                     minPulseWidth = nthPulseWidth[i];
18.             }
19.     }
20.     jitter = maxPulseWidth - minPulseWidth;
21.     if (jitter < jitterCutoff) {
22.             refTick = HAL_GetTick();
23.             for (i = 0; i < (30 * filteringDepth); i++) {
24.                     while (HAL_GPIO_ReadPin(GPIOB, RADAR_Pin));
25.                     while (!HAL_GPIO_ReadPin(GPIOB, RADAR_Pin));
26.             }
27.             deltaSystick = (HAL_GetTick() - refTick);
28.             normalizedInputPeriodTime = 2*deltaSystick/filteringDepth;
29.     }
30.     else {
31.             normalizedInputPeriodTime = 0;
32.     }
33. }
```

En la línea 3, se espera 20 ms para saltar al próximo pulso completo. Una vez finalizada esta espera, en la línea 7, se incrementa un entero de 16 bits sin signo para registrar el ancho del pulso. Posteriormente, en las líneas 9 a 18, se actualizan los valores máximo y mínimo de los pulsos. Finalmente, se calcula el jitter entre el máximo y el mínimo en la línea 20. Si el jitter es menor que el valor de corte, se inicia un proceso adicional en las líneas 22 a 28 para calcular el tiempo de duración de múltiples pulsos y normalizar el valor para obtener el periodo de la onda completa. Si el jitter es mayor o igual al valor de corte, se asigna un valor de cero a la variable normalizedInputPeriodTime en las líneas 30 a 32.

Anexo B: Callbacks del MCU

B.1 Interrupción del temporizador

```
1. void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef* htim){
2.      HAL_GPIO_TogglePin(LED_GPIO_Port, LED_Pin);
3.      tx[0] = 0x10;
4.      tx[1] = (unsigned char) normalizedInputPeriodTime;
5.      tx[2] = (unsigned char) benchmarkResult;
6.      tx[3] = (unsigned char) jitterCutoff;
7.      tx[4] = (unsigned char) filteringDepth;
8.      tx[5] = devId[0];
9.      tx[6] = devId[1];
10.     tx[7] = devId[2];
11.     tx[8] = devId[3];
12.     HAL_UART_Transmit(&huart1, tx, sizeof(tx), 255);
13.     normalizedInputPeriodTime = 0;
14.     maxPulseWidth = 0;
15.     minPulseWidth = 0;
16.     for(i=0; i < filteringDepth; i++){
17.     nthPulseWidth[i] = 0;
18.     }
19. }
```

En la función HAL_TIM_PeriodElapsedCallback, se llevan a cabo una serie de acciones. Primero, se alterna el estado del LED integrado en la línea 2. Luego, se llenan varios elementos del buffer de transmisión tx con valores relevantes para la comunicación UART en las líneas 3 a 11. Un valor que resulta crítico insertar es el byte 0, donde se carga h0x10 para indicar que la trama proviene de un endpoint. El resto de los valores incluyen información como el tiempo de periodo normalizado, el resultado del benchmark (esto es accesorio), el límite de jitter actual, la profundidad de filtrado y un identificador de dispositivo de 32 bits. En la línea 12, se envía el buffer tx a través de la UART en modo bloqueante. Posteriormente, se restablecen algunas variables temporales y se reinicia una matriz de ancho de pulso del correlador en las líneas 13 a 17 para futuras mediciones.

B.2 Interrupción DMA (Buffer UART Rx)

```
1. void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
2. {
3.      HAL_UART_Receive_DMA(&huart1, rx, 12);
4.      uint8_t controlByte = (rx[0] & 0xF0);
5.      uint8_t pA[4] = {0};
6.      uint8_t pB[4] = {0};
7.      uint8_t pC[3] = {0};
8.      for (int i=0;i<4;i++){
9.      pA[i]=rx[i+1];
10.     pB[i]=rx[i+5];
11.     if(i<3){
12.             pC[i]=rx[i+9];
13.     }
14.     }
15.     switch (controlByte){
16.     case 0x00:
17.             for (int i=0;i<4;i++){
18.                     dataWord[i]=pA[i];
19.                     if (devId[0] == pB[0] && devId[1] == pB[1] && devId[2] == pB[2] &&
devId[3] == pB[3]){
20.                     jitterCutoff = dataWord[2];
21.                     filteringDepth = dataWord[3];
22.                     if(dataWord[0] != 0){
23.                             tm1637DisplayDecimal((unsigned) dataWord[0], 0);
24.                     }
25.                     else{
26.                             tm1637DisplayDecimal((unsigned) 8080, 0);
27.                     }
28.                     }
29.             }
30.             break;
31.     case 0x10:
32.             break;
33.     default:
34.             break;
35.     }
36. }
```

La función `HAL_UART_RxCpltCallback` se activa cuando se completa el buffer de recepción del UART. En las líneas 3 a 13, se reciben y procesan los datos del buffer `rx`, extrayendo los bytes relevantes para su posterior análisis. Luego, en la línea 15, se realiza un switch basado en el byte de control `controlByte`. En el caso de que `controlByte` sea 0x00 (líneas 16 a 27), se procesa la trama con los datos relevantes para el dispositivo: se comprueba si el paquete es para este dispositivo, y si es así, se actualizan las variables `jitterCutoff` y `filteringDepth`. Además, se actualiza la pantalla TM1637 con datos relevantes si se reciben. En el caso de que `controlByte` sea 0x10 (línea 31), se ignora el paquete, ya que proviene de otra estación. Finalmente, cualquier otro valor de `controlByte` se considera un encabezado incorrecto y se maneja como un error.

Anexo C: Bucle infinito del gateway

```
1. while True:
2.     if ser.in_waiting > 0:
3.     response = ser.read(12)
4.     RxBuffer = struct.unpack('BBBBBBBBBBBB', response[0:12])
5.     ctrl = RxBuffer[0]
6.     if ctrl == 0x10:
7.             timestamp = int(time.time() * 1e9)
8.             velocidad = int((RxBuffer[1]/30)*12.5)
9.             devID = str(RxBuffer[4]) + str(RxBuffer[5]) + str(RxBuffer[6]) +
str(RxBuffer[7])
10.            r.set(timestamp, str((velocidad, devID)))
11.            TxBuffer = list(RxBuffer[0:12])
12.            TxBuffer[0] = 0x00
13.            TxBuffer[1] = velocidad
14.            TxBuffer[3] = cutoff
15.            TxBuffer[4] = depth
16.            send(bytes(TxBuffer))
17.            if len(r.keys()) == influxSleep:
18.                    lecturas = []
19.                    for key in r.keys():
20.                    velocidad, devID = eval(r.get(key))
21.                    linea = f"mi_medida,ubicacion=MW_Radar,id_dispositivo={devID}
velocidad={velocidad} {int(key)}"
22.                    lecturas.append(linea)
23.                    write_api.write(bucket=bucket, org=org, record=lecturas)
24.                    lecturas.clear()
25.                    r.flushdb()
26.    else:
27.            timestamp = time.strftime('%Y-%m-%d %H:%M:%S')
28.            logging.error('En %s, la trama Rx fue %s', timestamp, RxBuffer)
```

El bucle `while` se ejecuta continuamente para manejar las respuestas del sensor. En las líneas 2 a 28, se procesa la respuesta del sensor cuando hay datos disponibles en el buffer de entrada. Se desempaqueta la respuesta en `RxBuffer` y se extrae el byte de control `ctrl`. Si `ctrl` es igual a 0x10 (líneas 6 a 25), se procesa la trama de datos del sensor. Se obtiene la marca de tiempo actual en nanosegundos, se calcula la velocidad, se obtiene el ID del dispositivo y se almacenan en una base de datos Redis. Luego, se prepara un ACK y se envía, y si se alcanza un límite de lecturas, se escriben en una base de datos InfluxDB y se borra el caché de Redis. En caso contrario (líneas 26 a 28), se registra un error y se marca con la fecha y hora actuales.