ULPGC
Universidad de
Las Palmas de
Gran Canaria

ESCUELA DE
INGENIERÍA INFORMÁTICA

Trabajo de Fin de Grado

Integración de sensores de viento ultrasónicos y de transporte de arena en un sistema distribuido de dispositivos inalámbricos para la captura de datos en entornos dunares

TITULACIÓN: Grado en Ingeniería Informática

AUTOR: Oscar Labrada Love

TUTORIZADO POR:
Antonio Carlos Domínguez Brito
Jorge Cabrera Gámez

Fecha Julio/2023

Agradecimientos

Al concluir este Trabajo Fin de Título, deseo expresar mi más profundo agradecimiento a todas las personas que de una u otra manera contribuyeron a la realización de este proyecto.

En primer lugar, quiero agradecer a mis tutores, Antonio Carlos Domínguez Brito y Jorge Cabrera Gámez, por el invaluable apoyo, orientación y paciencia durante la realización de este TFT.

Un agradecimiento especial a mis compañeros de carrera, con quienes compartí innumerables jornadas de estudio y trabajo que enriquecieron este proceso. Su compañía y colaboración fueron un gran soporte durante estos años de estudio.

No puedo dejar de mencionar el apoyo constante de mi familia. Gracias a mis padres, por su amor, paciencia y motivación constante. A mi hermana, por estar siempre ahí para ofrecerme su ayuda y apoyo.

Finalmente, agradezco a todos aquellos que, de forma directa o indirecta, contribuyeron a este proyecto. Cada uno de ustedes ha sido parte esencial de este camino.

Resumen

Actualmente existe un sistema de medición de condiciones de viento en entornos dunares realizado por la División de Oceanografía Computacional del Instituto Universitario SIANI, en colaboración con el Institutuo Universitario IOCAG, como parte del proyecto DUNCAN. Este sistema contiene un conjunto de dispositivos sensores de viento que miden la intensidad y dirección del viento. Estos dispositivos se despliegan cubriendo una zona dunar a estudiar para recabar datos del estado de viento a lo largo del entorno dunar.

Este trabajo de fin de título (TFT) tiene como objetivo expandir la red de sensores con un nuevo tipo de dispositivo, integrando nuevos sensores y mayor funcionalidad.

En el sistema original, llamado versión uno, cada dispositivo de medida integra una veleta y un anemómetro de copa con las que se obtienen los datos de viento actuales. Estos dispositivos se basan en la placa de prototipado electrónico basada en microcontrolador Arduino UNO R3. Junto con esto se dispone de una estación base que recoge datos y sincroniza a los dispositivos, formando una topología de estrella. Esta comunicación entre cada dispositivo y la estación base se realiza a través del protocolo de comunicación inalámbrico ZigBee.

La nueva versión de estos dispositivos, llamada versión dos, pasa a utilizar un anemómetro ultrasónico como su único sensor de viento. Este anemómetro digital proporciona tanto la intensidad como la dirección del viento. Junto a esto también se integra un sensor láser para medir el transporte de arena. Este sensor informa sobre el número de partículas de arena que lo ha atravesado. También se ha actualizado la placa de prototipado electrónico utilizada en esta segunda versión de dispositivos sensores que será la placa Arduino MKR 1310 con interfaz de comunicaciones LoRa.

El uso de LoRa en estos nuevos dispositivos permite los sensores ser desplegados a mayor distancia, permitiendo realizar estudios cubriendo áreas mayores; pero con la consecuencia de recibir actualizaciones en tiempo real con una menor frecuencia.

Con estos cambios se ha creado una nueva versión del sistema de captura de datos de viento en entornos dunares, capaz de trabajar en conjunto con los dispositivos de versión uno con unos cambios mínimos de utilización de cara a los usuarios.

Abstract

Currently, there is a system for acquiring wind data in dune environments developed by the División de Robótica y Oceaonografía Computacional of University Institute SIANI in collaboration with University Institute IOCAG, as part of the DUNCAN project. This system comprises a set of wind sensor devices that measure wind intensity and direction. These devices are deployed around the dune field and are used to gather data on wind conditions throughout the dune environment.

This final degree project (TFT) aims to expand the sensor network with a new device that integrates new sensors and provides enhanced functionality.

The original system, version one, consists of a weather vane and a cup anemometer which collects current wind data. These devices are built around the microcontroller-based electronic prototyping board, Arduino UNO R3. Alongside this, there is a base station that collects data and synchronizes the devices, forming a star topology. Communication between each device and the base station is carried out through the ZigBee wireless communication protocol.

The new version of these devices, called version two, switches to using an ultrasonic anemometer as its sole wind sensor. This digital anemometer provides both wind intensity and direction. In addition, a laser sand transport sensor has been integrated. This sensor reports on the number of sand particles that have passed through it. The microcontroller-based electronic board has also been updated, using the Arduino MKR 1310 board in this new version of the device, which also integrates a LoRa link for communications.

Therefore, in this second version of the device, the communication system has been switched from ZigBee to LoRa. This change allows for the deployment of sensors over larger distances. This however comes with the trade-off of receiving real-time updates less frequently.

Overall, the wind-data acquisition system can integrate both the new second-version devices alongside the original devices when deployed in the field, with minimal changes to the end user's interaction with the system.

Índice general

Índice de figuras

Índice de cuadros

Índice de Listados

Capítulo 1

Introducción

Este TFT trata de ampliar un sistema de distribuido de sensores de viento desarrollado por el SIANI para medir las condiciones de viento en entornos dunares. El sistema consiste en una red de sensores inalámbricos de viento con la que se miden las condiciones a lo largo de la zona de despliegue. Este sistema de sensores es utilizado por el grupo IOCAG de la ULPGC para estudiar la relación entre las condiciones de viento y el movimiento de las dunas de arena.

En la versión uno de esta red de dispositivos, cada dispositivo dispone de un anemómetro de copa y una veleta como sensores de viento. Estos sensores de viento son ampliamente utilizados en este ámbito de aplicación por bajo costo y fácil integración.

Como parte de este TFT se quiere ampliar el sistema creando una segunda versión de los dispositivos sensores. Esta nueva versión de los dispositivos actualiza sus componentes y añade nuevos sensores.

En primer lugar, se mejora el microcontrolador utilizado, pasando de la placa de prototipado Arduino UNO R3 a la placa Arduino MKR 1310. Desde el punto de vista del microcontrolador esto nos trae una mayor capacidad computacional. Junto a esto la placa MKR trae varios beneficios sobre la placa Arduino UNO.

En primer lugar, la placa del Arduino MKR dispone de un controlador de carga de baterías Li-ion. Para el anterior sistema se utilizaba una placa de carga dedicada, permitiéndonos ahorrar espacio en el dispositivo final.

En segundo lugar, la placa del Arduino MKR dispone de un módulo LoRa, un protocolo de comunicación inalámbrica. Esto supone un cambio del protocolo XBEE utilizado en la anterior versión trayendo various beneficios pero también algunas ventajas. Junto a esto se elimina la placa Zigbee utilizado en la anterior versión para proporcionar el servicio XBee.

También se añaden dos dispositivos nuevos a esta nueva versión del sistema, un módulo GPS y un sensor de transporte de arena láser.

El módulo GPS aparte de proporcionar la ubicación también proporciona el tiempo actual. Esto pérmite solventar algunas desventajas que tenía el sistema de sincronización temporal

de la primera versión de este sistema. En la versión uno de este sistema antes de comenzar cualquier ensayo se enviaba un paquete de sincronización con el tiempo actual a todos los dispositivos. Con los dispositivos de esta nueva versión este paquete de sincronización ya no es necesaria.

También se introduce un nuevo sensor, un sensor láser que mide el volumen de arena movido. Este proporciona un nuevo dato para el estudio del movimiento dunar. Ayudando a relacionar las condiciones de viento con este.

Con estos nuevos sensores se ha podido crear una nueva versión de los dispositivos sensores. Utilizándolo para realizar pruebas en conjunto con los dispositivos versión uno. Realizando ensayos para comprobar el funcionamiento del trabajo realizado y probar su integración con el sistema preexistente.

Capítulo 2

Estado actual y objetivos iniciales

Este capítulo aborda el estado actual y los fundamentos que guiaron la creación inicial del sistema de sensores del SIANI, enfocándose en la medición de las condiciones del viento en entornos dunares. Se analizará la red de sensores, proporcionando detalles técnicos del sistema actual y objetivos específicos para la expansión de este sistema.

Primero, se describirá el sistema de sensores actualmente en uso, explicando los componentes individuales y su funcionamiento. Esta sección servirá para establecer una comprensión clara de cómo el sistema ha venido operando hasta ahora.

Posteriormente, se evaluarán los objetivos de una nueva iteración del sistema. Esta evaluación incluirá una propuesta detallada para ampliar la red de sensores, expandiendo su alcance y capacidad, y explorando la introducción de nuevas tecnologías y métodos de medición.

2.1. Estado actual de la red de sensores del SIANI

Actualmente, el SIANI ha desarrollado un sistema basado en una red inalámbrica de dispositivos sensores para medir la intensidad y dirección del viento en campos de arena. Este sistema fue desarrollado como parte del proyecto de investigación DUNCAN, realizada como una colaboración del IOCAG y SIANI, ambos de la ULPGC [5].

2.1.1. Sistema de sensores actuales

En este apartado describiremos el sistema de sensores de viento actualmente en uso, incluyendo sus componentes y su funcionamiento. Una descripción más detallada puede consultarse en [5].

2.1.1.1. Objetivos

Al diseñar el sistema anterior había una serie de objetivos fundamentales que se querían conseguir. Estos objetivos fueron claves para el diseño final del sistema anterior.

Por lo tanto, en este apartado se analizarán los objetivos que el sistema anterior quería cumplir. Esto también dará contexto a los objetivos del trabajo descrito en este documento y de las nuevas necesidades que motivaron este trabajo.

El objetivo de este sistema es monitorizar las condiciones de viento a lo largo de la superficie de un entorno dunar. Para facilitar la utilidad de este sistema se tuvieron en cuenta diversos requisitos que se deberían cumplir.

Uno de los requisitos fundamentales durante el diseño del sistema fue el presupuesto reducido para el desarrollo y compra de materiales. Esta limitación llevó a la utilización de materiales y sensores ampliamente disponibles, y que fueran de costo asequible.

Otro de los requisitos importantes era la robustez y la facilidad de uso de los dispositivos. Las condiciones más duras de un entorno dunar combinado con el aislamiento que provén estos entornos dificultan la asistencia técnica en tiempo real que se puede proveer a los usuarios durante una sesión de medida. Esto combinado con la inversión en tiempo y dinero que requiere el despliegue de un equipo de investigación en el entorno a estudiar hace que sea fundamental que el sistema no sea complejo de utilizar, y que permita la monitorización in situ del sistema de medida.

Finalmente, es importante que estos dispositivos tengan la autonomía para estar activos durante toda la duración de un despliegue experimental típico. Las tareas de investigación de entornos dunares en la que se utilizarían estos dispositivos van desde el amanecer hasta el anochecer. Por lo tanto, estos sistemas deben poder seguir funcionando durante la extensión total del periodo de estudio. Para la versión actual operativa del sistema se decidió que esto deberían ser unas 10 horas de autonomía [5].

2.1.1.2. Componentes hardware

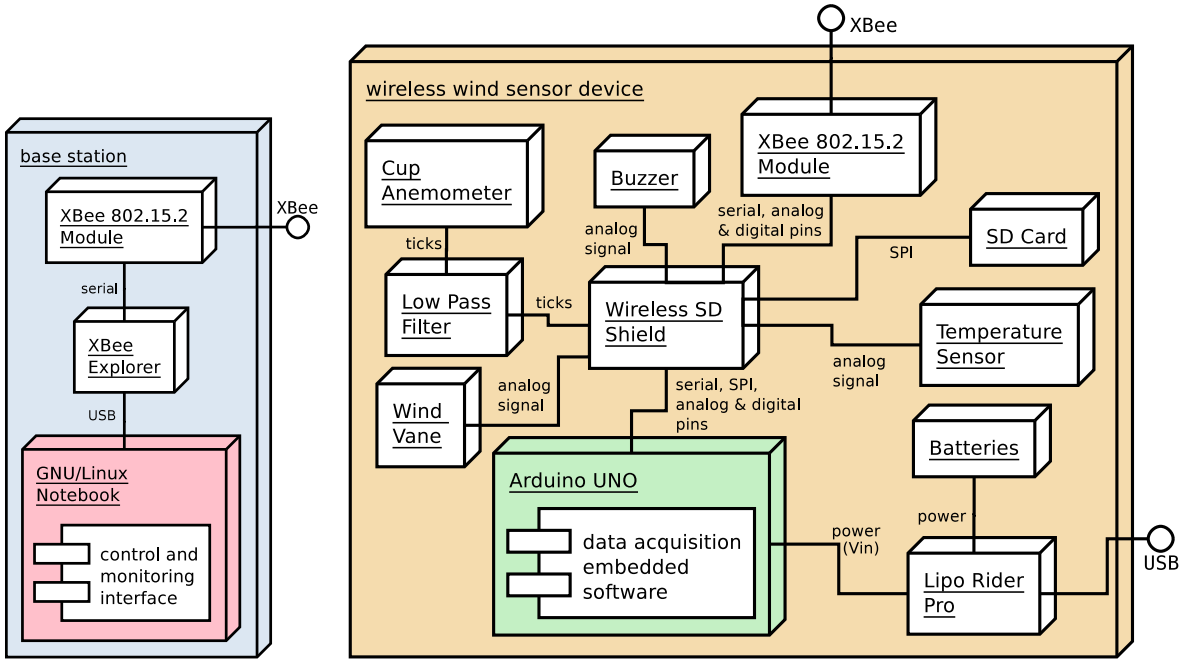
En este apartado se dará una vista de los componentes utilizados en la versión uno de este sistema. Se puede ver un diagrama de bloque del sistema v1 en la Figura 2.1 y de uso en la Figura 2.2.

Como componente central se utiliza un Arduino UNO. El Arduino UNO es una placa de desarrollo basada en el microcontrolador ATmega328P, diseñada para facilitar la creación de proyectos electrónicos interactivos. Es popular por su facilidad de uso, bajo costo, amplia comunidad y capacidad de programación a través del entorno de desarrollo Arduino IDE.

Además, se integran los siguientes periféricos:

Shield Wireless SD Arduino Un shield ya descontinuado. Los shields son placas que se integran modularmente con las placas de prototipado Arduino. En este caso este shield incorpora una bahía (slot) para almacenamiento secundario mediante el uso de tarjetas

Figura 2.1: Diagrama de bloque del sistema versión 1. Fuente: [5]



(a) Base station. **(b)** Wireless wind sensing device.

de memoria micro SD, y además incopora también la facilidad de conectar módulos de comunicaciones XBee creadas por Digi. [1]

Veleta y Anemómetro Para medir la intensidad y dirección del viento se utilizó una veleta y un anemómetro de copa. Estos dos componentes están muy estandarizados en la medición de las condiciones de viento. Esto permite que sean fáciles de conseguir e integrar con el resto del sistema en comparación con la creación de sensores únicos.

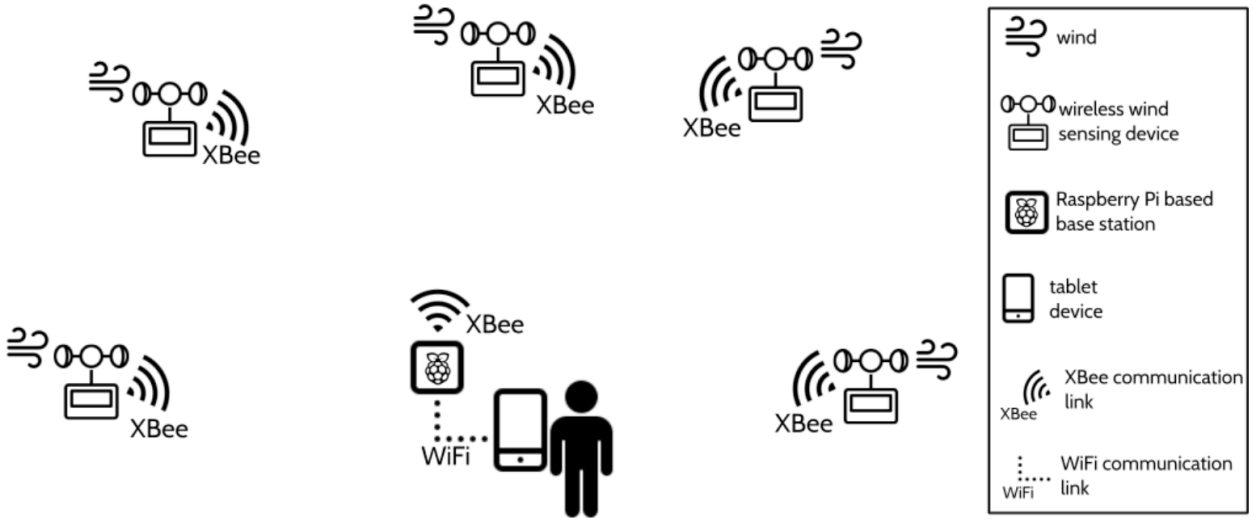
Zumbador Para avisar a los usuarios de cualquier error que surja en los dispositivos sensores durante un despliegue, éstos incorporan un zumbador. Distintos patrones de pitidos generados por el Zumbador indican distintos errores.

Sensor de Temperatura Para garantizar la integridad de los dispositivos durante una sesión de medida se incluyó un sensor de temperatura, dado que en los entornos típicos de despliegue las condiciones de temperatura podían ser muy altas. Permitiendo informar de cualquier condición de temperatura peligrosa.

2.1.2. Estación base

Aparte de los dispositivos sensores el sistema integra una estación base. Esta estación base se ocupa de dos tareas fundamentales. En primer lugar, se ocupa de sincronizar los relojes de los distintos dispositivos, una vez estos se han desplegado en campo y se va a iniciar una sesión de medida. En segundo lugar, se ocupa de informar a los investigadores de cualquier

Figura 2.2: Diagrama del despliegue del sistema versión 1. Fuente: [5]



incidencia que ocurra con los dispositivos.

A continuación se relacionan las características principales de la estación base.

- **Computadora base**: Raspberry Pi [10] corriendo GNU/Linux, equipada con un enlace de comunicación XBee [27, 8].
- **Módulo de comunicación**: XBee PRO S1 802.15.4 60mW [29, 28], con un alcance de línea de visión de aproximadamente 1.6 km.
- **Software**: Interfaz de control y monitoreo desarrollada en C++ usando GTK [20] y libxbee3 [11].
- **Interfaz de usuario**: Acceso al entorno gráfico del Raspberry Pi vía VNC [26] en una tableta móvil, facilitando su uso en campo.

2.1.3. Aplicación *meteo*

La aplicación *meteo* es la interfaz a través de la cual se monitoriza el estado del despliegue de los sensores en el campo. Permite ver todas las medidas actuales de los distintos dispositivos. Esta es la aplicación que se ejecuta en la estación base.

La aplicación *meteo* se dedica a recolectar estos datos y presentarlo de una forma intuitiva y de fácil acceso para el usuario. Esta interfaz gráfica de la estación base permite a los usuarios ver y controlar el estado de cada dispositivo registrado. Cada dispositivo aparece en la vista de datos de la interfaz mostrando gráficamente los datos del viento que la estación base está recibiendo. Al seleccionar un dispositivo en esta vista, se pueden ver los últimos minutos de datos acumulados para ese dispositivo específico. Además, la vista de datos proporciona tres indicadores LED virtuales para cada dispositivo, que muestran el estado de la

Figura 2.3: Captura de la aplicación *meteo*. Fuente [5]

señal/comunicación, la temperatura y la recepción de datos. Se puede ver la interfaz de esta aplicación en la Figura 2.3.

2.1.4. Sincronización de dispositivos

Con este sistema de sensores hacía falta sincronizar los relojes de los distintos dispositivos. Permitiendo así una correcta comparación y análisis de los datos recogidos. A continuación se describe este mecanismo de sincronización, más detalles en [5].

1. Sincronización Inicial:

Al inicio de cada sesión de adquisición de datos, todos los dispositivos de viento deben ser sincronizados con la estación base, una vez se han registrado conjuntamente con la estación base al finalizar el despliegue en campo, la estación base, equipada con un módulo de comunicación XBee, emite un paquete de sincronización a todos los dispositivos de viento.

2. Sincronización Periódica:

Para evitar el desajuste de los relojes internos de los dispositivos debido a la deriva del reloj (que puede ser significativa en dispositivos basados en Arduino UNO), se implementa un mecanismo de resincronización periódico. La estación base envía periódicamente paquetes de

señal de *Pulses Per Second* (PPS) a través del módulo XBee. Estos paquetes PPS permiten a cada dispositivo sincronizar su reloj interno con la estación base.

3. Consideraciones de Precisión:

La estación base emite los paquetes PPS con una periodicidad de 20 segundos. Esta periodicidad se ha determinado para mantener la sincronización con un error máximo de 100 milisegundos. Asegurando que el tiempo en los dispositivos se mantenga alineado con el reloj de la estación base dentro de un margen aceptable de error.

4. Procedimiento en Campo:

1. Durante la configuración inicial, los dispositivos son encendidos y ejecutan un procedimiento de calibración de la veleta, seguido por la sincronización del reloj con la estación base.
2. La estación base registra cada dispositivo conforme estos envían paquetes de registro después de la calibración y sincronización inicial.
3. Una vez registrados todos los dispositivos, la estación base inicia la sesión de adquisición de datos enviando el primer paquete de sincronización.
4. Durante la sesión de adquisición, los dispositivos continúan re-sincronizando sus relojes con los paquetes PPS periódicos emitidos por la estación base.

Beneficios de la Sincronización de Reloj:

- **Consistencia Temporal:** Asegura que todas las mediciones de viento estén alineadas temporalmente, permitiendo una comparación precisa de los datos recogidos.
- **Reducción de Errores:** Minimiza el riesgo de errores debido a la deriva del reloj interno de los dispositivos, mejorando la fiabilidad de los datos.
- **Simplificación del Análisis:** Facilita la correlación de datos entre múltiples dispositivos, mejorando la calidad del análisis de los patrones de viento.

La sincronización de reloj es esencial para mantener la integridad y precisión de los datos recogidos por el sistema de adquisición de datos de viento, asegurando que los estudios ambientales puedan realizarse manteniendo la coherencia temporal con respecto a los datos proporcionados por los dispositivos [5].

2.2. Objetivos de la nueva versión del sistema, versión 2(v2)

Uno de los requisitos principales durante el desarrollo de la primera versión del sistema (versión 1, v1) fue la disponibilidad de un presupuesto restringido para su desarrollo; así pues, ello conllevó al desarrollo basado en dispositivos sensores de costo asequible o bajo costo [12, 4]

En este trabajo se va a comenzar de una segunda versión del sistema (versión 2, v2) incorporando nuevos tipos de dispositivos al sistema equipados con mejores sensores de viento, así como con la posibilidad de incorporar sensores de transporte de arena basados en sensores de presencia láser.

En concreto se quiere desarrollar los siguientes puntos:

- Mejorar la distancia cubierta por la red de sensores. Añadir un sistema de comunicaciones de mayor alcance para cubrir distancias mayores.
- Actualización de los sensores de medición de viento. Introducción de sensores de viento ultrasonidos más capaces.
- Estudiar la introducción de nuevos sensores para la estimación del transporte de arena.
- Incorporar estas nuevas capacidades sin afectar las actuales capacidades de sistema. Permitiendo que los usuarios puedan utilizar dispositivos de la nueva versión, tanto con, o sin dispositivos de la versión anterior v1 durante los despliegues del sistema. Haciendo que ambas versiones puedan cooperar juntos y también funcionar independientemente.

Capítulo 3

Competencias específicas

3.1. CI9

> Capacidad de conocer, comprender y evaluar la estructura y arquitectura de lo computadores, así como los componentes básicos que los conforman.

La competencia CI9 ha sido desarrollada exhaustivamente a lo largo de este trabajo de fin de título. En particular, se ha demostrado la capacidad para conocer, comprender y evaluar la estructura y arquitectura de los computadores, así como los componentes básicos que los conforman. Esto se ha evidenciado en el diseño y configuración de un sistema distribuido de sensores basado en microcontroladores Arduino MKR, anemómetros ultrasónicos de viento y módulos de comunicación LoRa. Además, se ha realizado una evaluación crítica de las diferentes opciones de hardware disponibles, seleccionando aquellas que ofrecían la mejor relación costo-beneficio y un rendimiento óptimo para la captura de datos en entornos dunares. Este proceso ha implicado una serie de pruebas y ajustes para asegurar que todos los componentes funcionen de manera cohesiva y eficiente dentro del sistema.

3.2. CI11

> Conocimiento y aplicación de las características, funcionalidades y estructura de los sistemas distribuidos, las redes de computadores e Internet y diseñar e implementar aplicaciones basadas en ellas.

La competencia CI11 ha sido un pilar fundamental en el desarrollo de este proyecto, que ha requerido un conocimiento profundo y una aplicación práctica de las características, funcionalidades y estructura de los sistemas distribuidos, así como de las redes de computadores e Internet. Este conocimiento se ha aplicado en el diseño de una red de sensores distribuida,

utilizando la tecnología de comunicación LoRa para conectar múltiples sensores de viento y arena en un entorno dunar. La implementación de esta red ha implicado diseñar aplicaciones que puedan gestionar la transmisión y recepción de datos de manera eficiente y fiable. Se ha añadido funcionamiento a uns una aplicación centralizada, llamada *meteo*, que se ejecuta en un Raspberry Pi y utiliza una interfaz gráfica basada en GTK. Esta aplicación es capaz de recibir datos de múltiples nodos de sensores, procesarlos y mostrarlos en tiempo real, lo que permite al usuario monitorear las condiciones del entorno y el funcionamiento correcto del sistema durante los despliegues de manera efectiva.

3.3. CI14

> Conocimiento y aplicación de los principios fundamentales y técnicas básicas de la programación paralela, concurrente, distribuida y de tiempo real.

La competencia CI14 ha sido desarrollada mediante la aplicación de principios fundamentales y técnicas avanzadas de programación paralela, concurrente, distribuida y de tiempo real. Este proyecto ha requerido el diseño de un sistema que pueda manejar múltiples flujos de datos provenientes de diferentes sensores en tiempo real. La programación distribuida ha sido esencial para gestionar la comunicación entre los nodos de sensores y la estación central. Se ha utilizado la tecnología LoRa para asegurar que los datos se transmitan de manera eficiente y con baja latencia. La capacidad de aplicar técnicas de programación en tiempo real ha sido crucial para garantizar que los datos se procesen y almacenen de manera oportuna, permitiendo una monitorización continua y precisa de las condiciones ambientales en los entornos dunares.

Capítulo 4

Aportaciones

Este Trabajo de Fin de Grado ha realizado significativas aportaciones en la mejora de la red de sensores utilizada para la medición de la dirección e intensidad del viento en entornos dunares, así como en la cuantificación del transporte de arena. A continuación, se detallan las principales contribuciones del proyecto.

Se ha integrado en el sistema un sensor de viento ultrasónico de bajo consumo, el *Calypso Ultra-Low-Power Ultrasonic Wind Meter (ULP-STD)* [25, 21], que ofrece una mayor precisión en la medición de la velocidad y dirección del viento en comparación con los sensores anteriores basados en veletas y anemómetros de copa. Además, la inclusión de un contador de granos de arena permite cuantificar el número de partículas de arena transportadas por el viento, proporcionando datos valiosos para estudios geomorfológicos y ambientales. Estos avances permiten obtener datos más fiables y detallados, mejorando la calidad de los estudios y análisis.

En segundo lugar, se ha ampliado significativamente el alcance del sistema. La integración del sistema de comunicación LoRa permiten desplegar sensores en áreas mucho más amplias, pasando a una cobertura de varios kilómetros entre dispositivos. Esto amplía considerablemente el alcance y la utilidad de la red de sensores en estudios de campo, permitiendo un análisis más extenso y detallado de los entornos dunares. Ello significa la posibilidad de estudiar patrones y fenómenos en una escala mucho mayor.

Otro avance importante es la mayor robustez y autonomía del sistema. La implementación de la sincronización de dispositivos mediante GPS asegura que los nuevos nodos de sensores tengan una referencia temporal precisa y consistente, eliminando la necesidad de la sincronización inicial y periódica presente en la versión anterior.

El desarrollo de una arquitectura modular y escalable ha sido otra contribución clave. Se ha diseñado una arquitectura de sistema que permite la fácil integración y despliegue de nuevos sensores, así como la convivencia con los existentes. Esto facilita futuras expansiones y mejoras del sistema sin requerir rediseñaos significativos. La utilización de plataformas de desarrollo abiertas como Arduino y Raspberry Pi ha permitido crear un sistema flexible y adaptable, con una comunidad amplia que facilita el soporte y la evolución del proyecto.

Junto con esto se ha integrado un nuevo sistema de sensor para monitorizar el movimiento de arena. Para hacer esto se ha integrado un sensor láser que detecta cortes en el láser producidos por la presencia de distintos objetos. En concreto este sensor láser permite medir la presencia de partículas de arena. Estos datos proporcionan información acerca del movimiento de partículas de arena según las condiciones de viento en ese momento.

En resumen, este TFG ha logrado no solo expandir la red de sensores existente, sino también establecer un sistema robusto, preciso y expansible que contribuirá de manera sustancial a la captación de datos en entornos dunares. Los beneficios directos incluyen una mayor precisión en la recolección de datos, la capacidad de cubrir áreas de estudio más grandes, nuevos tipos de datos, una mayor fiabilidad y continuidad en la captura de datos, y una plataforma flexible y adaptable para futuras evoluciones del sistema.

Capítulo 5

Metodología

En este TFT se aplicó una metodología iterativa sobre cada nueva tecnología y sensor a integrar en el sistema.

Con cada tecnología a integrar primero se realizaba un proceso de análisis y familiarización. Este proceso también incluía pruebas con estas nuevas tecnologías para comprobar y aprender sobre su funcionamiento. Una vez familiarizado con una nueva tecnología se realizaba un diseño inicial y el desarrollo de su implementación. De forma contigua al desarrollo se realizaban pruebas comprobando el funcionamiento del trabajo desarrollado. Finalmente, tras integrar el nuevo sistema, se llevaban a cabo pruebas más extensivas, tanto sobre el nuevo sistema como sobre todo el sistema en general.

Para este TFT hay tres sistemas claves que se integraron de forma secuencial:

1. Creación del nuevo dispositivo sensor con enlace de comunicación LoRa e integrando el sensor de viento Calypso.
2. Integración del enlace de comunicación LoRa en la aplicación *meteo*.
3. Integración del sensor láser en el nuevo dispositivo sensor y en la estación base.

Teniendo esto en cuenta se puede describir el progreso de este TFT siguiendo los siguientes tares:

1. **Familiarización con el Arduino MKR**: El Arduino MKR es una placa de desarrollo basado sobre un microcontrolador ARM de 32 bits. Esta se diferencia de la placa utilizada en los dispositivos v1; el Arduino UNO, la cual utiliza un microcontrolador AVR, un microcontrolador de 8 bits y una menor capacidad computacional, al disponer además de menor memoria en el sistema, tanto para programas, como para datos.
2. **Familiarización con el sensor de viento ultrasónico Calypso**: En esta fase el objetivo era entender el funcionamiento del sensor de viento ultrasónico. Esto fue dificultado debido a la escasa documentación proporcionada por Calypso. Debido a esto se basó el estudio del comportamiento de este mediante la monitorización del canal UART del sensor, descrito en el apartado 7.1.1.2.

3. **Diseño y desarrollo del soporte para el sensor de viento en el Arduino MKR**: En esta etapa, se diseñó y desarrolló un soporte físico y lógico para integrar el sensor de viento con el Arduino MKR. Creando las interfaces necesarios para leer los datos correctamente a través del puerto UART conectado con el Arduino MKR.

4. **Diseño y desarrollo de la transmisión de datos del sensor de viento por LoRa**: En esta fase, se implementó la transmisión de los datos recolectados del sensor de viento utilizando la tecnología LoRa. Se diseñaron y desarrollaron los módulos de transmisión y recepción para asegurar que los datos pudieran ser enviados de manera eficiente y fiable a largas distancias. Para hacer esto se utilizaron dos Arduino MKRs, uno en el dispositivo sensor, que transmite, y otro en el sistema receptor, que funciona como periférico de la estación base.

5. **Familiarización con la aplicación *meteo***: Aquí, el objetivo fue comprender el funcionamiento y las capacidades de la aplicación *meteo*, que se utiliza para monitorizar y analizar los datos meteorológicos recolectados. Se exploraron sus funcionalidades y se evaluó cómo los datos del sensor de viento podían ser integrados en la aplicación.

6. **Diseño y desarrollo de la integración de los dispositivos LoRa en la estación base**: En esta etapa, se trabajó en la integración de los dispositivos LoRa en la aplicación *meteo*. Esto incluyó el desarrollo de interfaces y protocolos para asegurar que los datos transmitidos por LoRa pudieran ser recibidos, procesados y visualizados correctamente en la aplicación.

7. **Refactorización, actualización y prueba del sistema actual en su totalidad**: Una vez desarrollados todos los componentes, se procedió a realizar pruebas integrales del sistema completo. Esto incluyó la verificación de la correcta operación del sensor de viento, la transmisión de datos por LoRa y la integración con la aplicación *meteo*. Se identificaron y corrigieron errores y se realizaron los ajustes necesarios para optimizar el rendimiento del sistema.

8. **Familiarización con el sensor láser para medir el volumen de arena movido**: En esta fase, se investigó y comprendió el funcionamiento del sensor, un componente adicional a integrar en el sistema para ampliar sus capacidades de monitorización ambiental.

9. **Integración del sensor láser en el sistema**: Finalmente, se trabajó en la integración del sensor láser en el sistema existente. Esto incluyó el desarrollo del soporte físico y lógico necesario para leer los datos del sensor láser y transmitirlos a través de LoRa, así como la integración de estos nuevos datos en la aplicación *meteo* para su monitoreo y análisis.

Capítulo 6

Diseño

El nuevo sistema de sensores es una mejora y ampliación del sistema anterior. Utiliza nuevos componentes y nuevas tecnologías.

En este apartado se dará una vista general del diseño del nuevo sistema que se ha implementado. Los distintos componentes del sistema se describirán en detalle en sus respectivas secciones.

El propósito central del sistema que se está desarrollando es el obtener datos de los sensores y guardarlos. Debido a esto, el diseño llevado a cabo tiene el flujo de datos como elemento central. De qué sistemas se obtienen los datos, los procedimientos realizados sobre los datos y a dónde se mandan los datos.

Como entrada de datos se utiliza un sensor de viento, un módulo GPS, el reloj de tiempo del microcontrolador y un sensor láser que cuenta los granos de arena.

Como salida de datos se dispone de dos sistemas. Por un lado, cada nodo dispone de una tarjeta flash micro SD donde se escriben todos los datos de forma local. Cada nuevo nodo también viene integrado con un módulo LoRa. Utilizando este módulo se transmite el estado actual del sensor a la estación central.

Se puede ver en la Figura 6.1 un diagrama mostrando el diseño de los nodos del nuevo sistema de sensores.

El sensor de viento es clave en el funcionamiento de este sistema, cuyo objetivo principal es la monitorización de las condiciones del viento, influyendo directamente en el procesamiento y manejo de los datos. Cuando el sensor produce una nueva entrada de datos, se activa tanto la escritura de datos en la tarjeta SD como la transmisión a través de LoRa. Este impulso también desencadena la recopilación de los demás datos que el sistema monitoriza.

6.1. Sensor de viento

Para este nuevo diseño se ha utilizado un sensor de viento ultrasónico. Los anemómetros ultrasónicos ofrecen varios beneficios sobre los sensores de la versión 1 del sistema, que utiliza un anemómetro de copa mecánico más una veleta.

Con un anemómetro mecánico y veleta cualquier cambio de condición de viento tiene que físicamente mover estos sensores para que sea registrado por el sensor. Esto se hace muy notable cuando la intensidad del viento es muy bajo [3, 30]. En comparación los anemómetros ultrasónicos no sufren de este retardo de medición ni de una bajada de precisión a bajas intensidades de viento.

También el sensor Calypso ofrece beneficios en la interpretación y procesamiento de datos. Interpreta la entrada de datos el mismo, proporcionando de forma directa al usuario de las distintas medidas de dirección e intensidad del viento. Además, ofrece filtros y otros componentes de procesamiento adicionales, para reducir el ruido en la medida.

6.2. Sensor láser

Otro de los componentes fundamentales utilizado en este sistema es el sensor láser proporcionado por Wenglor "Wenglor fork sensor" [9]. Este sensor láser es un sensor de transporte de arena.

Cuando el láser del sensor se interrumpe por la presencia de un objeto, la señal de salida se pone a un valor lógico alto. En caso contrario, presenta un valor lógico bajo. Utilizando esto se puede contar el número de flancos de subida o bajada y se puede averiguar cuantas veces ha sido cortado el sensor láser.

Con este funcionamiento cada vez que un grano de arena atraviesa el láser del sensor se produce un corte de este, permitiendo contar así el número de granos de arena que atraviesan el sensor. Para contar estos flancos se utiliza el servicio de interrupción del microcontrolador, aumentando un contador cada vez que se recibe un flanco de subida.

6.3. Módulo GPS

En el capítulo 2.1.4 se explicó que los dispositivos de la versión uno requieren un esquema de sincronización de tiempo entre las unidades. Con los nuevos dispositivos se utilizan módulos GPS para conseguir el tiempo actual. Así cada dispositivo se sincroniza directamente a partir del módulo GPS sin depender de un esquema de comunicación entre dispositivos.

Esto permite que cada unidad opere de forma independiente sin necesitar depender de un proceso de sincronización con la estación base.

6.4. Almacenamiento

Al igual que en los dispositivos v1 los datos recogidos por los nuevos dispositivos sensores se guardan localmente en el sistema en una tarjeta micro SD. Cuando el usuario quiere recoger los datos de todos los ensayos realizados por el sistema se accede a las tarjetas micro SD de cada dispositivo y se copia la información almacenada en sus ficheros log.

A partir de los ficheros de log se generan ficheros CSV que contienen la información proporcionada por cada sensor.

6.5. Integración del protocolo de comunicación LoRa

6.5.1. Comparación de distintos protocolos

En esta sección se comparan los protocolos Zigbee y LoRa, destacando sus características, ventajas y limitaciones.

6.5.1.1. Zigbee

Zigbee es un protocolo de comunicación inalámbrica basado en el estándar IEEE 802.15.4, conocido por su baja latencia y consumo de energía eficiente.

- **Alcance**: Zigbee tiene un alcance típico de hasta 1,6 kilómetros en línea de vista.
- **Topología**: Soporta varias topologías de red, incluyendo estrella, árbol y malla.
- **Consumo energético**: Ideal para dispositivos de bajo consumo, como sensores y actuadores con baterías.
- **Frecuencia de operación**: Opera principalmente en la banda ISM de 2.4 GHz.
- **Latencia**: Baja latencia, adecuada para aplicaciones que requieren respuestas rápidas.
- **Capacidad de datos**: Tasa de transferencia de datos de hasta 250 kbps.

Limitaciones de Zigbee:

- **Alcance limitado**: No adecuado para áreas extensas, teniendo como alcance máximo de 1,6 kilómetros.
- **Interferencias**: La banda de 2.4 GHz, utilizada por Zigbee, puede tener interferencias de otros dispositivos.

6.5.1.2. LoRa

LoRa (Long Range) es una tecnología de modulación de espectro expandido para comunicaciones a larga distancia con bajo consumo energético, diseñada para aplicaciones IoT.

- **Alcance**: LoRa puede cubrir distancias de hasta 15 kilómetros en áreas rurales y varios kilómetros en entornos urbanos.
- **Topología**: Utiliza una topología de estrella, y comunicaciones punto a punto.
- **Consumo energético**: Muy bajo, permitiendo que los dispositivos funcionen durante varios años con una sola batería.
- **Frecuencia de operación**: Utiliza bandas sub-GHz no licenciadas (868 MHz en Europa y 915 MHz en América). Estas bandas están sujetas a una limitación de uso explicado en más detalle en el capítulo 7.2.2 y A.1.
- **Latencia**: Generalmente mayor que Zigbee.
- **Capacidad de datos**: Tasa de transferencia de datos de hasta 50 kbps.

Ventajas de LoRa:

- **Cobertura extensa**: Proporciona un mayor alcance, llegando hasta 15 km en línea de visión.
- **Interferencias reducidas**: Menos interferencias en comparación con la banda de 2.4 GHz.

Limitaciones de LoRa:

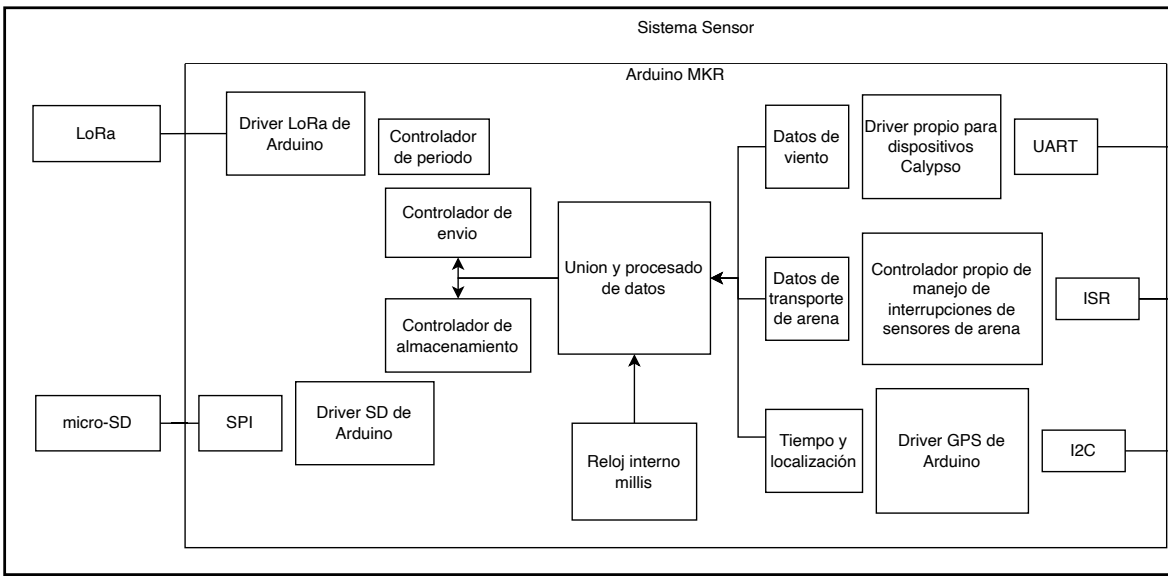
- **Latencia alta**: No adecuada para aplicaciones que requieren respuestas inmediatas.
- **Baja capacidad de datos**: Restricciones en aplicaciones que necesitan transmitir grandes volúmenes de información.

6.5.2. Resultado final

Debido al deseo de cubrir áreas mayores durante los despliegues se incorporó el protocolo LoRa para los dispositivos de la nueva versión 2 a desarrollar.

Con esto el nuevo sistema hereda la desventaja principal de LoRa, su alta latencia; pero también sus ventajas. Con esta latencia se consigue actualizar la estación central del estado actual de cada sistema cada 25 segundos aproximadamente para los nuevos dispositivos v2.

Figura 6.1: Diagrama de bloque del nuevo diseño del sistema de sensor. Elaboración propia.



Capítulo 7

Desarrollo

7.1. Sensor de viento Calypso

Para el desarrollo del sistema se utilizará el sensor de viento "Calypso Ultra-Low-Power Ultrasonic Wind Meter (ULP-STD)" en su modelo UART [25]. Como indica el nombre, este es un sensor de ultrasonido, que como medidas proporciona la dirección y velocidad del viento.

Se decidió utilizar este sensor debido a su bajo coste de precio en comparación con otros anemómetros ultrasónicos, su facilidad de utilización y la mejora en prestaciones en comparación con el sistema anterior de medición de viento.

7.1.1. Configuración del sensor de viento

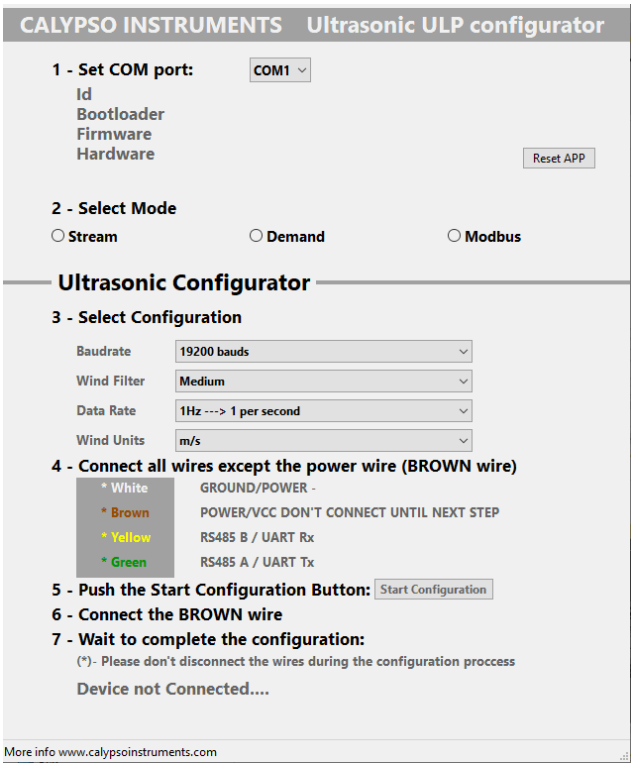
Calypso para configurar sus sensores proporciona un programa de configuración para ordenadores Windows [21, 13]. Esta aplicación muestra al usuario distintas opciones para configurar el sensor (ver figura 7.1). Una vez seleccionado la configuración deseada el programa carga y guarda la configuración en el propio sensor.

El razonamiento utilizado por Calypso es que el usuario guarda su configuración una sola vez en el sensor. Posteriormente, cuando se quiera utilizar el sensor desde algún equipo el usuario únicamente tendría que "encender" el sensor.

Pero para el caso de uso del sistema a desarrollar, va a haber muchos sistemas utilizando estos sensores. Donde se debe garantizar que todos están configurados con los mismos parámetros.

Para realizar esta configuración se intentó crear una API para dispositivos Arduino que permita una fácil configuración de los distintos parámetros de los sensores Calypso. Esto facilitaría la configuración, garantizando que todos los sensores tienen la misma configuración. Esto además facilitaría la utilización futura de estos sensores y sus cambios de configuración incluso durante el despliegue del sistema.

Figura 7.1: Captura de la aplicación utilizada para configurar sensor de viento Calypso. Fuente: [13].



Este intento de configurar el sensor de viento durante la inicialización del sistema finalmente se encontró con problemas que inhibieron su inclusión en este proyecto.

Gran parte de la dificultad de intentar configurar este sensor utilizando el propio dispositivo fue la falta de información y documentación alrededor de como configurar el sensor Calypso. Debido a eso en este apartado se describirá los conocimientos adquiridos y trabajo realizado acerca de la configuración de los sensores Calypso.

7.1.1.1. Cómo se configura el sensor Calypso

La configuración del sensor es el único elemento de comunicación con el sensor de viento que no sigue el estándar NMEA 0183. La configuración se manda en texto plano sin checksum ni identificación de dispositivo. Podemos ver un ejemplo de esto en el Listado 7.1.

El sensor Calypso no incluye información de sus registros de configuración en su datasheet ni está disponible ningún manual que indica su configuración.

Para realizar esta configuración se realizó una API para dispositivos que permite una fácil configuración de los distintos parámetros de configuración del sensor.

7.1.1.2. Metodología de recogida de datos

Para poder ver como se configura, se utilizó un Arduino DUE interpuesto como *middleman* entre el sensor y la aplicación de configuración. Capturando todo lo que recibía por una de sus entradas seriales, guardándolo, y transmitiéndolo por el otro canal serie. Este es el método con el que se obtuvo el Listado 7.1.

Una vez capturada la salida bruta por las dos canales UART (TX del sensor y TX de la aplicación de configuración) se creó un script de python para limpiar y aumentar la legibilidad de la salida.

Gracias a esto se pudo deducir la existencia y comportamiento de distintos registros.

7.1.1.3. Modo del sensor

Para la lectura de datos, los sensores de vientos Calypso habilitan dos formas de obtenerlos. Se puede inicializar el sensor en modo polling o en modo stream.

En el modo polling se tiene que solicitar el dato al sensor que devuelve el dato solicitado bajo demanda.

Mientras que en el modo stream se especifica la frecuencia de medición durante la configuración del sensor Calypso. Con esto, el sensor mide con la periodicidad configurada, enviando por el canal serie los resultados de cada medición sin solicitud previa.

```
RX: $START_ULTRA_CALYPSO
RX: $START_CONFIG
RX: $ULTRAMODE_02*00
RX: $BAUDRATE_019200*45
RX: $FILTER_MEDIAN_03*09
RX: $FILTER_REPETICION_01*15
RX: $FILTER_DAMPING_00*58
RX: $DATARATE_04,00*65
RX: $NMEA_UNITS_K*19
RX: $STOP_CONFIG
RX: MODE NMEA STREAM
```

Listado 7.1: Ejemplo de configuración de los registros del sensor de viento Calypso. Elaboración propia

7.1.2. Como se comunican los datos

Para la comunicación sobre el canal serie todos los datos van codificados utilizando el estándar NMEA 0183 [23, 17].

NMEA 0183 establece una estructura de mensaje para mandar sobre la UART. Esto permite una forma de comunicación de datos estándar a través de distintos proveedores de dispositivos.

Arduino no proporciona una biblioteca para parsear este tipo de mensajes. Por lo tanto para interaccionar con el sensor Calypso se creó una serie de funciones para interpretar paquetes NMEA 0183.

7.1.2.1. Formato de los datos. NMEA 0183

En el Listado 7.2 se muestra una línea de datos enviada por el sensor de viento [23].

Como se puede observar en la parte de ID del protocolo NMEA 0183 los sensores de viento llevan el identificativo "MWV". Esto representa el identificativo «Wind speed and angle». NMEA 0183 utiliza identificativos de sentencias para tener un formato de datos estandarizado. Este formato de sentencia actualmente se considera obsoleto [24, 18].

El formato de sentencia MWV de NMEA 0183 contiene los siguientes registros:

- 023: Angulo del viento(0 a 360 grados)
- R: Angulo relativo(R) o absoluto(T)
- 000.0: Velocidad del viento
- K: Unidad de medición Km/h(K), m/s(M) y nudos(N)
- A: Estado. A índica que todas las mediciones fueron válidas.

```
$IIMWV,023,R,000.0,K,A*27
```

Listado 7.2: Ejemplo de respuesta de datos del sensor Calypso en formato NMEA 0183 MWV

En el dispositivo sensor, en cada iteración del código se comprueba si existe alguna entrada disponible en el puerto serial del sensor de viento. Si ese es el caso se lee la entrada serial hasta que se detecta un salto de línea y se interpreta la línea de datos descomponiéndola en sus distintos elementos.

7.1.2.2. Desarrollo de una biblioteca NMEA 0183

La biblioteca creada para interpretar mensajes NMEA 0183 tiene una funcionalidad limitada a las necesidades del sistema creado. Permite validar mensajes entrantes y mandar mensajes.

Para ello se calcula un checksum del cuerpo del mensaje. El cálculo del checksum consiste en realizar un XOR entre cada byte del mensaje de forma secuencial. El código utilizado para calcular el checksum se puede ver en la Listado 7.4

En caso de querer validar un mensaje recibido, se calcula el checksum del mensaje y se comprueba que corresponde al checksum incluido en el mensaje. En caso de querer enviar un mensaje, se calcula el checksum y se añade al mensaje.

```
namespace nmea0183
{
    byte checksum(const char *s);
    bool validate(const char *s);

    void send_command(Stream &serial, const char *s);
}
```

Listado 7.3: Interfaz de las utilidades NMEA 0183

```
byte nmea0183::checksum(const char *s)
{
    byte c = 0;
    for (int i = 0; s[i] != '*' && s[i] != 0; i++)
    {
        c ^= s[i];
    }
    return c;
}
```

Listado 7.4: Calculo del checksum de NMEA 0183

7.1.3. Implementación

Para nuestro sistema, continuando con el modelo basado en interrupciones, adoptamos el modo stream. Este ajuste permite que nuestro sistema opere centrado en el sensor de viento. Cada vez que este sensor entrega nuevos datos, consultamos otros sensores y dispositivos del sistema; posteriormente, enviamos una actualización a la estación base cuando es pertinente.

7.1.3.1. Lector de datos

Al estar el sensor configurado en modo stream, el microcontrolador Arduino se limita a escuchar por el canal serial a la espera de recibir mensajes desde el sensor.

Durante la iteración principal del Arduino MKR se comprueba si hay entradas de datos en el registro serial asociado con el sensor de viento. Arduino proporciona la función `Serial.available()` para fácilmente comprobar esto. En caso de que haya una línea de texto se lee hasta que se vacíe el registro. Con el registro leído, se valida que la entrada tiene formato NMEA 0183.

También se comprueba que la entrada de serial viene desde un dispositivo de viento siguiendo el formato de sentencia MWV de NMEA 0183. Esto se hace comprobando que en el apartado de ID de la entrada están los caracteres “MWV”.

Con todas estas comprobaciones ya realizadas se parsean los datos contenidos en la entrada. Se convierten a la estructura `ulp_sensor_data` mostrado en el Figura 7.2. Con esta entrada de datos el resto del programa puede llevar su curso utilizando los datos recibidos.

Figura 7.2: Interfaz Calypso Arduino

```
typedef struct ulp_sensor_data
{
    wind_angle_t wind_angle;
    wind_reference_t wind_reference;
    wind_speed_t wind_speed;
    wind_speed_units_t wind_speed_units;
    bool valid;
} ulp_sensor_data_t;

void initialize_ulp_sensor(Stream &stream);

bool read_ulp_sensor(Stream &stream, ulp_sensor_data_t &data);

const char *wind_speed_units_to_string(wind_speed_units_t units);
```

7.1.3.2. Código en el proyecto

En la Figura 7.2 se puede ver la interfaz del módulo del sensor del viento Calypso.

Utilizando esta interfaz nuestra application cada iteración comprueba si hay una nueva entrada utilizando la función `read_ulp_sensor`. Si hay una nueva entrada devuelve `true` y escribe el resultado final en el parametro `data`. Si no devuelve `false`.

7.2. Comunicación mediante LoRa

El proyecto actual dispone de una estación base, para la monitorización de los dispositivos sensores. Como se explicó en el apartado 6.5 se quiere utilizar LoRa con el objetivo de cubrir áreas más extensas manteniendo la comunicación con la base central. En este apartado se explicará la utilización e integración de LoRa en la nueva versión del sistema.

7.2.1. Hardware utilizado

Para este proyecto se utiliza un Arduino MKR WAN 1310 [16]. Este es una placa que contiene un microcontrolador ARM Cortex M0+ SAMD21 y un módulo LoRa de Murata.

Es capaz de transmitir tanto con las frecuencias LoRa europeas y americanas.

Al venir el módulo integrado en la misma placa, se consigue acceso al sistema LoRa por un coste menor.

7.2.2. Limitaciónes legales. El duty cycle

7.2.2.1. Que es el duty cycle

El duty cycle es un concepto general relacionado con señales. Describe el porcentaje del periodo que está activo una señal[6].

Cuando hablamos de LoRa este duty cycle se puede interpretar como el ratio de tiempo que estamos transmitiendo contra el que no estamos transmitiendo.

$$\text{Duty cycle} = \frac{\text{tiempo transmitiendo}}{\text{tiempo transmitiendo} + \text{tiempo no transmitiendo}}$$

7.2.2.2. Implementación en el código

Al crear este sistema de transmisión se debe respetar la normativa legal en vigor. Esto se divide en dos pasos importantes.

En primer lugar, el transceptor LoRa debe estar correctamente configurado para utilizar una frecuencia, potencia y ancho de banda adecuado. Esto se configura durante la inicialización del microcontrolador utilizando la biblioteca proporcionada por Arduino.

En segundo lugar, se debe garantizar que el periodo de transmisión respeta el duty cycle establecido para la frecuencia utilizada. Para hacer esto se debe garantizar que el tiempo de transmisión ocupa menos de un 1 % del periodo de transmisión, que se corresponde con un duty cycle del 1 %.

Para hacer esto dentro del módulo LoRa se establece un controlador de transmisión. Este controlador lleva a cabo dos tareas fundamentales. El primero es medir y vigilar el tiempo de transmisión, si el tiempo de transmisión excede el duty cycle el controlador aumenta el periodo de transmisión. Este periodo de transmisión se aumenta un 50 % cada vez que se excede las limitaciones de duty cycle. La segunda responsabilidad de este controlador es asegurar que se respete el periodo de transmisión. Esto se consigue prohibiendo cualquier intento de transmisión que ocurra antes de que el tiempo del periodo desde la última transmisión que se cumpla.

7.2.3. Estructura del paquete LoRa

En este proyecto solo se utiliza la capa física OSI, LoRa y no la de enlace de datos proporcionado por LoRaWAN [22]. Debido a esto se tiene que definir y establecer una estructura propia del paquete al transmitir.

La estructura del paquete tiene dos componentes principales. El header que contiene los metadatos del paquete, y la trama de datos.

El header contiene los siguientes elementos:

- ID del destinatario
- ID del transmisor
- ID del mensaje
- Longitud del paquete de datos

Este suele ser una estructura muy usual al utilizar LoRa de forma directa.

Además, en este proyecto los datos a transmitir van a tener sólo un tipo de mensajes. Por lo tanto, solo existe una única configuración para el paquete de datos.

A continuación se muestra la estructura del paquete de datos:

- Ángulo del viento en grados (2 bytes)
- Velocidad del viento en metros por segundo (2 bytes)
- Temperatura interna del microcontrolador en grados centígrados (2 bytes)
- Hora de medición epoch (segundos). Segundos transcuridos desde el primero de enero 1970 (4 bytes)
- Número de sensores de láser (1 byte)
- Por cada sensor láser, su valor (4 bytes)

7.2.4. Implementación del módulo LoRa

En la Figura 7.3 se puede ver la interfaz del módulo LoRa. Esta interfaz facilita que desde el resto del sistema se pueda realizar la comunicación entre dispositivo sensor y estación base.

Tenemos dos funciones básicas utilizadas tanto en los dispositivos transmisores como en el receptor. Estas son `initialize_LoRa` y `lora_loop`.

Con `initialize_LoRa` configuramos los parámetros de transmisión del módulo LoRa. Esta configuración inicial permite ajustar el módulo a las condiciones específicas del entorno y a los requisitos reglamentarios del área de operación. Por ejemplo, ajustar el 'spreadingFactor' a un valor alto aumenta la distancia y la robustez de la señal, pero también reduce la tasa de transferencia de datos. Este tipo de configuraciones es importante para optimizar el rendimiento del módulo en diferentes escenarios de aplicación.

`lora_loop` debe ser llamado de forma frecuente por el usuario desde el hilo principal del programa para permitir un correcto funcionamiento del sistema LoRa. Esta función realiza tareas fundamentales para controlar la integridad de cada transmisión y recepción; además juega un papel fundamental para garantizar el control del duty cycle. Dentro de los dispositivos se llama esta función durantre cada iteración de código del Arduino MKR.

Junto a estas dos funciones disponemos de las funciones `can_send_package`, `read_lora_message` y `send_lora_message`.

Figura 7.3: Interfaz LoRa Arduino

```
typedef struct
{
    uint8_t bandwidth_index; // ANCHO DE BANDA
    uint8_t spreadingFactor; // FACTOR DE PROPAGACION
    uint8_t codingRate;      // TASA DE CODIFICACION
    uint8_t txPower;         // POTENCIA DE TRANSMISION
} LoRaConfig_t;

struct LoRaPackage
{
    ulp_sensor_data_t data;
    uint32_t time;
    float temperature;
    std::vector<uint32_t> sand_sensor_values;
};

void initialize_LoRa(const LoRaConfig_t &thisNodeConf);
void lora_loop();

bool can_send_package();

bool read_lora_message(struct LoRaPackage &lora_package, uint8_t &device_id);

void send_lora_message(const struct LoRaPackage &lora_package);
```

Con la función `can_send_package` podemos comprobar el estado del módulo LoRa, en el caso de que el módulo LoRa actualmente esté transmitiendo o esté bajo las limitaciones de duty cycle indicará al usuario que actualmente no se puede enviar ningún paquete.

La función `read_lora_message` se encarga de recibir y decodificar los mensajes que llegan a través del módulo LoRa. Esta función toma como argumentos una referencia a una estructura `LoRaPackage`, que almacenará los datos recibidos, y una variable `uint8_t` para el identificador del dispositivo que envía el mensaje. Si un mensaje es recibido y decodificado correctamente, la función devuelve `true`, permitiendo al sistema procesar adecuadamente los datos recibidos. Esto es especialmente útil en aplicaciones donde es crucial recibir y responder a los datos de manera oportuna.

Por último, la función `send_lora_message` facilita el envío de mensajes a través del módulo LoRa. Esta función acepta un argumento que es una referencia constante a una estructura `LoRaPackage`, que contiene todos los datos a transmitir. La función se asegura de que el mensaje se formatee correctamente y se envíe respetando los parámetros de configuración del módulo LoRa, como el ancho de banda, el factor de propagación, la tasa de codificación y la potencia de transmisión configurados en la inicialización. Esto asegura que los datos se transmitan de manera eficiente y conforme a las normas reguladoras.

Estas funciones trabajan en conjunto para gestionar la comunicación a través del módulo LoRa, asegurando que el sistema pueda enviar y recibir datos de manera fiable y eficiente, respetando las restricciones impuestas por la regulación del duty cycle y las características técnicas del módulo.

7.3. Almacenamiento

Aunque nuestro sistema dispone de un sistema de envío de datos mediante LoRa, descrito en el capítulo 7.2. Este no puede enviar los datos a la estación base con suficiente frecuencia como para que sea factible utilizar un sistema de guardado centralizado.

Debido a esto, para guardar los datos recogidos para su posterior análisis se utiliza una tarjeta micro SD en cada dispositivo como almacenamiento secundario local.

7.3.1. Ventajas y desventajas del almacenamiento local

El uso de tarjetas SD para el almacenamiento local de datos presenta varias ventajas y desventajas:

7.3.1.1. Ventajas

- **Disponibilidad local de datos**: Permite almacenar grandes volúmenes de datos directamente en el dispositivo, asegurando que los datos estén siempre accesibles localmente y a una mayor frecuencia de muestreo.
- **Reducción de dependencia de conectividad**: No requiere una conexión constante para transmitir los datos ni la frecuencia de datos está limitado por el duty cycle de LoRa.

7.3.1.2. Desventajas

- **Riesgo de pérdida de datos**: En caso de fallo de la tarjeta micro SD, los datos almacenados pueden perderse o corromperse.
- **Necesidad de mantenimiento manual**: Es necesario extraer manualmente los datos de la tarjeta micro SD para su análisis.

7.3.2. Formato de los datos

Utilizando esta tarjeta micro SD se guardan en ficheros de log todas las mediciones realizadas, localizada en el almacenamiento secundario.

A partir de los ficheros que se guardan se generan ficheros con un formato CSV que se utilizan para la interpretación y procesamiento de los datos.

Este fichero se nombra basándose en el ID del dispositivo con el formato siguiente. `"data"+ std::to_string(DEVICE_ID) + ".csv"`

La estructura de este fichero CSV es la siguiente:

- Tiempo en milisegundos del microcontrolador.
- Tiempo en segundos del módulo GPS.
- Longitud del módulo GPS.
- Latitud del módulo GPS.
- Velocidad del viento.
- Ángulo del viento en grados.
- Unidad de velocidad del viento.
- Referencia de la dirección de viento.
- Número de sensores de arena Wenglor.
- Una entrada por cada sensor de arena: Número de interrupciones.

7.3.3. Características de la tarjeta SD

Para conectar la tarjeta micro SD a nuestro microcontrolador se utiliza un shield de prototipado para un Arduino MKR que tiene un puerto micro SD.

Arduino proporciona una biblioteca para interactuar con este módulo shield. Utilizando esta biblioteca se consigue un acceso fácil y directo con la tarjeta micro SD permitiendo guardar datos y utilizar la micro SD como memoria secundaria.

7.4. Sincronización de dispositivos

Para poder interpretar correctamente los datos recogidos por los dispositivos sensores es importante considerar la dimensión del tiempo. La utilidad de los datos muestreados sería muy limitado si no se crea un sistema para sincronizar el tiempo de los datos entre los distintos dispositivos.

La red de sensores versión uno requiere que todos los dispositivos v1 se hayan registrado con la estación base para que ésta envíe un mensaje de sincronización inicial. Así todos los nodos v1 ajustan su reloj interno con el momento en el que reciben el mensaje de sincronización de la estación base (ver apartado 2.1.4).

Si cualquiera de los dispositivos v1 no estuviera encendido o no recibieran el paquete de inicialización se tendría que volver a realizar el procedimiento de inicialización.

Para evitar este sistema de sincronización con los dispositivos v2 se ha incorporado un receptor GPS para recibir el tiempo actual. Así todos los dispositivos de la versión dos de este proyecto dispondrán del tiempo actual de forma independiente entre ellos.

7.4.1. Beneficios de utilizar un receptor GPS

Los módulos GPS reciben y proporcionan el tiempo actual de forma muy precisa. Esto permite que un sistema con módulo GPS pueda tener un seguimiento muy fiable del tiempo actual.

Utilizar esto en la nueva versión de los dispositivos permite que estos no requieran de un esquema de sincronización con la estación base.

Por otro lado, además de poder conseguir el tiempo el sistema GPS, este también permite obtener la ubicación actual del dispositivo. Aunque esta utilidad actualmente no forma parte de los requisitos del proyecto, también es un dato que se recoge y se guarda. Pudiendo esto ser útil tanto para el proyecto actual como para algún otro proyecto futuro.

7.4.2. Detalles de los recursos utilizados

7.4.2.1. Hardware

El shield de GPS de Arduino([15]) soporta dos modos de comunicación, I2C y UART. Disponiendo además de un conector dedicado de I2C.

La intención detrás de esto es que mientras se esté utilizando la placa en modo shield (con los pines de la placa GPS conectados con la placa MKR) estén directamente conectados los pines correspondientes a UART y, por lo tanto, se utilizan estos para la comunicación.

Mientras que cuando se esté utilizando la placa GPS de forma desconectado se utiliza los conectores I2C para conectar la placa GPS con la placa MKR.

7.4.2.2. Software

Para la implementación del shield GPS en el código se utilizó la biblioteca `Arduino_MKRGPS` [2]. Esta biblioteca permite configurar el módulo GPS y leer los datos de este. Proporciona una interfaz que permite solicitar datos al módulo GPS.

7.4.3. Implementación - Desarrollo

Para acceder a los datos GPS se creó un módulo encapsulando el funcionamiento del GPS. Podemos ver la interfaz de este módulo en la Figura 7.5. Esta interfaz proporciona 4 funciones al usuario, 2 de control y 2 de obtención de datos.

Estas 4 funciones y sus características se detallan a continuación:

`initialize_gps_sensor` se debe llamar una sola vez antes de empezar a utilizar el módulo GPS. Este se ocupa de configurar la biblioteca y dispositivo GPS.

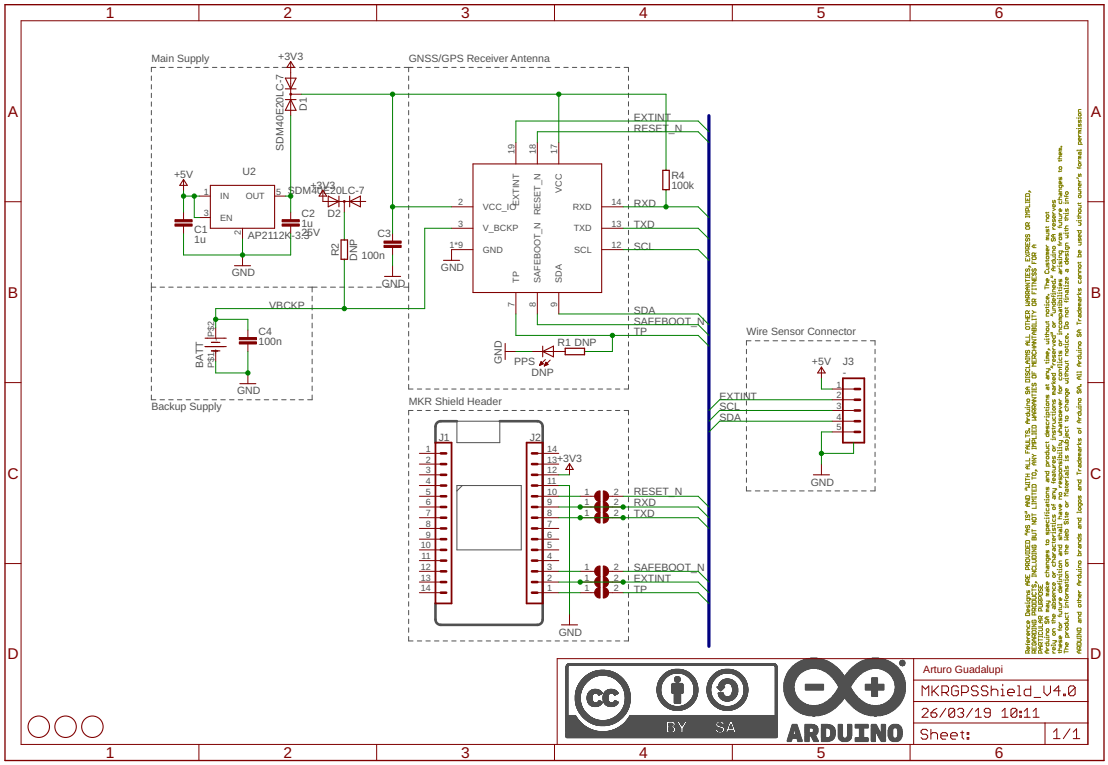


Figura 7.4: Esquemática del shield GPS del Arduino MKR. Fuente: [15].

`gps_loop` se debe llamar de forma frecuente para mantener activo el módulo GPS. Dentro del código se llama esta función en cada iteración del Arduino MKR. El dispositivo GPS de la placa en modo I2C está configurado para funcionar en modo polling. Esto significa que únicamente se solicitan los datos GPS bajo demanda. Si estos datos GPS se solicitan con poca frecuencia el módulo GPS se "enfriará", haciendo que no pueda proporcionar los datos GPS al serle solicitados. Por lo tanto, esta función se dedica a mantener el módulo GPS "caliente", permitiendo que el usuario pueda solicitar y obtener datos GPS en cualquier momento.

`read_gps_time_if_available` y `read_gps_position_if_available` son las responsables de conseguir los datos del módulo GPS. Si el dato está disponible se devuelve verdadero y se asigna el valor al parámetro pasado por referencia correspondiente. Si no está disponible el dato se devuelve falso y no se asigna ningún valor a los parámetros pasados por referencia.

`read_gps_time_if_available` proporciona la época actual en segundos a través del módulo GPS. Esto corresponde al tiempo Unix, que representa la cantidad de tiempo que ha pasado desde la medianoche UTC del 1 de enero de 1970.

Figura 7.5: Interfaz GPS Arduino

```
bool initialize_gps_sensor();

bool read_gps_time_if_available(uint32_t &epoch_time);

bool read_gps_position_if_available(float &longitude, float &latitude);

void gps_loop();
```

7.4.3.1. Interferencia del módulo GPS con la conexión UART del sensor de viento

Debido a que el microcontrolador MKR solo dispone de una interfaz UART, utilizado por el sensor de viento, se tuvo que utilizar la placa MKR en modo I2C.

Como se ve en la Listado 7.4 los puertos UART del módulo GPS están conectados con los pines UART de la placa GPS. Esto permite que cuando la placa este montado encima de la placa del microcontrolador podemos comunicarnos mediante UART. Mientras que por el otro lado mirando la misma figura se puede observar que la salida I2C del módulo GPS no va conectado a los pines de la placa, sino que la placa GPS dispone de un conector al que se conecta la salida I2C mediante cable.

Como se va a utilizar el modo I2C se conectó la placa mediante el connector I2C. Inicialmente, para ahorrar espacio y facilitar el manejo del dispositivo se incorporó la placa GPS conectándolo encima del microcontrolador pero todavía utilizándolo en modo I2C.

Al hacer esto se descubrió que aunque se configure la biblioteca GPS para utilizar el módulo GPS por I2C, este todavía sigue transmitiendo datos por su puerto UART.

Esto causa que cuando el módulo GPS esté directamente conectado a la placa MKR introduce ruido al puerto UART haciendo ilegibles los datos transmitidos por el sensor de viento.

Para solventar este fallo durante el despliegue del sistema se plantea dos soluciones. La primera de ellas es mantener la placa GPS desconectada y separada de la placa MKR. Esto aumentaría la superficie y espacio ocupado por la electrónica del sistema. La segunda solución propuesta sería cortar los pines UART de la placa GPS. Así al conectar la placa GPS con la placa MKR el módulo GPS no estará conectado el puerto UART del módulo GPS con el microcontrolador.

7.5. Sensor de transporte

Para la nueva versión de este sistema se realizó la integración de un nuevo componente, un sensor láser para contabilizar granos de arena. Este sensor permite ir midiendo el movimiento

Figura 7.6: Imagen del sensor láser de Wenglor. Imagen proporcionada por Wenglor.

de arena que ocurre en cada estación y poder relacionarlo con las condiciones de viento en cada momento.

Para ello se va a utilizar un sensor láser fabricado por Wenglor [9]. Este sensor consiste en un láser para detectar presencia. Cuando el haz láser no se interrumpe con un objeto, el sensor mantiene la salida de datos a un voltaje bajo, correspondiente a un 0 lógico. Mientras que si interrumpe el haz láser se proporciona un voltaje alto en la salida de datos, correspondiente a un 1 lógico.

Teniendo esto en cuenta se podría deducir que entre más pulsaciones ocurren más granos de arena han pasado por el sensor láser.

Estos sensores tienen algunas desventajas. En primer lugar, disponen de una área de captación limitada. También tienen una limitación de precisión. Si varios granos de arena atraviesan el láser en un corto periodo de tiempo este conjunto de granos serán percibidos como uno solo.

El uso de este sensor en específico es popular dentro del área de investigación dunar. Por lo tanto, la utilización de este sensor permitiría una mejor comparativa con el estado del arte en este ámbito.

7.5.1. Integración eléctrica del sensor Wenglor

El sensor de láser Wenglor tiene unas características eléctricas distintas al resto del sistema actual. En concreto si miramos la Figura 7.7 podemos ver que el voltaje mínimo al que funciona este sensor es 10 V. Esto contrasta con el voltaje máximo de la placa MKR, que son 5 V. Por lo tanto, esto crea la necesidad de proporcionar ambos voltajes para el funcionamiento de los dispositivos.

En los siguientes apartados se explicarán las tres formas con las que se abordó este problema durante el desarrollo. Este TFT se centra más en el desarrollo de este nuevo sistema y no en el trabajo de despliegue final.

7.5.1.1. Lectura de la señal del sensor láser

La señal que emite el sensor láser para señalizar el corte del láser se hace con el voltaje de alimentación de este sensor. Mientras que el arduino MKR está capacitado para leer señales con un voltaje de 3.3 V. Esto significa que el voltaje de la señal del sensor tiene que ser reducido para ser utilizado.

Para hacer esto se utiliza un divisor de tensión con dos resistencias. Los valores de estas dos resistencias están elegidos de tal forma que la señal de salida real se encuentra alrededor de 3,3 V cuando la salida es alta y 0 V cuando es baja.

7.5.1.2. Utilización de dos fuentes de alimentación

En este modelo se utilizan dos fuentes de alimentación separadas. Una de un voltaje alta para alimentar el sensor láser. Y otra de un voltaje de 5V para alimentar el resto del sistema.

Es importante tener en cuenta que se deben conectar las tierras de ambas fuentes. Esto es debido a que la señal del sensor es leído por el MKR en base a su tierra, no en base a la tierra de la fuente del sensor láser. Si no fuera así el MKR percibiría ruidos en la señal del sensor láser causando lecturas “fantasmas”.

Debido a la complejidad que añaden dos fuentes para experimentar en el campo se decidió probar otros métodos de alimentar el sensor láser.

7.5.1.3. Utilización de un boost converter

En este modelo se utiliza una sola fuente proporcionada por el MKR. La placa MKR tiene integrado un controlador de batería, capaz de controlar la alimentación y recarga de la batería.

A esta fuente se le aplica un boost converter, que aumenta el voltaje del MKR de 5 V a un voltaje mayor. En el caso de este proyecto se utilizó para aumentar el voltaje a 12 V.

Figura 7.7: Datos técnicos del sensor láser de Wenglor. Tabla proporcionada por Wenglor.

Technical Data

Optical Data	
Fork Width	80 mm
Smallest Recognizable Part	40 µm
Smallest Detectable Gap	50 µm
Switching Hysteresis	< 20 µm
Light Source	Laser (red)
Wavelength	655 nm
Service Life (T = +25 °C)	100000 h
Laser Class (EN 60825-1)	1
Max. Ambient Light	10000 Lux
Light Spot Diameter	0,6 mm
Repeat Accuracy	< 5 µm
Electrical Data	
Supply Voltage	10...30 V DC
Current Consumption (Ub = 24 V)	< 50 mA
Switching Frequency	10 kHz
Response Time	50 µs
Off-Delay	0...100 ms
Temperature Range	-25...60 °C
Switching Output Voltage Drop	< 1,5 V
PNP Switching Output/Switching Current	200 mA
Internal Load Switching Output	5100 Ohm
Short Circuit Protection	yes
Reverse Polarity Protection	yes
Overload Protection	yes
Teach Mode	NT, MT
Protection Class	III
FDA Accession Number	0820592-000
Mechanical Data	
Setting Method	Teach-In
Housing Material	Plastic; Steel, nickel-plated
Full Encapsulation	yes
Degree of Protection	IP67
Connection	M8 × 1; 3-pin
Safety-relevant Data	
MTTFd (EN ISO 13849-1)	1436,4 a
PNP NO	●
Connection Diagram No.	158
Control Panel No.	H1
Suitable Connection Equipment No.	8

Este modelo tiene problemas en la cantidad de corriente que es capaz de aplicar el MKR al sensor láser.

7.5.1.4. Utilización de un buck converter

Debido a las limitaciones del boost converter se decidió implementar el sistema inverso. Utilizar una fuente de 12V que alimenta el sensor laser, y bajar el voltaje de este para el resto del sistema.

Con esta solución no se aprovecha el controlador de carga de la placa MKR, pero simplifica el sistema de alimentación.

7.5.2. Consideraciones a tener en cuenta

Adicionalmente, en este trabajo se consideró también tener la posibilidad de tener varios sensores láser a distintas alturas. El objetivo de esto es poder estudiar las diferencias en el movimiento de la arena a distintas alturas.

Permitir una cantidad indeterminada de estos sensores introduce varias complejidades a la hora de recoger y enviar los datos.

7.5.3. Manejo de los datos

Para la recogida de datos disponemos de una lista de enteros, de 32 bits, con una entrada por sensor. Estas entradas actúan como contadores, contando la cantidad de pulsaciones que genera cada uno de los sensores asociados.

7.5.4. Implementación del ISR

La posibilidad de tener varios de estos sensores platea un problema espacial a la hora de implementarlo.

En especial a la hora de crear los ISR, Interrupt Service Routine. Estas rutinas tienen que estar definidas de forma global para poder ser asignadas en la tabla de vectores del procesador.

Para facilitar la incorporación de más sensores láser se utilizan las capacidades del precompilador de C. Se definió un macro y una función que permite crear nuevos ISRs y asociarlo con el pin relacionado. Con esto fácilmente se puede modificar el código para incluir más sensores de arena. Se puede ver el código en la Listado 7.5. Gracias a esto solo la función de inicialización tiene que saber a qué pin de la placa está conectado cada sensor láser.

```
#define CREATE_INCREMENT_FUNCTION(pin_id)     \
    static void increase_pin_value_##pin_id() \
    {                                         \
```

```
        sensor_values[pin_id]++;              \
    }

CREATE_INCREMENT_FUNCTION(0);

static inline void assign_interrupts_to_sensor_id(pin_size_t sand_interrupt_pin, uint8_t sand_sensor_id)
{
    switch (sand_sensor_id)
    {
    case 0:
        attachInterrupt(digitalPinToInterrupt(sand_interrupt_pin), increase_pin_value_0, RISING);
        break;
    }
}
```

Listado 7.5: Código de configuración de interrupciones ISR para sensores de arena wenglor

7.5.5. Transmisión por LoRa

Los distintos nodos pueden disponer de distinto número de sensores de transporte. Debido a esto el protocolo de comunicación mediante LoRa necesita ser capaz de soportar una cantidad dinámica de valores de sensores.

Para hacer esto se utiliza un byte para indicar el número de valores de conteo de arena que van adjuntados en el mensaje. Con esto por cada sensor de arena se manda en el mensaje un entero de 4 bytes indicando el conteo de pulsaciones que ha tenido el sensor de transporte de arena.

7.6. Estación base *meteo*

Como se ha mencionado en otros apartados la red de sensores dispone de una estación base. Con los dispositivos versión dos la estación base se ocupa de recibir y mostrar el estado de los dispositivos sensores. Esto se hace para permitir la monitorización del experimento durante su realización. Ayudando a la detección de los problemas que surjan durante un experimento y poder arreglarlo en el campo.

Esto es importante debido a las condiciones y cuestiones logísticas en el lugar de despliegue. Para poder realizar estos estudios en entornos dunares se requiere bastante planificación; solicitar permisos, asegurar financiación, etc. Por lo tanto, es fundamental poder detectar y arreglar los problemas que surjan durante el despliegue y el experimento.

La estación base consiste de un Raspberry Pi en la que se ejecuta una aplicación con una interfaz gráfica llamada ***meteo*** que es la que realiza la monitorización del sistema.

Esta aplicación se ocupa de procesar la entrada de los sensores y mostrar los datos y el estado actual de estos.

Posteriormente, para visualizar los datos se utiliza una tablet como pantalla mediante VNC. La Raspberry Pi utilizada tiene un módulo WiFi que se utiliza para proporcionar un ponto de acceso. Utilizando una tablet el usuario puede conectarse a este punto y utilizar el servicio VNC.

7.6.1. Características básicas de la estación base

La aplicación *meteo* es un programa desarrollado en C++ utilizando GTK [20] como librería gráfica.

Esta aplicación se ocupaba en la versión previa de este proyecto de recibir, interpretar y mostrar los datos recibidos de los sensores mediante Zigbee. Además, también se ocupaba del registro y sincronización de los mismos durante el despliegue.

Toda esta funcionalidad se mantiene en la nueva actualización del sistema, únicamente expandiendo la funcionalidad de esta aplicación para integrar en la misma los nuevos dispositivos versión dos. Esto se hace con el objetivo de permitir coexistir el nuevo sistema de sensores con el antiguo. Así el sistema de sensores anterior se podrá seguir utilizando en conjunto con los nuevos sensores desarrollados durante un despliegue en campo.

7.6.1.1. Periféricos de la estación base

La aplicación *meteo* requiere que la estación que lo esté utilizando disponga de unos periféricos concretos para su correcto funcionamiento.

Estos periféricos son lo que utiliza la estación base para recibir los paquetes provenientes de los dispositivos.

En primer lugar, tenemos un módulo Zigbee, requisito desde la anterior versión de este sistema y método de comunicación con los dispositivos versión uno. Este módulo Zigbee sirve exclusivamente como interfaz al protocolo Zigbee. Desde la aplicación *meteo* se tiene que realizar un procedimiento de interpretación y manejo de los distintos paquetes recibidos desde esta interfaz.

Por el otro lado con esta nueva versión se añade un nuevo periférico, un Arduino MKR 1310. El mismo dispositivo utilizado en los sistemas sensores. Este periférico se conecta mediante UART con la estación base. El software de comunicaciones LoRa que se ejecuta en el Arduino MKR conectado a la estación base ha sido también parte del trabajo desarrollado para la versión 2 del sistema.

Este periférico no actúa como un portal al protocolo LoRa. Si no que abstrae por completo los detalles del protocolo de comunicación. Este dispositivo capta los paquetes LoRa y los interpreta. Si el paquete recibido es correcto y supera todos los controles, entonces se envía a la estación base a través del canal serie en formato de texto plano y utilizando comas como separadores.

7.6.2. Estructura preexistente de *meteo*

Como componente esencial del código está la estructura **app_context**. Este struct se inicializa de forma global. Su propósito es mantener el estado actual de todo el programa. Esto incluye todos los dispositivos registrados y todas las entradas de datos recibidos.

En el Listado 7.6 se puede observar la definición de **app_context**. Este estructura de datos contiene 3 elementos fundamentales.

La primera son los regitros de datos, que corresponden con la entrada de datos desde los dispositivos. Esto se puede ver en variables como **messages**, **packages** y **devices**.

La segunda son las distintas variables de control que son utilizados para el funcionamiento interno del programa. Esto se ve por ejemplo en el mutex utilizado para controlar el acceso a las estructuras de datos, o en variables como **synchronized** o **selected_device**.

La última son los variables relacionados con la interfaz gráfica. Esto corresponde a distintas variables proporcionadas por GTK. Se utilizan estas variables para construir y controlar la interfaz.

```
struct app_context
{
  struct xbee* xbee;
  struct xbee_con* con;
  string address;

  bool synchronized;

  mutex mtx;
  list<string> messages;
  list<packet_entry> packets;
  map<string,device_entry> devices;

  GtkWidget* text_view;
  GtkWidget* data_view;
  GdkPixmap* pixmap;
  int devices_per_row;
  int devices_per_column;
  int pixmap_height;

  GtkWidget* samples_frame;
  GtkWidget* wind_velocity_view;
  GdkPixmap* wind_pixmap;
  GtkWidget* temperature_view;
  GdkPixmap* temperature_pixmap;
  GtkWidget* device_text_view;

  device_entry* selected_device;

  fstream file;

  chrono::system_clock::time_point starting_time;

  app_context()
  : xbee(nullptr),
    con(nullptr),
```

```
    address(),
    synchronized(false),
    text_view(nullptr),
    data_view(nullptr),
    pixmap(nullptr),
    samples_frame(nullptr),
    wind_velocity_view(nullptr),
    wind_pixmap(nullptr),
    temperature_view(nullptr),
    temperature_pixmap(nullptr),
    device_text_view(nullptr),
    selected_device(nullptr)
  {}
};
```

Listado 7.6: Código anterior a este TFT del manejador central del estado del programa. La clase **app_context**

7.6.3. Primer paso. Refactorización

La aplicación inicialmente fue creada para permitir un único tipo de dispositivo, los dispositivos versión uno, las cuales utilizan XBee para sus comunicaciones. Una parte del trabajo de expansión se dedicó a refactorizar la aplicación existente.

El código original estaba contenido en un solo fichero de texto de 2483 líneas. Esto dificultaba tanto la lectura como la modificabilidad del fichero.

Para sofocar esto se llevó a cabo una ligera reorganización. Se creó un fichero header que contuviese las definiciones de las distintas estructuras de datos utilizados dentro del código original. Este era un cambio fácil de realizar y que tendría un impacto significativo en la accesibilidad los distintos componentes del código. Teniendo así un lugar centralizado donde poder realizar modificaciones sobre estos componentes.

7.6.4. Integración de los nuevos dispositivos LoRa

Un objetivo al añadir estos nuevos dispositivos a la aplicación *meteo* fue intentar tener un impacto mínimo sobre la aplicación actual.

Para hacer esto se intentó seguir utilizando las mismas estructuras de datos que originalmente existían. En especial seguir utilizando las estructuras de datos **samples** y **devices**. Estos actúan como listas de estructuras, las cuales se pueden ver en el Listado 7.7 y el Listado 7.8. Estas dos estructuras de datos son las que contienen toda la información de los dispositivos y de las entradas de datos de estos.

La aplicación original sigue un patrón MVC donde estás estructuras de datos son parte del modelo. Por lo tanto, si se pudiese seguir utilizando estas estructuras de datos los cambios que se tendrían que hacer a la interfaz gráfica serían mínimos.

Debido al papel central que juegan estas dos estructuras de datos se comentarán primero los cambios realizados sobre estas. Después se mirarán los cambios realizados en el resto del código de la aplicación para acomodar esto.

7.6.4.1. Cambio de las estructuras de datos fundamentales

Como se explicó anteriormente se quiere intentar obtener el mayor provecho de las estructuras de datos existentes.

Estas estructuras de datos están implementadas en el código fuente como structs. Además, las variables internas de estos structs son accedidas de forma directa por el resto de funciones del programa.

Gran cantidad de las variables de ambos dispositivos son compartidas entre ambas versiones de los dispositivos. Los dos sistemas se dedican primordialmente a medir la velocidad y dirección del viento. También comparten datos secundarios como la temperatura y el RRSI de la señal recibida, entre otras.

Debido a esto se decidió realizar una modificación más enfocada sobre estas estructuras. Definiendo así un enum que etiqueta cada instancia de **sample** y de **device** como un dispositivo LoRa o Zigbee. Este indicador será utilizado después en las zonas del programa donde se diferencian los datos de los distintos sistemas. Las nuevas versiones de estas estructuras se encuentran en la Listado 7.7 y la Listado 7.8.

Más allá del diferenciador entre distintos tipos de paquetes y dispositivos también se tuvieron que añadir los nuevos tipos de datos que utilizan los dispositivos LoRa.

```
struct sample
{
    double v_wind;                              // Wind speed (m/s)
    double th_wind;                             // Wind angle (degrees)
    double temperature;                         // Temperature (celsius degrees)
    long long time;                             // milliseconds
    chrono::system_clock::time_point base_time; // Time sample was recieved.
};
```

Listado 7.7: Struct sample de la aplicación *meteo*

```
struct sensor_device
{
    string ni;
    string time;
    string angle;
    string angle_direction;

    string wind_1;
    string wind_2;
    string wind_3; // xbee: Number of turns of anemometer
    string wind_4;

    string temperature;

    int rssi;
```

```
    string text_buffer;

    int x, y, width, height;

    double v_wind_max;

    double temperature_max;

    std::vector<uint32_t> sand_sensor_values;

    chrono::system_clock::time_point last_change;

    device_type i_device_type;
};
```

Listado 7.8: Struct device de la aplicación *meteo*

7.6.4.2. Nueva forma de crear dispositivos

En la versión Zigbee los dispositivos al inicializarse tienen que registrarse con la aplicación. Con este registro se crea un nuevo objeto de dispositivo y se guarda como parte del contexto de la aplicación. La aplicación de *meteo*, una vez todos los dispositivos Zigbee estuviesen registrados, se utiliza para sincronizar estos dispositivos.

Con el modelo LoRa esto cambia. Los nuevos dispositivos ya no requieren de sincronización. Por lo tanto, no existe la necesidad de tener conocimiento de estos dispositivos antes de que empiezan a recoger datos. Esto hizo que para los dispositivos LoRa no hay un mensaje de registro. Simplemente, disponen de un único tipo de mensaje, el paquete de datos.

Así la creación de una nueva instancia de un dispositivo LoRa se hace de forma dinámica. Al recibir un nuevo mensaje LoRa primero se comprueba si existe ya un dispositivo con el identificador indicado, y en tal caso se registra el mensaje vinculado a este dispositivo. En caso contrario, se instancia un nuevo objeto de dispositivo y se registra el mensaje.

7.6.5. Recepción de los datos LoRa por parte de la aplicación *meteo*

Como se mencionó, el periférico MKR de la estación de base se ocupa de recibir los mensajes LoRa entrantes. Pero estos todavía tienen que ser comunicados a la estación base e interpretados por *meteo*.

Por lo tanto, se tiene que crear la infraestructura de recepción e interpretación para los dispositivos nuevos.

Con los dispositivos versión uno la biblioteca de XBee utilizada proporcionaba distintos callbacks. Por ejemplo cuando entraba un mensaje nuevo se añadía el texto de este mensaje a una nueva entrada de una lista de strings. Esta lista de strings, llamado `messages` forma

parte del contexto de la aplicación. De forma periódica se procesan estos mensajes. Sacando de ellas la información y guardándola dentro de un objeto sample (ver Listado 7.7).

Por lo tanto, para los dispositivos versión dos que utilizan LoRa se sigue un esquema similar.

Debido a las diferencias en los datos a transmitir por cada dispositivo se decidió que el punto en común de ambos sistemas fueran las estructuras `packet` y `sample_device`. Esto significa que para el procesamiento de los strings entrantes se crea una lista nueva, paralela a `messages`. Aparte de eso también se tiene que crear un método que se ocupa de leer el canal serial y escribir cada nueva entrada en la lista de mensajes.

Para hacer todo esto se decide generar una clase que se ocupa de actuar como interfaz del dispositivo llamado `Station`. Este maneja la conexión con el Arduino MKR proporcionando los datos originados por parte de él. Se puede ver esta interfaz en el Listado 7.9.

La clase `Station` al inicializarse crea un nuevo hilo. Este hilo se queda monitorizando el puerto serie asociado con el Arduino MKR. Después de cada salto de línea escribe la línea completa dentro de la variable privada `m_queue`. Cuando el usuario está listo para procesar la entrada de datos se dedica a llamar a la función `try_read_line`. Esta función se dedica a quitar el primer valor de la cola y devolverlo al usuario. Si no hay más datos en la cola devuelve falso.

Para abstraer más la interfaz desde el punto de vista del usuario también se creó una clase llamada `ComDevice`. Utilizando esta clase el usuario únicamente tiene que especificar la ruta del dispositivo sin tener que inicializar la conexión. `ComDevice` inicializa el puerto serie y pasa este puerto serie a la clase de estación que se ocupa exclusivamente de leer y posteriormente cerrar el canal.

```
class Station
{
public:
    Station(boost::asio::serial_port &&stream);
    bool try_read_line(std::string &value);
    bool close();
    void wait_close();

private:
    std::mutex m_mutex;
    std::queue<std::string> m_queue;
    std::thread device_thread;
    std::atomic<bool> is_closed{false};
};

class ComDevice
{
public:
    ComDevice(const std::string &com_port, boost::asio::io_service &my_io_service);

    bool try_read_line(std::string &value)
    {
        return this->lora_device.try_read_line(value);
    }
```

```
    bool close()
    {
        return this->lora_device.close();
    }

    void wait_close()
    {
        return this->lora_device.wait_close();
    }

private:
    Station lora_device;
};
```

Listado 7.9: Interfaz `lora_station` aplicación *meteo*

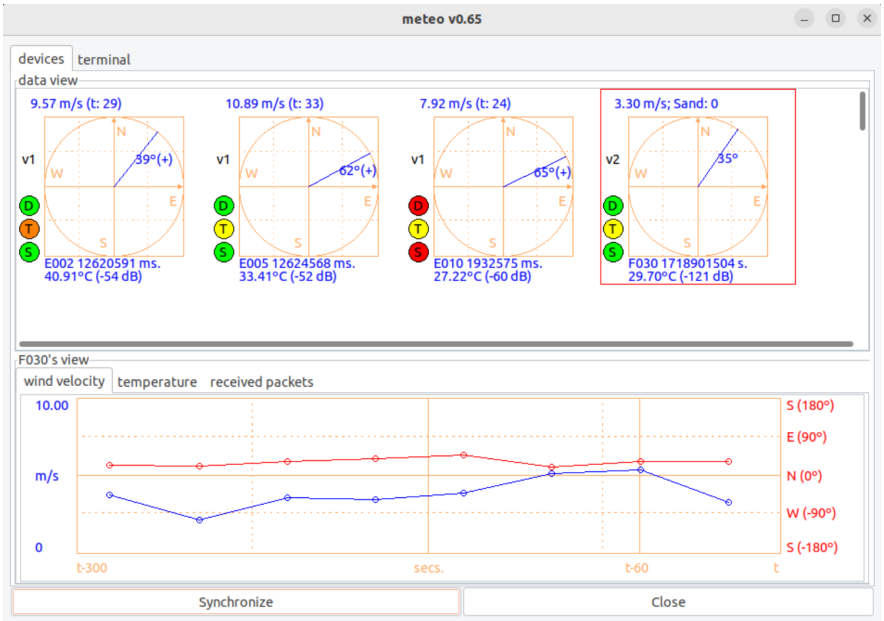
7.6.6. Cambio en la muestra de datos entre dispositivos LoRa y Zigbee

Como se ha visto los nuevos dispositivos LoRa no tienen los mismos datos que los dispositivos Zigbee. Hasta ahora hemos cambiado lo que corresponde al modelo de la aplicación *meteo*. Pero varios de estos datos exclusivos al Zigbee se muestran en la vista. Añadido a esto varios de los datos nuevos que recogen los dispositivos LoRa serían de interés monitorizarlos también en la interfaz.

Para integrar estos nuevos dispositivos se modifican unos puntos críticos del programa donde se debe diferenciar entre los dispositivos Zigbee y los dispositivos LoRa.

En primer lugar, se ha añadido un indicador de versión de dispositivo, permitiendo diferenciar entro los dispositivos v1 y v2. Junto a esto se han añadido los nuevos datos de los sensores de transporte de arena. Finalmente, se han realizado pequeños cambios en el formato de los datos. Por ejemplo en los dispositivos v2 el tiempo viene en segundos, mientras que en los dispositivos v1 vienen en milisegundos.

(a) Versión previa



(b) Versión actual

Figura 7.8: Comparación de la interfaz gráfica entre la versión *meteo* anterior a este TFT y la versión actual

Capítulo 8

Resultados y Pruebas

Para comprobar el funcionamiento de la nueva versión de los dispositivos, tanto de manera individual como su integración con el resto del sistema preexistente, se realizaron varios despliegues y pruebas. Estas se realizaron afuera del Edificio Polivalente I del Parque Científico Tecnológico de la ULPGC en el Campus Universitario de Tafira. Durante este capítulo se detallarán algunas de las pruebas realizadas y los resultados de estos.

En estos ensayos se desplegaron tres dispositivos de versión uno más un dispositivo versión dos(ver 8.1). Además, en muchos de estos ensayos se colocaba un dispositivo versión uno a lado del dispositivo versión dos para estudiar y comprobar el funcionamiento del nuevo dispositivo. Con esto se podía comprobar que los datos obtenidos por los nuevos dispositivos (versión 2) eran coherentes con los proporcionados por los dispositivos de la versión 1.

En la Figura 8.2 se muestran las imágenes de los ensayos realizados en exteriores.

Durante estos ensayos se recogieron datos en los ficheros de log de los dispositivos. En el Listado 8.1 se puede un extracto del fichero de log generado durante una de las sesiones por el dispositivo versión dos. Las primeras 4 líneas muestran las entradas de logs antes de que sean disponibles los datos GPS. Mientras que las 4 últimas líneas muestran la entrada de datos con los datos GPS. La visualización en una hoja de cálculo de estos datos se puede ver en la Figura 8.3

Mirando los datos producidos entre los dos tipos de dispositivos se puede observar como los valores producidos por los nuevos dispositivos contienen menos ruido que los dispositivos anteriores. Los datos del nuevo dispositivo tiene valores mucho más estables, subiendo y bajando de forma calmada. En comparación los datos producidos por los dispositivos de versión uno contienen muchos más cambios y discontinuidades. En el Listado 8.2, un extracto del fichero log generado por el dispositivo v1, se puede ver un ejemplo de esta inestabilidad con los datos de la dirección de viento, localizado en la tercera columna. Se pueden observar cambios significativos en el valor de este en un periodo de tiempo corto. En cambio, en el Listado 8.1 se puede ver como tanto la velocidad del viento, columna cinco, como el ángulo, columna seis, muestra una estabilidad que contrasta con lo visto en los dispositivos v1.

Gracias a la realización de estos ensayos se pudo detectar y corregir diversos fallos. Es-

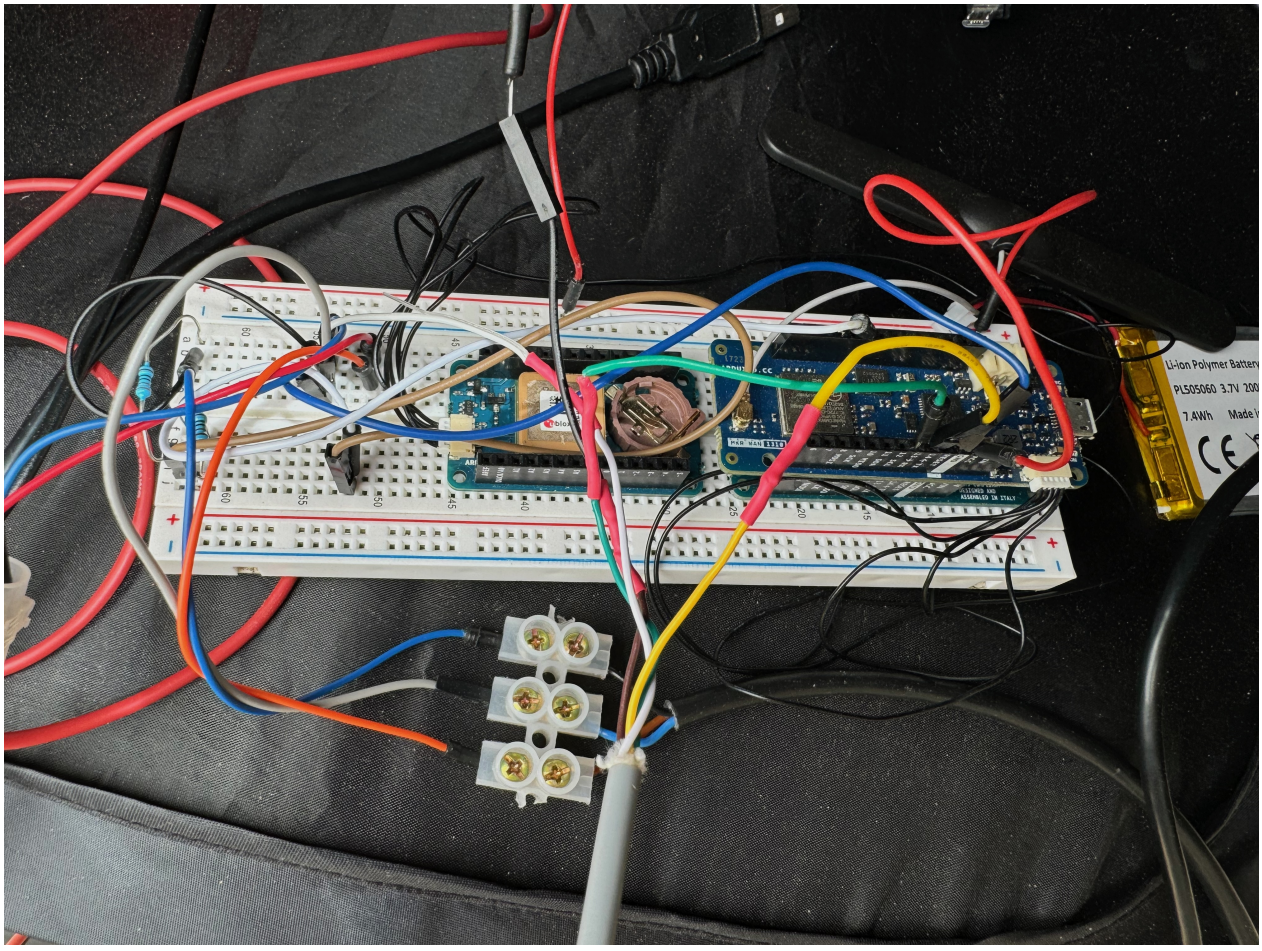


Figura 8.1: Circuitería de la unidad de desarrollo del dispositivo versión dos.

(a) Sensor dispositivo versión dos puesto a lado de un dispositivo versión uno.

(b) Puesto del resto de dispositivos versión uno para probar el despliegue completo del sistema.

(c) Despliegue de un dispositivo v1 y uno v2.

Figura 8.2: Fotografías de algunos ensayos realizado en exteriores.

```
25676,,,,2.900,16,MPS,0,28.60,1,[0,]
25919,,,,3.300,28,MPS,0,28.60,1,[0,]
26162,,,,3.700,28,MPS,0,28.60,1,[0,]
26404,,,,3.700,27,MPS,0,28.49,1,[0,]
26647,1720019191,-15.4525470734,28.0798377991,3.400,28,MPS,0,28.49,1,[0,]
26889,1720019191,-15.4525470734,28.0798377991,3.400,28,MPS,0,28.60,1,[0,]
27132,1720019191,-15.4525470734,28.0798377991,3.400,28,MPS,0,28.60,1,[0,]
27374,1720019192,-15.4525508881,28.0798301697,3.200,28,MPS,0,28.60,1,[0,]
27617,1720019192,-15.4525508881,28.0798301697,3.200,29,MPS,0,28.60,1,[0,]
```

Listado 8.1: Extracto de los logs del dispositivo versión dos

	Millis	GPS epoch	longitude	latitude	wind vel	wind angle	vel unit	is_relative	temp	n lasers	laser_i val	
71												
72	25676				2.9	16	MPS	0	28.6	1	[0	]
73	25919				3.3	28	MPS	0	28.6	1	[0	]
74	26162				3.7	28	MPS	0	28.6	1	[0	]
75	26404				3.7	27	MPS	0	28.49	1	[0	]
76	26647	1720019191	-15.452547	28.0798378	3.4	28	MPS	0	28.49	1	[0	]
77	26889	1720019191	-15.452547	28.0798378	3.4	28	MPS	0	28.6	1	[0	]
78	27132	1720019191	-15.452547	28.0798378	3.4	28	MPS	0	28.6	1	[0	]
79	27374	1720019192	-15.452551	28.0798302	3.2	28	MPS	0	28.6	1	[0	]

Figura 8.3: Datos del listado 8.1 mostrado en Excel.

tas pruebas obligaron en varias ocasiones a replantear y reconstruir distintos elementos del programa.

```
E002,1982312,141,-,130,99,2,-1,4563,11,0,-1,-1,0
E002,1984313,145,+,126,99,2,-1,4759,11,0,-1,-1,0
E002,1986313,143,-,122,99,2,-1,4547,11,0,-1,-1,0
E002,1988313,148,+,119,99,2,-1,4759,12,0,-1,-1,0
E002,1990313,147,-,116,107,3,-1,4759,11,0,-1,-1,0
E002,1992313,144,-,114,107,3,-1,4759,10,0,-1,-1,0
E002,1994313,94,-,112,107,3,-1,4531,10,0,-1,-1,0
E002,1996313,104,+,112,132,3,-1,4759,10,0,-1,-1,0
E002,1998312,98,-,111,157,4,-1,4775,11,0,-1,-1,0
E002,2000313,101,+,110,165,5,-1,4791,11,0,-1,-1,0
E002,2002313,98,-,111,173,5,-1,4531,11,0,-1,-1,0
E002,2004313,95,-,112,173,5,-1,4775,11,0,-1,-1,0
E002,2006313,98,+,113,173,4,-1,4775,11,0,-1,-1,0
E002,2008313,98,+,112,173,4,-1,4775,10,0,-1,-1,0
E002,2010313,91,-,112,173,3,-1,4580,10,0,-1,-1,0
E002,2012313,96,+,112,173,3,-1,4808,10,0,-1,-1,0
E002,2014312,86,-,111,173,3,-1,4824,10,0,-1,-1,0
E002,2016313,114,+,111,173,3,-1,4791,11,0,-1,-1,0
E002,2018313,97,-,109,173,3,-1,4580,10,0,-1,-1,0
E002,2020313,117,+,108,165,4,-1,4563,11,0,-1,-1,0
E002,2022313,71,-,108,140,3,-1,4808,11,0,-1,-1,0
E002,2024313,86,+,111,157,4,-1,4824,10,0,-1,-1,0
E002,2026313,107,+,117,231,6,-1,4775,10,0,-1,-1,0
E002,2028313,64,-,123,280,8,-1,4775,11,0,-1,-1,0
E002,2030312,93,+,128,280,8,-1,4775,10,0,-1,-1,0
E002,2032312,95,+,136,280,9,-1,4775,11,0,-1,-1,0
E002,2034313,79,-,143,280,8,-1,4775,11,0,-1,-1,0
E002,2036313,84,+,152,280,9,-1,4563,12,0,-1,-1,0
E002,2038313,89,+,158,280,8,-1,4775,11,0,-1,-1,0
E002,2040313,88,-,165,280,9,-1,4775,10,0,-1,-1,0
E002,2042313,95,+,171,289,8,-1,4531,10,0,-1,-1,0
E002,2044313,85,-,177,289,8,-1,4531,10,0,-1,-1,0
E002,2046312,89,+,183,289,8,-1,4775,10,0,-1,-1,0
E002,2048313,82,-,188,289,8,-1,4531,11,0,-1,-1,0
```

Listado 8.2: Extracto de los logs del dispositivo versión uno

Capítulo 9

Conclusiones y trabajo futuro

En este trabajo se ha conseguido cumplir con los objetivos inicialmente planteados. Consiguiendo crear un prototipo funcional de una nueva versión de los dispositivos sensores integrado el sistema de estudio dunar. Se ha conseguido que todo funcione correctamente y de cara a los usuarios finales no suponga un coste de aprendizaje significativo.

Concretamente, se han conseguido los siguientes puntos planteados:

- Se ha creado un sistema basado en el protocolo de comunicación LoRa, consiguiéndose, en consecuencia, crear un sistema con una capacidad de alcance mayor.
- Se ha integrar en el sistema un nuevo tipo de dispositivo basado en un sensor de viento ultrasónico proporcionado por Calypso que tiene mejores prestaciones que los integrados en los dispositivos versión 1.
- Se ha introducido un sensor de transporte de arena. Permitiendo la estimación de la cantidad de arena transportada con la visión de poder relacionar esto con las condiciones de viento.
- Se ha conseguido que este estos nuevos dispositivos trabajen en conjunto con el sistema preexistente de sensores. De esta forma estos nuevos dispositivos no son una sustitución del sistema anterior sino una expansión.

9.1. Dificultades

Una de las mayores dificultades durante el desarrollo de este TFT fueron los fallos de código y bugs que iban surgiendo y que debían ser corregidos. Estos llevaron un tiempo considerable mayor de lo estimado, además para poder realizar el despliegue completo era necesario moverse hacia el laboratorio de robótica del SIANI. El buen funcionamiento del trabajo es aún más importante considerando el uso futuro que va a tener. Considerando esto, se intentó focalizar el mayor esfuerzo posible en arreglar todos los bugs detectados realizando extensivas pruebas para detectar y reproducir distintos fallos.

También debido a la dependencia en recursos físicos que este trabajo requiere, el desarrollo y prueba de los distintos componentes tuvo una complejidad adicional.

A la hora de integrar e implementar el sensor de viento Calypso se presentaron diversas dificultades. Este sensor dispone de escasa documentación tanto por parte del fabricante como por la comunidad. La tarea de estudio y experimentación que se tuvo que realizar complicaron y retrasaron el tiempo de desarrollo para su integración.

9.2. Trabajo futuro

Este trabajo solo ha sido la primera fase de la creación de estos nuevos dispositivos. Ahora se deberá preparar y estudiar como convertir el prototipo actual en un modelo desplegable en campo para ser utilizado en sesiones y despliegues experimentales reales.

Esto implica la creación de una estructura física para contener la circuitería y para desplegar el sensor de viento a una altura adecuada. Similar a la creada para los dispositivos versión uno.

También se tendrá que probar el funcionamiento de los nuevos dispositivos dentro de un depliegue experimental real. Todas las pruebas se realizaron en el entorno del Campus Universitario de Tafira de la ULPGC, pero el entorno de estudio final son los campos de dunas, por lo tanto, sería fundamental comprobar el correcto funcionamiento de los dispositivos durante un despliegue en campo en condiciones reales.

Apéndice A

Normativa y Legislación relevante

A.1. Cuadro Nacional de Atribución de Frecuencias

Las radiofrecuencias un recurso limitado y compartido. Debido a esto los estados han regulado la utilización de esto para garantizar un acceso e utilización justo y equitativo.

Debido también al carácter trans-fronterizo de la comunicación por radio ha habido amplia cooperación entre distintos países para regular estos. Esto lo vemos en organizaciones como la Unión Internacional de Telecomunicaciones, una agencia de la ONU encargado de regular la telecomunicación a nivel internacional. Esta también tiene el privilegio de ser la agencia de la ONU más antigua, siendo fundado originalmente en 1865. [14]

En Europa se permite el uso sin licencia para LoRa las frecuencias 868MHz hasta 870MHz. Permitiendo una utilización con cualquier fin de estas frecuencias. [19, 7]

Pero estas bandas siguen siendo regulados. Según la normativa europea el tipo de dispositivo de radio que tenemos es un ′Non–specific Short Range Devices′. La utilización de estas bandas por estos dispositivos viene regulado por la normativa ETSI EN 300–220. Concretamente, la normativa ETSI EN 300 220 regula la potencia máxima de transmisión y duty cycle máximo que se pueden utilizar en distintos segmentos de banda. Se puede ver un extracto de la normativa ETSI EN 300–220 en la Listado A.1. [7]

Los transmisores LoRa utilizado están configuradas para utilizar la banda M de la normativa ETSI EN 300–220. Esto permite una transmisión con una potencia máxima de 25mW con un duty cycle menor del 1 %.

Estos estandares europeos vienen implementados como parte del Cuadro Nacional de Atribución de Frecuencias. Establecido en el orden ETD/1449/2021(BOE-A-2021-21346) y actualizado en el orden ETD/625/2023(BOE-A-2023-14422). El extracto exacto relevante para la utilización de ondas de este proyecto se encuentra en la página 57.

Figura A.1: Extracto de la normativa ETSI EN 300–220 V3.2.1 página 22. Describe las limitaciones de uso de bandas autorizado para LoRa.

Operational Frequency Band		Maximum effective radiated power, e.r.p.	Channel access and occupation rules (e.g. Duty cycle or LBT + AFA)	Maximum occupied bandwidth	Other usage restrictions	Band number from EC Decision 2017/1483/EU [2]
I	433,050 MHz to 434,790 MHz	1 mW e.r.p. -13 dBm/10 kHz power spectral density for bandwidth modulation larger than 250 kHz	No requirement	The whole band	Audio and video applications are excluded.	44a, 45a
J	434,040 MHz to 434,790 MHz	10 mW	No requirement	25 kHz	Audio and video applications are excluded.	45c
K	863 MHz to 865 MHz	25 mW e.r.p.	≤ 0,1 % duty cycle or polite spectrum access	The whole band except for audio & video applications limited to 300 kHz		46a
L	865 MHz to 868 MHz	25 mW e.r.p.	≤ 1 % duty cycle or polite spectrum access	The whole band		47
M	868,000 MHz to 868,600 MHz	25 mW e.r.p.	≤ 1 % duty cycle or polite spectrum access	The whole band		48
N	868,700 MHz to 869,200 MHz	25 mW e.r.p.	≤ 0,1 % duty cycle or polite spectrum access	The whole sub-band		50
P	869,400 MHz to 869,650 MHz	500 mW e.r.p.	≤ 10 % duty cycle or polite spectrum access	The whole band		54
P	869,700 MHz to 870,000 MHz	5 mW e.r.p.	No requirement	The whole band	Audio and video applications are excluded.	56a
Q	869,700 MHz to 870,000 MHz	25 mW e.r.p.	≤ 1 % duty cycle or polite spectrum access	The whole band	Analogue audio applications are excluded. Analogue video applications are excluded.	56b

UN-39 Banda 868-870 MHz. BOE-A-2021-21346

Aplicaciones de baja potencia con la consideración de uso común en el rango de frecuencias 868 a 870 MHz. Ver figura 24.
Esta banda se destina para aplicaciones de baja potencia y de datos en general de acuerdo con la Decisión de Ejecución (UE) 2019/1345 de la Comisión, por la que se modifica la Decisión 2006/771/CE, y se actualizan las condiciones técnicas armonizadas en el ámbito del uso del espectro radioeléctrico para los dispositivos de corto alcance, así como la Recomendación 70-03 (anexos 1 y 7) de la CEPT, conforme a la siguiente clasificación de dispositivos.

Dispositivos de baja potencia no específicos:

- 868,000 - 868,600 MHz con 25 mW (p.r.a.) de potencia radiada aparente máxima.
  - Estos dispositivos deberán utilizar técnicas de acceso y mitigación de interferencias con rendimiento al menos equivalente a las técnicas descritas en las normas armonizadas según la Directiva 2014/53/UE, o alternativamente no sobrepasar el 1 % de ciclo de trabajo.
- 868,700 - 869,200 MHz con 25 mW (p.r.a.) de potencia radiada aparente máxima.
  - Estos dispositivos deberán utilizar técnicas de acceso y mitigación de interferencias con rendimiento al menos equivalente a las técnicas descritas en las normas armonizadas según la Directiva 2014/53/UE, o alternativamente no sobrepasar el 0.1 % de ciclo de trabajo.
- 869,400 - 869,650 MHz con 500 mW (p.r.a.) de potencia radiada aparente máxima.
  - Estos dispositivos deberán utilizar técnicas de acceso y mitigación de interferencias con rendimiento al menos equivalente a las técnicas descritas en las normas armonizadas según la Directiva 2014/53/UE, o alternativamente no sobrepasar el 10 % de ciclo de trabajo.
- 869,700 - 870,000 MHz con 5 mW (p.r.a.) de potencia radiada aparente máxima.
  - Se permiten aplicaciones de voz con técnicas de mitigación avanzadas, excluyéndose otras aplicaciones de audio y de video.
- 869,700 - 870,000 MHz con 25 mW (p.r.a.) de potencia radiada aparente máxima.
  - Estos dispositivos deberán utilizar técnicas de acceso y mitigación de interferencias con rendimiento al menos equivalente a las técnicas descritas en las normas armonizadas según la Directiva 2014/53/UE, o alternativamente no sobrepasar el 1 % de ciclo de trabajo.

Apéndice B

Manual de Usuario

B.1. Application *meteo*

La aplicación *meteo* es ejecutado en un equipo local para monitorizar el estado de los sistemas sensores. Se puede ver una captura de esta aplicación en la Figura B.1.

Además, sirve para sincronizar los dispositivos anteriores(XBee). Debido a esto si se va a utilizar dispositivos sensores V1(XBee) es obligatorio la utilización de esta aplicación. Si no, esta aplicación es de uso opcional, sirviendo únicamente como observador de los dispositivos V2(LoRa).

B.1.1. Instalación de la aplicación *meteo*

El primer pasa para poder utilizar esta aplicación es instalarlo en el sistema en la que se va a utilizar. Esta sección explica el proceso de instalación desde el código fuente.

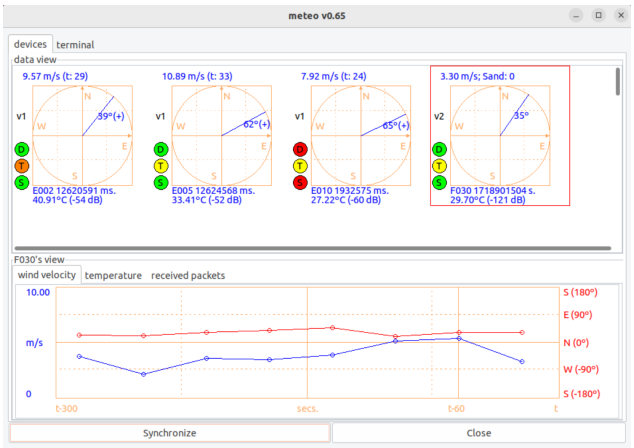


Figura B.1: Captura de la version actualizada de *meteo*. Elaboración propia

Esta aplicación está hecha únicamente para entornos Linux.

Actualmente, no se especifica requisitos mínimos. Para referencia el dispositivo de validación de esta aplicación es un Raspberry Pi con entorno gráfico. Junto a esto el dispositivo de desarrollo principal es una máquina Ubuntu.

B.1.1.1. Instalación de libxbee3

Para interactuar con el módulo XBee desde el programa su utiliza la biblioteca libxbee3. Este proporciona interfaces y callbacks con la que se puede interactuar fácilmente con el módulo XBee.

Se puede encontrar la biblioteca en https://github.com/attie/libxbee3 la cual se debe descargar o clonar utilizando git. Uno vez localizado en el equipo se debe configurar las opciones de construcción de esta biblioteca para habilitar ciertas funcionalidades que se utilizará en la aplicación *meteo*.

Para hacer esto primero ejecutamos `make configure`. Esto creará el fichero `config.mk`, que contiene distintas opciones de configuración de compilación. En este fichero se debe descomentar, para así activar, las siguientes dos opciones, `XBEE_NO_RTSCTS` y `XBEE_API2`.

Con estas dos opciones habilitadas se prosigue a primero construir la biblioteca utilizando el comando `make`. Una vez construido la biblioteca se instala con el comando `sudo make install`

B.1.1.2. Intalación del paquete GTK

Otro paquete requerido es la biblioteca GTK. Este es una biblioteca de creación de interfaces gráficas sobre la cual va construido la aplicación *meteo*.

Este paquete se suele poder instalar mediante el distribuidor de paquetes del el entorno Linux utilizado.

Utilizando el distribuidor de paquetes `apt` se puede instalar los paquetes GTK necesarios utilizando el comando `sudo apt install gtk2.0 libgtk2.0-dev`.

B.1.1.3. Construir *meteo*

Una vez instalado todos los paquetes se puede proceder a construir la aplicación *meteo*. Para hacer esto primero se entra en la carpeta donde está localizado el código de la aplicación *meteo*. Una vez dentro se ejecuta el comando `make`. Esto construirá el ejecutable binario, llamado por defecto *`meteo`*.

B.1.1.4. Habilitación de permisos de usuario

En la distribución Ubuntu y otras distribuciones los usuarios por defecto no disponen de permiso suficiente para aceder a dispositivos seriales. Esto se puede solventar ejecutando la aplicación *meteo* con permisos elevados o concediendo permisos al usuario para acceder a los puertos serial.

En Ubuntu el control de permiso a los puertos serial se hace con el grupo `dialout`. Para añadir un usuario a este grupo ejecuta el comando `sudo usermod -a -G dialup <usuario>`.

B.1.2. Utilización de la aplicación *meteo*

En la figura B.1 se puede ver los distintos elementos que componen la interfaz gráfica. En la parte superior de la interfaz se puede ver todos los dispositivos que están activos en este momento e información importante de cada uno de ellos. Cada dispositivo tiene asociado tres leds. El primero ´D' es de datos, indica si el dispositivo lleva un tiempo significativo sin cambiar los valores de dirección y velocidad del viento. Debajo de ese esta 'T' de temperatura, este indica si los dispositivos tienen una temperatura alta. Finalmente, está la 'S' que indica el estado del dispositivo. Si un dispositivo lleva un tiempo significativo sin comunicarse con la estación base se activa este led.

Como se puede ver en la figura B.1 entre más severo el fallo de datos, temperatura o estado, más rojizo se pone la luz led. En caso de ponerse algún led rojo deberías comprobar el estado del dispositivo indicado.

Al seleccionar algún dispositivo el segundo elemento de la interfaz, localizado abajo, empieza a ser útil. En este apartado se puede ver el historial de cada dispositivo repartido entre varias pestañas. Estas pestañas son: velocidad del viento, temperatura y paquetes recibidos. Las primeras dos gráficos de sus respectivos temas. Mientras que la pestaña de paquetes recibidos muestra un terminal de texto con el texto crudo de todos los paquetes recibidos.

B.2. Despliegue de los nuevos dispositivos del sistema (v2)

En esta sección se dará una explicación de distintos puntos a tener en cuenta al montar un dispositivo versión dos.

B.2.1. Conexión de las placas Arduino

Se debe disponer de las tres placas utilizadas en el proyecto: la placa de prototipado Arduino MKR 1310, el shield GPS del Arduino MKR y el shield proto SD del Arduino MKR.

En primer lugar, se monta la placa del Arduino MKR 1310 y el shield proto juntos. Después se conecta el Arduino MKR 1310 con el shield GPS mediante un cable I2C, que viene incluido con el shield GPS, conectando los puertos I2C disponible en ambas placas.

B.2.2. Introducción del anemómetro

El sensor de viento Calypso utilizado en este proyecto se conecta con la placa MKR mediante una conexión UART. Este contiene una línea RX, de recepción, y otro TX, de envío. Además tiene una línea de carga y otra de tierra.

Estos cuatro cables se conectan a la placa MKR 1310. La de carga puede conectarse tanto al puerto VCC como al de 5 V, el sensor es capaz de funcionar a ambos voltajes. Mientras que la línea RX se conecta al puerto TX y la línea TX se conecta al puerto RX. Finalmente, la línea de tierra se conecta a la tierra de la placa MKR 1310.

Estos cuatro cables del sensor vienen en cuatro colores:

- **Blanco**: GND (Power -)
- **Marrón**: VCC (Power +)
- **Amarillo**: Data (B -)/RX
- **Verde**: Data (A +)/TX

B.2.3. Introducción del sensor láser

El sensor láser Wenglor se puede alimentar con un voltaje de 10 V a 30 V mientras que el voltaje I/O del Arduino MKR es de 3,3 V. Debido a esto se debe bajar el voltaje de la señal del sensor láser, en la versión de desarrollo se utilizó dos resistencias que están colocados en serie conduciendo desde la salida del sensor hasta tierra.

Cuando el usuario despliegue el sistema debe elegir dos resistencias apropiadas para bajar la tensión de la señal. Para calcular los valores de las dos resistencias se puede utilizar la ley de Ohm. Suponiendo que hay dos resistencias R_1 y R_2, V_i describe el voltaje de la señal del sensor y V_o describe el voltaje entre las dos resistencias entonces:

$$V_o = V_i \frac{R_2}{R_1 + R_2}$$

Si se define las resistencias como $R_2 = nR_1$ se puede despejar la ecuación anterior tal que:

$$n = \frac{V_i - V_o}{V_o} = \frac{V_i - 3,3V}{3,3V}$$

Apéndice C

Código Configuración Calypso de forma dinámica

```
enum class FilterMedian
{
    LOW_POWER, // = 3,
    LOW,       // = 5,
    MEDIUM,    // = 10,
    HIGH,      // = 20
};

enum class FilterRepetition
{
    LOW_POWER, // = 1,
    LOW,       // = 2,
    MEDIUM,    // = 5,
    HIGH,      // = 10
};

enum class FilterDamping
{
    DEFAULT, // = 0,
};

enum class DataRate
{
    HZ_0_1, // = 0
    HZ_1_0, // = 4
};

enum class NMEAUnits
{
    KNOTS,
    MPS,
    KM_H
};

enum class NMEA_MODE
{
    NMEA_STREAM,
    NMEA_POLL
```

```
};

typedef struct
{
    FilterMedian filter_median;
    FilterRepetition filter_repetition;
    FilterDamping filter_damping;
    DataRate data_rate;
    NMEAUnits nmea_units;
    NMEA_MODE nmea_mode;
} ulp_sensor_config_t;

constexpr unsigned int ulp_serial_delay = 0;

std::unordered_map<FilterRepetition, char *> filterRepetitionMap = {
    {FilterRepetition::LOW_POWER, "FILTER_REPETICION_01"},
    {FilterRepetition::LOW, "FILTER_REPETICION_02"},
    {FilterRepetition::MEDIUM, "FILTER_REPETICION_05"},
    {FilterRepetition::HIGH, "FILTER_REPETICION_10"},
};

std::unordered_map<FilterDamping, char *> filterDampingMap = {
    {FilterDamping::DEFAULT, "FILTER_DAMPING_00"},
};

std::unordered_map<DataRate, char *> dataRateMap = {
    {DataRate::HZ_0_1, "DATARATE_01"},
    {DataRate::HZ_1_0, "DATARATE_04"},
};

std::unordered_map<NMEAUnits, char *> nmeaUnitsMap = {
    {NMEAUnits::KNOTS, "NMEA_UNITS_K"},
    {NMEAUnits::MPS, "NMEA_UNITS_M"},
    {NMEAUnits::KM_H, "NMEA_UNITS_N"},
};

std::unordered_map<NMEA_MODE, char *> nmeaModeMap = {
    {NMEA_MODE::NMEA_STREAM, "MODE␣NMEA␣STREAM"},
    {NMEA_MODE::NMEA_POLL, "MODE␣NMEA␣POLL"},
};

static void configure_ulp_sensor(Stream &stream, const ulp_sensor_config_t &config);

void initialize_ulp_sensor(Stream &stream, const ulp_sensor_config_t &config)
{
    configure_ulp_sensor(stream, config);
}

static inline void initialize_config(Stream &stream);

static void
configure_ulp_sensor(Stream &stream, const ulp_sensor_config_t &config)
{
    initialize_config(stream);

    nmea0183::send_command(stream, dataRateMap.at(config.data_rate));
    delay(ulp_serial_delay);

    nmea0183::send_command(stream, filterMedianMap.at(config.filter_median));
```

```
    delay(ulp_serial_delay);

    nmea0183::send_command(stream, filterRepetitionMap.at(config.filter_repetition));
    delay(ulp_serial_delay);

    nmea0183::send_command(stream, filterDampingMap.at(config.filter_damping));
    delay(ulp_serial_delay);

    nmea0183::send_command(stream, nmeaUnitsMap.at(config.nmea_units));
    delay(ulp_serial_delay);

    stream.println("$STOP_CONFIG");
    delay(ulp_serial_delay);

    stream.println(nmeaModeMap.at(config.nmea_mode));
    delay(ulp_serial_delay);

#ifdef DEBUG
    Serial.println("Configured␣ULP␣sensor:");
    initialize_config(Serial);
    nmea0183::send_command(Serial, dataRateMap.at(config.data_rate));
    nmea0183::send_command(Serial, filterMedianMap.at(config.filter_median));
    nmea0183::send_command(Serial, filterRepetitionMap.at(config.filter_repetition));
    nmea0183::send_command(Serial, filterDampingMap.at(config.filter_damping));
    nmea0183::send_command(Serial, nmeaUnitsMap.at(config.nmea_units));
    Serial.println("$STOP_CONFIG");
    Serial.println(nmeaModeMap.at(config.nmea_mode));
    Serial.println("\nConfigured␣ULP␣sensor\n");
    Serial.flush();
#endif
}

static inline void initialize_config(Stream &stream)
{
    stream.println("$RESET94399*6B");
    delay(ulp_serial_delay);
    stream.println("$START_ULTRA_CALYPSO");
    delay(ulp_serial_delay);
    stream.println("$START_CONFIG");
    delay(ulp_serial_delay);
}
```

Listado C.1: Código de configuración del sensor de viento Calypso para Arduino

Bibliografía

[1] Arduino Wireless SD Shield. 15 de oct. de 2019. URL: https://web.archive.org/web/20191015221450/https://store.arduino.cc/arduino-wireless-sd-shield (visitado 04-06-2024).

[2] Arduino_MKRGPS - Arduino Reference. URL: https://www.arduino.cc/reference/en/libraries/arduino_mkrgps/ (visitado 01-07-2024).

[3] Comparing Ultrasonic versus Mechanical Wind Speed Measurement Sensors Comptus. Comptus. 30 de dic. de 2020. URL: https://www.comptus.com/comparing-ultrasonic-versus-mechanical-wind-speed-measurement-sensors/ (visitado 24-06-2024).

[4] Daniel Cressey. «The DIY Electronics Transforming Research». En: Nature 544.7648 (abr. de 2017), págs. 125-126. ISSN: 1476-4687. DOI: 10.1038/544125a. URL: https://www.nature.com/articles/544125a (visitado 08-07-2024).

[5] Antonio C. Domínguez-Brito et al. «A DIY Low-Cost Wireless Wind Data Acquisition System Used to Study an Arid Coastal Foredune». En: Sensors 20.4 (15 de feb. de 2020), pág. 1064. ISSN: 1424-8220. DOI: 10.3390/s20041064. URL: https://www.mdpi.com/1424-8220/20/4/1064 (visitado 01-06-2024).

[6] Duty Cycle. Wikipedia. 9 de abr. de 2024. URL: https://en.wikipedia.org/w/index.php?title=Duty_cycle&oldid=1218091142 (visitado 31-05-2024).

[7] ETSI EN 300 220. URL: https://docdb.cept.org/document/28427 (visitado 01-06-2024).

[8] Explore the Digi XBee Ecosystem. URL: https://www.digi.com/xbee (visitado 08-07-2024).

[9] Fork Sensor (YH08PCT8). wenglor sensoric group. URL: https://www.wenglor.com/en/Sensors/Photoelectronic-Sensors/Fork-Sensors/Fork-Sensor/p/YH08PCT8 (visitado 11-03-2024).

[10] Raspberry Pi Foundation. Teach, Learn, and Make with the Raspberry Pi Foundation. Raspberry Pi Foundation. 5 de jul. de 2024. URL: https://www.raspberrypi.org/ (visitado 08-07-2024).

[11] Attie Grande. Attie/Libxbee3. 6 de abr. de 2024. URL: https://github.com/attie/libxbee3 (visitado 28-06-2024).

[12] Cindy Harnett. «Open Source Hardware for Instrumentation and Measurement». En: IEEE Instrumentation & Measurement Magazine 14.3 (jun. de 2011), págs. 34-38. ISSN: 1941-0123. DOI: 10.1109/MIM.2011.5773535. URL: https://ieeexplore.ieee.org/document/5773535 (visitado 08-07-2024).

[13] How to Configure Your ULP. Calypso Instruments EMEA, S.L. 24 de ago. de 2022. URL: https://calypsoinstruments.com/blog/blog-1/post/how-to-configure-your-ulp-143 (visitado 30-06-2024).

[14] International Telecommunication Union. Wikipedia. 16 de abr. de 2024. URL: https://en.wikipedia.org/w/index.php?title=International_Telecommunication_Union&oldid=1219223582 (visitado 31-05-2024).

[15] MKR GPS Shield — Arduino Documentation. URL: https://docs.arduino.cc/hardware/mkr-gps-shield/ (visitado 01-07-2024).

[16] MKR WAN 1310 — Arduino Documentation. URL: https://docs.arduino.cc/hardware/mkr-wan-1310/ (visitado 01-07-2024).

[17] NMEA 0183. Wikipedia. 4 de feb. de 2024. URL: https://en.wikipedia.org/w/index.php?title=NMEA_0183&oldid=1203390440 (visitado 11-03-2024).

[18] NMEA Revealed. GPSD. URL: https://gpsd.gitlab.io/gpsd/NMEA.html (visitado 02-06-2024).

[19] Regional Limitations of RF Use in LoRaWAN. The Things Network. URL: https://www.thethingsnetwork.org/docs/lorawan/regional-limitations-of-rf-use/ (visitado 31-05-2024).

[20] The GTK Team. The GTK Project - A Free and Open-Source Cross-Platform Widget Toolkit. The GTK Team. URL: https://www.gtk.org/ (visitado 08-07-2024).

[21] Technical Information — Calypso Instruments. Prodeo Ingeniería y Consultoría, S.L. URL: https://calypsoinstruments.com/technical-information (visitado 11-03-2024).

[22] The Arduino Guide to LoRa® and LoRaWAN® — Arduino Documentation. URL: https://docs.arduino.cc/learn/communication/lorawan-101/ (visitado 11-03-2024).

[23] The NMEA 0183 Protocol. Prodeo Ingeniería y Consultoría, S.L. 13 de sep. de 2022. URL: https://calypsoinstruments.com/blog/blog-1/post/the-nmea-0183-protocol-145 (visitado 11-03-2024).

[24] The NMEA 0183 Protocol. URL: http://archive.org/details/manualzz-id-733016 (visitado 02-06-2024).

[25] Ultra Low Power-Ultrasonic Wind Meter. Calypso Instruments EMEA, S.L. URL: https://calypsoinstruments.com/shop/product/ultra-low-power-ultrasonic-wind-meter-ulp-std-6?category=2 (visitado 23-05-2024).

[26] Virtual Network Computing. Wikipedia. 30 de mar. de 2024. URL: https://en.wikipedia.org/w/index.php?title=Virtual_Network_Computing&oldid=1216325354 (visitado 01-07-2024).

[27] XBee Buying Guide - SparkFun Electronics. URL: https://www.sparkfun.com/pages/xbee_guide (visitado 08-07-2024).

[28] XBee Pro 60mW Wire Antenna - Series 1 (802.15.4) - WRL-08742 - SparkFun Electronics. URL: https://www.sparkfun.com/products/retired/8742 (visitado 08-07-2024).

[29] XBee/XBee-PRO S1 802.15.4 (Legacy) User Guide. URL: https://www.digi.com/resources/documentation/digidocs/pdfs/90000982.pdf.

[30] Zhou Yufeng y Wang Yan. «To Measure Wind Speed Using the Theory of One-dimensional Ultrasonic Anemometer» (jun. de 2011).