

CARRERA DE ESPECIALIZACIÓN EN SISTEMAS EMBEBIDOS

MEMORIA DEL TRABAJO FINAL

Prototipo de dispositivo de geolocalización basado en tecnología GNSS

Autor:
Ing. Luis David Díaz Charris

Director:
Esp. Ing. Santiago Esteva (FIUBA)

Jurados:
Dr. Ing. Javier Jiménez (Universidad de la Costa, CUC)
Dr. Ing. Emiro De La Hoz (Universidad de la Costa, CUC)
Mg. Ing. Jorge Díaz (Universidad de la Costa, CUC)

*Este trabajo fue realizado en la ciudad de Barranquilla, Colombia,
entre marzo de 2023 y agosto de 2024.*

Resumen

En esta memoria se describe el diseño y desarrollo de un prototipo de dispositivo de geolocalización basado en tecnología GNSS. Este permite reportar la posición geográfica de una cava móvil de transporte de alimentos con una precisión mínima de diez metros. El prototipo implementa un receptor de señales GNSS capaz de recibir simultáneamente señales de los sistemas GPS y BeiDou. Además de incorporar comunicación 4G, lo que permite la posibilidad de transmitir datos vía internet a una aplicación web de monitoreo en tiempo real.

Para el desarrollo del presente trabajo, se aplicaron todos los conocimientos en planificación y gestión de proyectos, ingeniería del software para sistemas embebidos, programación de microcontroladores, sistemas operativos en tiempo real, diseño de circuitos impresos y protocolos de comunicación.

Agradecimientos

Agradezco a Dios por su infinita misericordia.

Agradezco a mi esposa Yuranys, por su aguante para conmigo.

Agradezco enormemente a Ariel Lutenberg por la oportunidad, la confianza y sus inacabables ganas de ayudar.

Agradezco a Martín Menéndez, a Santiago Germino y a Ramiro Ghignone por darme un lugar en su grupo.

Agradezco a Santiago Esteva por su guía en este trabajo.

Índice general

Resumen	I
1. Introducción general	1
1.1. Descripción del proyecto	1
1.1.1. Contexto y problema	1
1.1.2. Alternativa de solución	2
1.2. Estado del arte	2
1.2.1. Antecedentes	3
1.3. Objetivos y alcance	4
1.3.1. Objetivo general	4
1.3.2. Objetivos específicos	4
1.3.3. Alcance	4
2. Introducción específica	5
2.1. Requerimientos	5
2.2. Estructura del sistema	6
2.3. Componentes del sistema	6
2.3.1. Microcontrolador ESP32	7
2.3.2. Receptor GNSS	9
2.3.3. Módulo 4G Antena Quectel YG0035AA	9
2.3.4. Sistema operativo de tiempo real	11
FreeRTOS™	12
2.3.5. Servicio web para mapas	12
3. Diseño e implementación	15
3.1. Diseño e implementación general del sistema	15
3.2. Diseño e implementación de hardware	15
3.2.1. Diseño del Hardware	15
3.2.2. Implementación del Hardware	16
Placa base del sistema	16
Adaptador para el módulo Quectel L76	17
3.3. Diseño e implementación del firmware	19
3.3.1. Diseño del firmware	19
Arquitectura del firmware	19
3.3.2. Implementación del firmware	21
Flujo general del firmware	21
Implementación de las tareas del sistema	22
Otros componentes del firmware	25
3.4. Diseño e implementación de la interfaz web	28
3.4.1. Diseño de la interfaz web	28
3.4.2. Implementación de la interfaz web	29

3.4.3. Metodología de la implementación	29
4. Ensayos y resultados	35
4.1. Pruebas funcionales del hardware	35
5. Conclusiones	37
5.1. Conclusiones generales	37
5.2. Próximos pasos	37
A. Esquemáctico de la placa base del sistema	39

Índice de figuras

1.1. Ejemplo de una cava móvil ¹	1
2.1. Diagrama general del sistema.	6
2.2. Placa de desarrollo ESP32-Devkit-V1 ²	8
2.3. Módulo Quectel L76 ³	9
2.4. Módulo SIM A7670SA ⁴	10
2.5. Módulo A7670SA ⁵	11
2.6. Antena Quectel YG0035AA ⁶	11
3.1. Arquitectura del hardware desarrollado.	16
3.2. Modelo 2D de la placa base del sistema.	17
3.3. Modelo 3D de la placa base del sistema.	17
3.4. Esquemático del adaptador L76.	18
3.5. Modelo 2D del adaptador L76.	18
3.6. Modelo 3D del adaptador L76.	18
3.7. Arquitectura del firmware.	20
3.8. Diagrama de flujo del firmware (parte 1).	22
3.9. Diagrama de flujo del firmware (parte 2).	23
3.10. Diagrama de flujo del firmware (parte 3).	23
3.11. Arquitectura MQTT ⁷	28
3.12. Maqueta de la interfaz web.	34
3.13. Maqueta de la interfaz web.	34

Índice de tablas

2.1. Tabla comparativa de Microcontroladores.	7
2.2. Tabla comparativa de APIs de mapas.	13

Dedicado a...

Mis hijos, Tomás y María Helena, son lo mejor de mi vida.
Mi esposa Yuranys.
Mi madre Luz Fabiola.

Capítulo 1

Introducción general

En este capítulo se exponen los aspectos fundamentales del trabajo, con el propósito de familiarizar al lector con las estrategias de desarrollo que se detallan en los capítulos subsiguientes.

1.1. Descripción del proyecto

A continuación, se presenta una descripción del contexto, problema y propuesta de solución que guiaron el presente trabajo.

1.1.1. Contexto y problema

Una cadena de supermercados en Colombia tiene importantes retos logísticos a la hora de abastecer las tiendas distribuidas en todas las ciudades. La empresa implementa centros de abastecimiento que cubren distintas ciudades en determinados rangos de acción. Uno de los retos más relevantes a los que se enfrenta, es la distribución de alimentos en cadena de frío. Dicha tarea, se lleva a cabo a través de cavas móviles que tienen la capacidad de conservar la temperatura interna y refrigerar su interior. Estas cavas representan un activo costoso y son totalmente reutilizables. En la figura 1.1, se puede apreciar un ejemplo de cava móvil para transporte de alimentos.



FIGURA 1.1. Ejemplo de una cava móvil¹.

¹<https://www.coldtainer.es>

El centro de despacho enfrenta problemas logísticos cuando se queda sin cavas disponibles para la distribución. En esta situación, es necesario realizar un esfuerzo adicional para ubicar y recuperar las cavas. La dificultad radica en que no es posible saber su ubicación exacta en tiempo real. Por lo tanto, se requiere solicitar a cada sucursal el reporte de cavas disponibles para su retorno al centro de distribución.

1.1.2. Alternativa de solución

Por lo anterior, se planteó como una alternativa de solución, el desarrollo de un dispositivo que permita rastrear la ubicación de cada una de las cavas. Este dispositivo permite replicarse con facilidad, es de bajo consumo de energía y tiene una precisión de diez metros en el reporte de ubicación de las cavas.

1.2. Estado del arte

Los sistemas de navegación por satélite son actualmente las herramientas por excelencia en las aplicaciones de movilidad y transporte. El geo-posicionamiento es una utilidad necesaria en todos los tipos de transporte: navegación espacial, la aviación, la navegación marítima, los ferrocarriles y el transporte automotor por carretera [GNSS-Junta-Adalucia].

Uno de los sistemas de navegación más conocidos es el Sistema de Posicionamiento Global (GPS), desarrollado por el Departamento de Defensa de Estados Unidos. Sin embargo, a nivel mundial existen varios de estos sistemas, y al conjunto de ellos se le conoce como Sistemas de Navegación Global por Satélite (GNSS, del inglés Global Navigation Satellite Systems) [GNSS-ONU]. Los GNSS están compuestos por constelaciones de satélites que orbitan la Tierra y transmiten datos sobre su ubicación espacial y temporal. Además, incluyen redes de estaciones de control terrestres y receptores que calculan las posiciones en tierra mediante la técnica de trilateración [GNSS-ONU].

Algunos de los GNSS que existen actualmente son [GNSS-Junta-Adalucia]:

- El Sistema GPS (Global Position System): de EE. UU., con 30 satélites, a una altura de 20 200 km.
- El Sistema Glonass (Global Orbiting Navigation Satellite System): de Rusia, con 24 satélites, a una altura de 25 500 km.
- El Sistema Galileo (ESA): de Europa, con 30 satélites, a una altura de 23 200 km.
- El Sistema BeiDou (BDS): de China, con 34 satélites, a una altura entre 21 600 km. y 35 800 km.

En la literatura científica, estos sistemas son ampliamente estudiados y revisados, debido a su gran rango de aplicaciones en diferentes áreas del conocimiento (aparte de la navegación) y la técnica. Entre las más relevantes se destacan:

- Ingeniería Civil y Arquitectura. Por ejemplo, para el replanteo de infraestructuras y estructuras, control del movimiento de tierras, medición de perfiles transversales, entre otras.

- Cartografía y Topografía, para mejorar la precisión de la cartografía y la modelización del mundo físico, desde montañas y ríos, hasta calles, edificios, cables y tuberías de los servicios públicos.
- Geodesia y Geofísica, para mejorar la precisión en el estudio de los fenómenos físicos que afectan la forma y dimensiones de la Tierra.
- Agricultura en el desarrollo de mejores técnicas para la gestión de los cultivos.

1.2.1. Antecedentes

A continuación, se presentan algunos antecedentes en la literatura científica sobre el estudio de los sistemas GNSS.

Di Garzia et al. [DiGrazia] evaluaron el rendimiento del chip receptor de GNSS Te-seoV STA8135GA, de STMicroelectronics. Su trabajo se contextualiza en las aplicaciones automotrices de alta precisión. Determinaron la viabilidad técnica y las ventajas de que en un solo chip se pudiera incorporar la capacidad de recibir señales de tres bandas. Lo anterior, permitió agregar resiliencia y redundancia en caso de fallas completas de alguna de las bandas de frecuencia.

Shu et al. [Shu] evaluaron el rendimiento de un GNSS multi-constelación para averiguar el impacto en la determinación de la posición en un ambiente dinámico. Lo anterior, para saber cuánto mejora el GNSS la precisión de la posición. Con su estudio determinaron que:

- En un caso estático, los sistemas GPS, GLONASS y BeiDou tienen el mismo rendimiento.
- En el caso dinámico, los sistemas GPS y GLONASS tienen rendimientos muy similares, donde el de GPS es superior.
- Cuando se combinan GPS, BeiDou, GLONASS y Galileo, la respuesta es significativamente mejor en comparación con el empleo de un único sistema de posicionamiento.

Por su parte, *Villien y Denis [Villien]* emplean una combinación de tres estrategias para mejorar la calidad de la recepción GNSS:

1. Utilizan un receptor GNSS de múltiples constelaciones y para ampliar el rango de transmisores satelitales de los que obtener información de posicionamiento.
2. Utilizan una antena de banda ultra ancha (UWB, del inglés ultra-wideband) para optimizar en una única antena la capacidad de recibir señales GNSS con un ancho de banda amplio.
3. Emplean una unidad de medida inercial junto con un filtro Kalman. Esta configuración permite rastrear de forma completa el objeto observado y reducir la carga computacional del sistema.

El trabajo de *Villien y Denis* logra un posicionamiento con una precisión de alcance entre 3 cm y 11 cm en entornos controlados. Además, se evidencia una precisión horizontal en las mediciones GNSS superior a 40 cm durante una prueba

de campo, incluso en condiciones degradadas de recepción de la señal GNSS. Este trabajo es especialmente relevante en aplicaciones de geo-posicionamiento en espacios cerrados o túneles, donde un único sistema GNSS resulta insuficiente.

1.3. Objetivos y alcance

1.3.1. Objetivo general

Desarrollar un prototipo de dispositivo de geolocalización basado en tecnología GNSS, que permita reportar y hacer seguimiento en tiempo real de la posición geográfica de una cava móvil de transporte de alimentos con una precisión mínima de diez metros.

1.3.2. Objetivos específicos

1. Diseñar la arquitectura de hardware y firmware, del dispositivo de geolocalización con tecnología GNSS.
2. Implementar el hardware y el firmware del dispositivo de geolocalización.
3. Implementar un servicio online para el seguimiento de la posición geográfica.
4. Realizar pruebas de consumo energético, precisión y funcionalidad del prototipo.

1.3.3. Alcance

Este trabajo abordó:

- El análisis de métodos ingenieriles adecuados y tecnologías disponibles para la implementación del sistema.
- El diseño y prototipado del sistema y subsistemas de hardware requerido para la interconexión con los sistemas GNSS y la red celular.
- El diseño y prototipado del sistema y subsistemas de hardware requerido para la interfaz humano-máquina tales como pilotos, teclas o pantallas.
- El desarrollo del firmware necesario para ejecutar las funciones de conectividad, funciones lógicas y reporte de estados.
- Configuración de la API web necesaria para realizar pruebas de seguimiento en tiempo real del dispositivo.
- Optimización del consumo de energía a través de firmware.

Este trabajo no incluyó:

- El desarrollo de una interfaz web a medida.
- El diseño del aspecto físico del dispositivo.

Capítulo 2

Introducción específica

Este capítulo presenta en detalle los temas aplicados en el desarrollo del trabajo y las definiciones realizadas en el inicio del trabajo. Permite conocer los requisitos y organización de tareas planteados.

2.1. Requerimientos

A continuación, se presentan los requerimientos planteados en la planificación del trabajo:

1. Requerimientos funcionales
 - a) El sistema a desarrollar debe ser un prototipo funcional para pruebas en ambiente de laboratorio.
 - b) El sistema debe reportar en una plataforma de software la posición geográfica de las cavas en tiempo real.
 - c) La plataforma de software deberá permitir la visualización en un mapa de la posición geográfica de las cavas.
 - d) La precisión del reporte de la posición geográfica debe ser de diez metros (10 m).
 - e) Como requisito mínimo, el sistema debe ser capaz de conectarse con dos constelaciones de satélites para obtener la posición geográfica.
 - f) El hardware debe tener botones para que los operarios puedan reportar la ocupación o desocupación de las cavas.
 - g) El hardware debe poseer un display LCD que permita a los operarios visualizar las coordenadas geográficas reportadas por el sistema y el estado de ocupación de la cava.
 - h) El hardware debe poseer pilotos que permitan a los usuarios visualizar e interpretar de forma lógica el estado de ocupación de las cavas.
2. Requerimientos no funcionales
 - a) El sistema debe ser escalable y debe poderse replicar.
 - b) El sistema a desarrollar debe usar componentes electrónicos disponibles en Colombia.

2.2. Estructura del sistema

En la figura 2.1 se observa el diagrama general del prototipo que se construyó. Allí se representa al microcontrolador, que funge como unidad de procesamiento central. Además, se distinguen los componentes de comunicación satelital y celular: un receptor GNSS y un módulo 4G LTE. Se aprecian, las infraestructuras de comunicación que se emplearon para hacer posible la comunicación. Se distingue la infaltable fuente de alimentación para garantizar la energía. Por último, el sistema comprende de una interfaz web como herramienta para la presentación y seguimiento de la información.

En el capítulo siguiente se aborda con más detalle la implementación de cada uno de estos módulos.

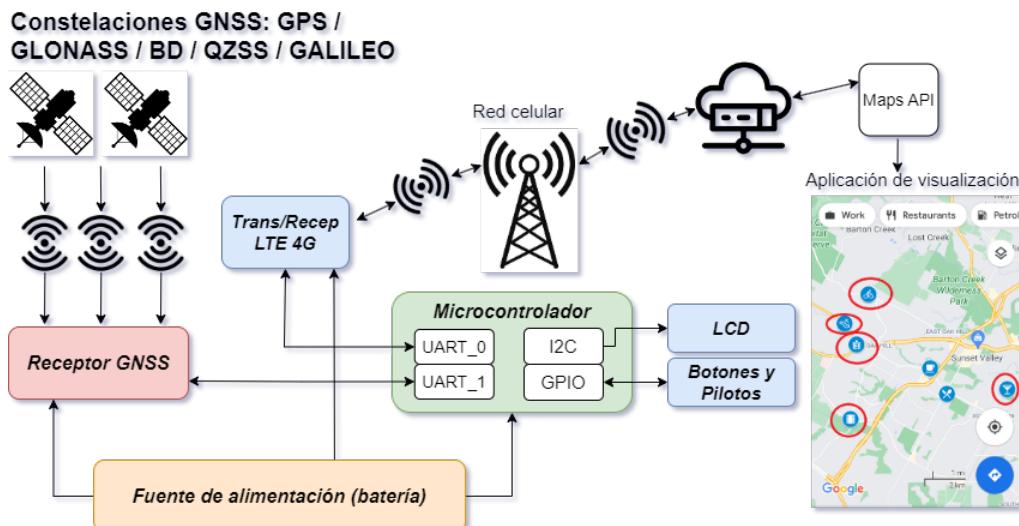


FIGURA 2.1. Diagrama general del sistema.

2.3. Componentes del sistema

En este trabajo se desarrolló un prototipo para realizar el posicionamiento geográfico de cajas móviles de transporte de alimentos. Dicho sistema comprende los siguientes componentes:

- Microcontrolador, que realiza las labores de unidad central de procesamiento.
- Receptor GNSS, para la recepción de datos de posicionamiento.
- Módulo de comunicación 4G, para la conexión a Internet.
- Plataforma de software para la presentación de la información.
- Fuente de alimentación.

Para la selección de los principales componentes se buscaron alternativas en el mercado a partir de los requisitos planteados en la sección 2.1.

2.3.1. Microcontrolador ESP32

Luego de realizar un estudio comparativo entre tres plataformas de microcontroladores disponibles en el mercado colombiano, se seleccionó el microcontrolador ESP32. En la tabla 2.1 se presenta un extracto de la comparación entre las plataformas.

TABLA 2.1. Tabla comparativa de Microcontroladores.

Característica	ESP-WRoom-32	Arduino ME-GA	STM32 F401
CPU	Dual-core Tensilica Xtensa LX6	ATmega2560 (8-bit)	ARM Cortex-M4 @ 84 MHz
Número de núcleos	2	1	1
Frecuencia del procesador (MHz)	160	16	84
ROM	448 kB	256 kB (Flash)	512 kB (Flash)
RAM	520 kB	8 kB (SRAM)	96 kB (SRAM)
Comunicación	Wi-Fi, Bluetooth, 2x UART, 1x SPI, 1x I2C y 1x USB.	1x USB, 2x UART, 1x SPI e 1x I2C	3x USART, 4x SPI, 3x I2C, 1x CAN y 1x USB
Lenguajes de programación	Arduino, C/C++ y Micropython	Arduino y C/C++	Arduino y C/C++
Cantidad de GPIO's	36	54 (Digital), 16 (Analog)	82
Soporte de hardware para RTOS	Sí	No	Sí
		16 (Analog)	

El microcontrolador ESP32 comprende una familia o serie de microcontroladores tipo System on Chip (SoC), desarrollado por la empresa Espressif [ESPRESSIF].

El ESP32 es un sistema de doble núcleo con dos CPU Harvard Architecture Xtensa LX6. Toda la memoria integrada, la memoria externa y los periféricos se encuentran en el bus de datos y/o el bus de instrucciones de estas CPU. Las dos CPU se denominan PRO_CPU y APP_CPU (de protocolo y aplicación), sin embargo, para la mayoría de los propósitos las dos CPU son intercambiables [ESPRESSIF].

Entre las principales características de este microcontrolador se encuentran:

- Espacio de direcciones de 4 GB (32 bits) para el bus de datos y el bus de instrucciones.
- Espacio de direcciones DMA de 328 kB. Trece de sus módulos son capaces de operar con DMA.
- Espacio de direcciones periférico de 512 kB.
- ROM interna de 448 kB.
- SRAM interna de 520 kB.
- Admite hasta 16 MB de Flash SPI o hasta 8 MB de SPI SRAM.

En la figura 2.2 se presenta la placa de desarrollo ESP32-Devkit-V4.

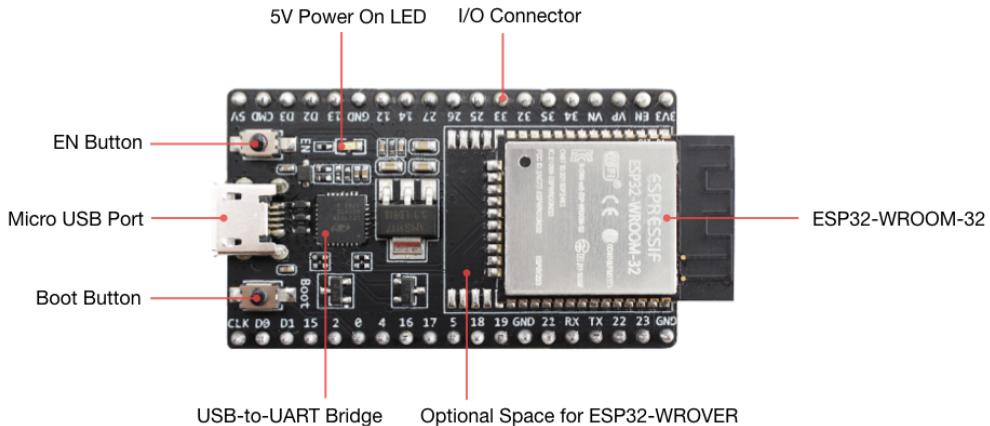


FIGURA 2.2. Placa de desarrollo ESP32-Devkit-V1¹.

A continuación, se presentan los beneficios de haber aplicado el microcontrolador ESP32 en el presente trabajo:

- Es de resaltar la significativa capacidad de cómputo inherente a su diseño, ya que cuenta con un procesador de doble núcleo. Esto fue especialmente útil para implementar multitarea a través de FreeRTOS.
- Incorpora dos módulos de comunicación inalámbrica: Wi-Fi y Bluetooth. Esto permite la conexión con Internet y la conexión en redes de área personal.
- Otro punto a favor importante para los requerimientos del trabajo, es que presenta modos de bajo de consumo energético configurables.
- El fabricante proporciona una documentación extensa de su producto. Además, provee el *framework* ESP-IDF que es bastante potente y está disponible gratuitamente.
- Debido a que es un producto *Open Source* y *Open Hardware*, goza de una comunidad de desarrolladores bastante activa y una gran disponibilidad de documentación. Estas dos características posibilitan que cualquier inconveniente que se consulte tenga una alta probabilidad de ser resuelto.
- Por último, es un dispositivo de bajo costo, con un precio asequible y una relación calidad/prestaciones muy buena. Esto hizo posible abaratar los costos del trabajo.

¹Imagen tomada de: <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/esp32/get-started-devkitc.html>

2.3.2. Receptor GNSS

Un receptor de Sistemas de Navegación Global por Satélite (GNSS, por sus siglas en inglés) es un dispositivo que permite realizar posicionamiento global. Esto lo hace a través de la recepción de señales provenientes de distintas constelaciones de satélites. Estas señales contienen información detallada sobre la posición de un objetivo a nivel mundial, lo que ofrece precisiones que abarcan desde los 10 metros hasta los 0,1 milímetros.

Para este trabajo se seleccionó el módulo Quectel L76 (QL76), que es un receptor GNSS que admite cinco sistemas de navegación y posicionamiento global: GPS, GLONASS, Galileo, BeiDou y QZSS. En la figura 2.3 se puede observar el aspecto físico de dicho módulo [QL76]. Se eligió este debido a la disponibilidad en el mercado local y a que sus características cumplen perfectamente con los requerimientos planteados.



FIGURA 2.3. Módulo Quectel L76².

A continuación, se describen las características principales del módulo QL76 [QL76]:

- La configuración de constelación predeterminada es GPS+GLONASS.
- Admite interfaces de comunicación UART e I2C.
- Ofrece diversos modos de funcionamiento que permiten la optimización del consumo de energía, a saber: GLP, AlwaysLocate™, Standby y Backup
- Tiene un consumo de energía extremadamente bajo.

2.3.3. Módulo 4G

Un transmisor/receptor 4G es un dispositivo terminal que permite la comunicación inalámbrica para dispositivos electrónicos a través de las redes de telefonía móvil de cuarta generación. Estos módulos facilitan la transmisión de datos, mensajes SMS, llamadas y conexión a internet a través de la infraestructura de redes móviles.

Para este trabajo se escogió el SoC (*Sistem on Chip*) 4G SIMA7670SA con tecnología 4G. Soporta las bandas de frecuencia GSM y LTE-FDD. Está optimizado para

²Imagen tomada de: <https://www.quectel.com/product/gnss-l76>

casos de uso de IoT, diseñando bajo la especificación CAT-1. Puede operar en condiciones difíciles, entre las que se pueden mencionar:

- Cobertura en lugares remotos.
- Interferencia por el fenómeno de *handover*.
- Congestión de la red.
- Condiciones meteorológicas adversas.

El SoC también integra el *stack* de protocolos TCP/IP, MQTT. Este se escogió debido a que presenta buenas prestaciones en un tamaño compacto [A7670SA].



FIGURA 2.4. Módulo SIM A7670SA³.

Las características principales de este SoC son:

- Frecuencias de trabajo: LTE-FDD(B1/B2/B3/B4/B5/B7/B8/B28/B66), GSM (850/900/1800/1900MHz).
- Soporta interfaces UART, USB, I2C y GPIO.
- Velocidad de descarga/carga en modo LTE de 10/5 Mbps.
- Velocidad de descarga/carga en modo GPRS/EDGE de 236.8/236.8 Kbps.
- consumo de corriente en modo LTE 3.8 mA.
- consumo de corriente en modo GSM 3.5 mA.

Para el presente trabajo, se utilizó el módulo A7670SA 4G LTE CAT1 MULTIBANDA GSM GPRS IOT (*shield*) comercial basado en el SoC SIM A7670SA, debido a la facilidad para integrarlo al sistema general y que dispone de las interfaces de hardware necesarias para su interconexión con la ESP32, demás de su propio circuito de alimentación y un diseño fiable para evitar las interferencias electromagnéticas. Ver figura 2.5.

³Imagen tomada de: <https://www.simcom.com/product/A7670X.html>

⁴Imagen tomada de: <https://ssdselect.com/gsm-y-gprs-1/4872-a7670sa-sin-gps.html>



FIGURA 2.5. Módulo A7670SA⁴.

Antena Quectel YG0035AA

Debido al requerimiento de conectarse, como mínimo, con dos constelaciones de satélites para obtener la posición geográfica, se debe utilizar una antena capaz de captar las frecuencias de transmisión de al menos dos constelaciones satelitales. Para este proyecto, se considera la antena Quectel YG0035AA. En la figura 2.6 se observa la antena.

La antena está diseñada, según su fabricante, está diseñada con las características de alta eficiencia y excelente rendimiento. Soporta la recepción de señales de la constelación GPS y BeiDou (BD). Presenta las siguientes características eléctricas:

- Antena tipo activa.
- Rango de frecuencias de 1561 MHz a 1575 MHz.
- Impedancia de entrada de 50 ohm
- Ganancia entre 0.5 dBi y 1.4 dBi
- Polarización tipo RHCP.



FIGURA 2.6. Antena Quectel YG0035AA⁵.

⁵Imagen tomada de: <https://www.quectel.com/product/yg0035aa-gnss-active-magnetic-mount-antenna/>

2.3.4. Sistema operativo de tiempo real

Un sistema operativo de tiempo real (RTOS, del inglés: Real-Time Operating System), es un tipo de sistema operativo diseñado específicamente para controlar y gestionar tareas de tiempo real en sistemas embebidos, como microcontroladores y sistemas integrados. La característica principal de un RTOS es que puede responder a eventos y ejecutar tareas de manera predecible y en un intervalo de tiempo garantizado.

El firmware que se implementó en este trabajo está basado en FreeRTOS™. En el apartado siguiente, se explica qué es y cuáles son las ventajas que se obtuvieron con su utilización.

FreeRTOS™

FreeRTOS™ es un sistema operativo de tiempo real de código abierto diseñado para sistemas embebidos y microcontroladores. Es apto para microcontroladores y microprocesadores pequeños [FreeRTOS]. A continuación, se presentan las ventajas de haber implementado este sistema operativo en el trabajo.

- Debido a que proporciona un kernel multitarea, permitió generar concurrencia entre tareas, como lo fueron la recepción y parseo de tramas GNSS, la transmisión de datos en tiempo real.
- Gracias a la temporización precisa, se pudo asegurar la transmisión de datos de posicionamiento en tiempo real.
- Permitió la gestión de la memoria y los periféricos UART del microcontrolador. De hecho, con FreeRTOS™, es posible gestionar todos los recursos del dispositivo con colas y semáforos. Al mismo tiempo, el kernel no representa una carga pesada para la capacidad de cómputo del microcontrolador utilizado.
- Se pudo acceder a las ventajas de un RTOS de forma libre con la licencia MIT de FreeRTOS™.
- El código desarrollado es altamente portable, lo que permitirá escalar o transferir fácilmente el prototipo hacia otras plataformas.

2.3.5. Servicio web para mapas

Un servicio de mapas es un conjunto de herramientas y servicios de software que permiten que los mapas estén disponibles en la web. A menudo este es ofrecido por un proveedor especializado.

Para este trabajo se creó una aplicación web basada en los servicios de la biblioteca para JavaScript Leaflet. Esta es de código abierto y ofrece todas las funciones cartográficas necesarias para la creación de aplicaciones que requieren el uso de mapas, como es el caso de la geolocalización.

En la tabla 2.2 se presenta una comparación entre tres bibliotecas que prestan herramientas para la creación de mapas en la web:

TABLA 2.2. Tabla comparativa de APIs de mapas.

Característica	Google Maps API	Leaflet.js	Carto.js
Licencia	De pago, con un nivel gratuito limitado.	Open-source (BSD 2-clause License).	De pago, con un nivel gratuito limitado.
Facilidad de uso	Alta	Alta	Media
Documentación	Extensa y detallada.	Buena, pero poco extensa. Complementa con muchos ejemplos.	Buena, orientada a usuarios con conocimientos en GIS
Soporte y Comunidad	Amplio soporte y gran comunidad.	Gran comunidad open-source.	Comunidad activa, pero reducida.
Capacidades de Mapa	Incluye vistas de satélite, terreno, y Street View.	Básico, pero extensible con plugins.	Enfocado en visualización de datos geoespaciales.
Personalización	Alta, mediante opciones y controles avanzados.	Muy alta, altamente personalizable con plugins.	Alta, especialmente en visualización de datos.
Rendimiento	Muy bueno, optimizado por Google.	Muy bueno, especialmente en navegadores modernos.	Bueno, puede ser pesado con grandes conjuntos de datos.
Integración de Datos	Fácil integración con servicios de Google como Places, Routes, etc.	Necesita plugins adicionales para integración avanzada de datos.	Muy bueno para integración de datos geoespaciales y análisis.
Soporte para Móviles	Excelente, completamente responsive.	Muy bueno, compatible con dispositivos móviles.	Bueno, pero depende del tamaño de los datos.
Características Avanzadas	Geocoding, Directions, Distance Matrix, Places API, etc.	Depende de plugins externos.	Ánálisis espacial avanzado, mapas temáticos, SQL API.
Facilidad de Integración	Muy fácil con otros servicios de Google.	Fácil con bibliotecas JavaScript estándar.	Requiere conocimiento de GIS y SQL para máxima utilidad.

Capítulo 3

Diseño e implementación

Este capítulo explica los componentes del sistema seleccionados y la descripción detallada de cada módulo de hardware y software desarrollados en el trabajo. Utilizando las definiciones y criterios descritos en capítulos anteriores, se comprende el diseño final del prototipo.

3.1. Diseño e implementación general del sistema

El diseño e implementación general del sistema se subdividió en 3 partes:

1. Diseño e implementación del hardware.
2. Diseño e implementación del firmware.
3. Diseño e implementación de la interfaz web.

A continuación, se describen cada una de estas partes.

3.2. Diseño e implementación de hardware

En esta sección, se describen las consideraciones de diseño del hardware del sistema.

3.2.1. Diseño del Hardware

En la figura 3.1 se presenta la arquitectura del hardware desarrollado. Se incluyen todos los componentes descritos en las secciones anteriores. El sistema cuenta con 2 antenas correspondientes a la comunicación GNSS y 4G.

Como parte de la interfaz humano-máquina, se cuenta con una pantalla LCD de 16x2, dos botones para la indicación manual de estado ocupado/desocupado y dos pilotos que permiten visualizar el estado de ocupación actual.

En el diagrama se pueden ver los protocolos de comunicación entre módulos, entre los que se destacan:

- El protocolo UART para la comunicación entre el módulo 4G SIM A7670SA por el puerto UART 0 y el módulo GNSS Quectel L76 por el puerto UART 1.
- El protocolo I2C para la comunicación con la pantalla LCD.

Se presenta una fuente de alimentación correspondiente a una batería de 12 VDC. Debido a que los componentes del sistema operan con tensiones diferentes, se

hizo necesaria la implementación de circuitos adicionales. En el diagrama, estos circuitos se denominan «Fuente». Estos suplen 2 niveles de tensión distintos:

- 3,3 VDC para el Quectel L76.
- 5 VDC para la ESP32, el módulo SIM A7670SA y la pantalla LCD.

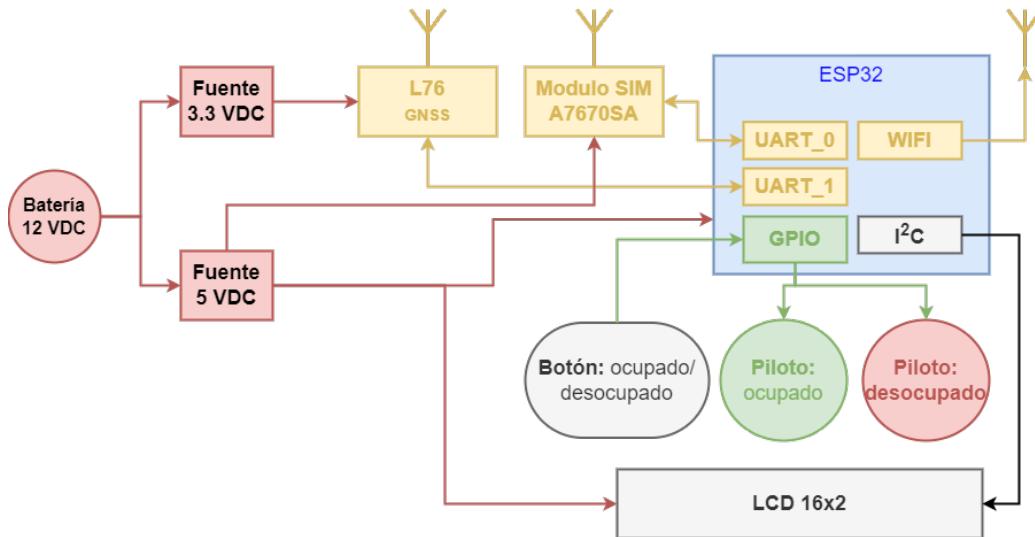


FIGURA 3.1. Arquitectura del hardware desarrollado.

3.2.2. Implementación del Hardware

En la etapa de implementación del hardware, se trabajó en 2 placas de circuito impreso.

1. Placa base del sistema.
2. Adaptador para el módulo Quectel L76.

Placa base del sistema

La placa base se encarga de interconectar los módulos y periféricos principales del sistema. Por lo tanto, en el apéndice A se presenta el circuito esquemático correspondiente a esta implementación. En esta se empleó el regulador MIC29302WU, con una configuración de ajuste por defecto en 5 VDC, para la alimentación de la ESP32, del módulo SIM A7670SA, la placa adaptador para el módulo Quectel L76 y la pantalla LCD. En la figura 3.2 se puede apreciar el diseño final del PCB en un modelo de dos dimensiones. Además, en la figura 3.3 se observa el modelo en tres dimensiones.

Los componentes U1, U2 y H3, corresponden a las ranuras donde se conectan la ESP32 DevKit V4, el adaptador para el módulo Quectel L76, y el módulo SIM A7670SA, respectivamente. Las terminales marcadas con CN6 y CN7 permiten la conexión de los pilotos que indican la ocupación de la cava. Las terminales CN4 y CN5 corresponden a las conexiones de los botones de que dispondrá el operario de la cava para la indicación de la ocupación. Por otro lado, las terminales CN2 y CN3 corresponden a la interfaz I2C y la alimentación de la pantalla LCD, respectivamente. La terminal CN1 corresponde a la entrada de alimentación del sistema. Finalmente, las terminales CN9 y CN11 externalizan los pines GPI 34 y

GPI 35, y GPIO 5 y GPIO 18 de la ESP32, respectivamente; esto para facilitar, en una etapa posterior, la agregación de nuevas funcionalidades.

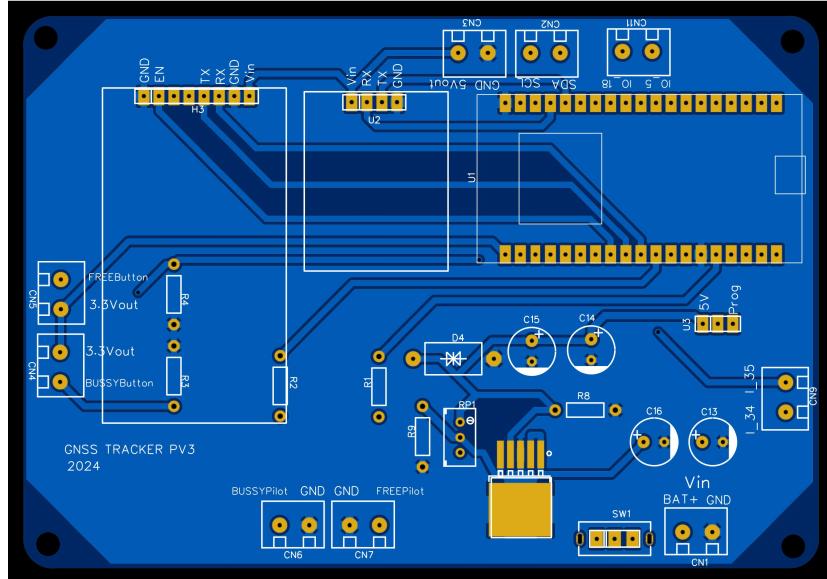


FIGURA 3.2. Modelo 2D de la placa base del sistema.

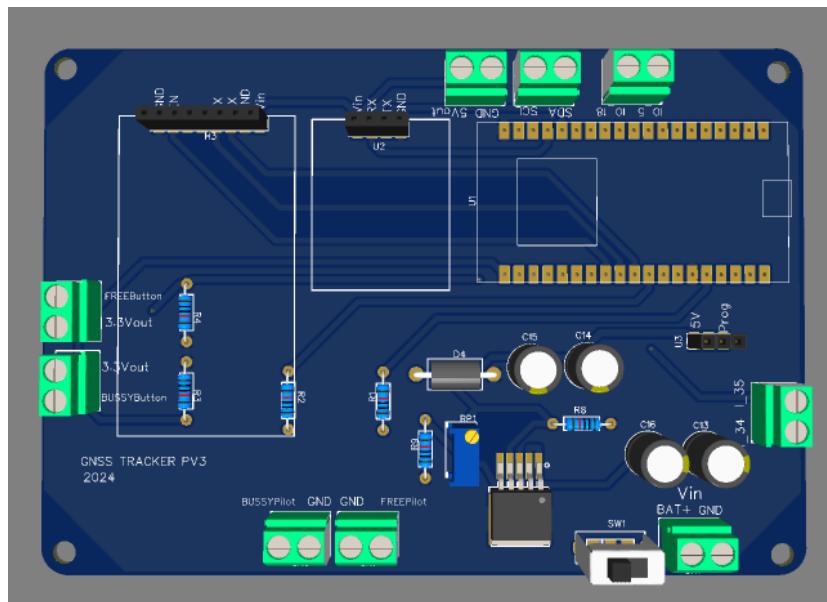


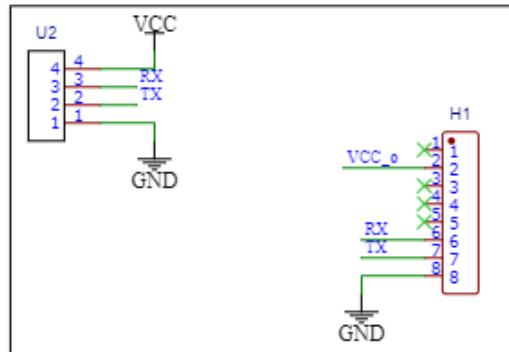
FIGURA 3.3. Modelo 3D de la placa base del sistema.

Adaptador para el módulo Quectel L76

El circuito esquemático del adaptador L76 se encuentra en la figura 3.4. En este esquemático se puede observar la fuente de 3.3 VDC de entrada de alimentación, la conexión UART y la conexión entre el módulo y el microcontrolador ESP32.

En las figuras 3.5 y 3.6, se muestran los modelos 2D y 3D del circuito diseñado.

Conexión entre módulos y entrada de alimentación



Fuente de 3.3 VDC

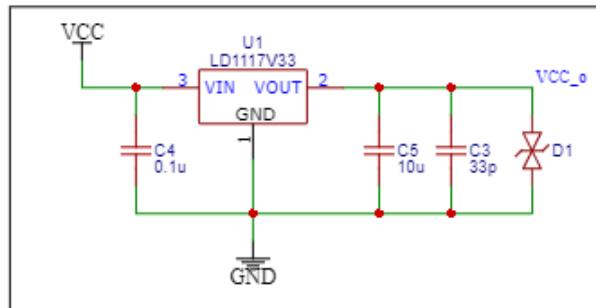


FIGURA 3.4. Esquemático del adaptador L76.

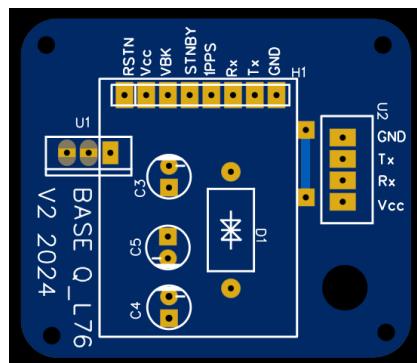


FIGURA 3.5. Modelo 2D del adaptador L76.

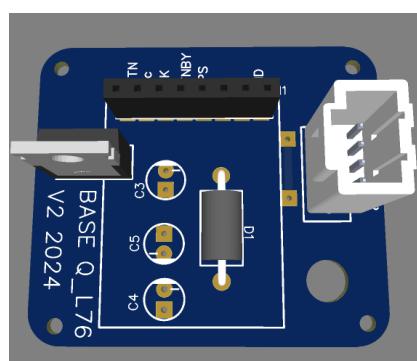


FIGURA 3.6. Modelo 3D del adaptador L76.

3.3. Diseño e implementación del firmware

En esta sección se presentan las consideraciones de diseño y el proceso de implementación del firmware.

3.3.1. Diseño del firmware

El propósito del firmware desarrollado es satisfacer los requerimientos de conectividad, administración del hardware e interfaz con el usuario. A continuación se describen las principales funcionalidades del firmware:

1. Realizar las funciones de conexión a Internet a través de la red celular, a través de la comunicación por comandos AT-UART con el módulo SIM A7670SA, para asegurar la conexión con el *Broker* MQTT.
2. Comunicación a través de tramas NMEA por el puerto UART con el módulo Quectel L76. Para la recepción e interpretación de los datos de geolocalización de las constelaciones GPS y BeiDou.
3. Monitoreo de los botones de ocupación a través de interrupciones y visualización de los estados de ocupación a través de los pilotos.
4. Generación de interfaz visual a través de la comunicación I2C con la pantalla LCD 16x2.
5. Administración de la energía del sistema a través de los modos de bajo consumo.
6. Administración de los recursos del microcontrolador a través de la implementación del sistema operativo en tiempo real FreeRTOS.

Arquitectura del firmware

De acuerdo con lo planteado anteriormente, en la figura 3.7 se presenta un diagrama de componentes UML, que representa la arquitectura del sistema general, con énfasis en los componentes del firmware.

En este diagrama (figura 3.7) se representan como subsistemas (*Subsystem*) los componentes de hardware como la ESP32, la pantalla LCD, el módulo 4G SIM A7670SA, los botones y los pilotos. Así mismo, se entienden como subsistemas los componentes de firmware proporcionados por el fabricante del microcontrolador, que corresponden a los drivers GPIO, UART y el bus I2C. Además, se hizo necesaria la implementación de una biblioteca propia para el control de la pantalla LCD a través del bus I2C.

De acuerdo con esto, los componentes del firmware se establecen en la ESP32, que corresponde al dispositivo programable encargado de realizar el control general del sistema.

En el diagrama de la figura 3.7 se presentan los componentes de firmware más relevantes del sistema y la forma en que interactúan entre ellos. A continuación, se describe la funcionalidad de cada componente del firmware:

1. El firmware inicia con la tarea *init_mqtt_server_task()*, resaltada con color rojo, la cual se encarga de tres funciones primordiales: realizar la inicialización de los servicios MQTT que proporciona el módulo 4G (SIM A7670SA),

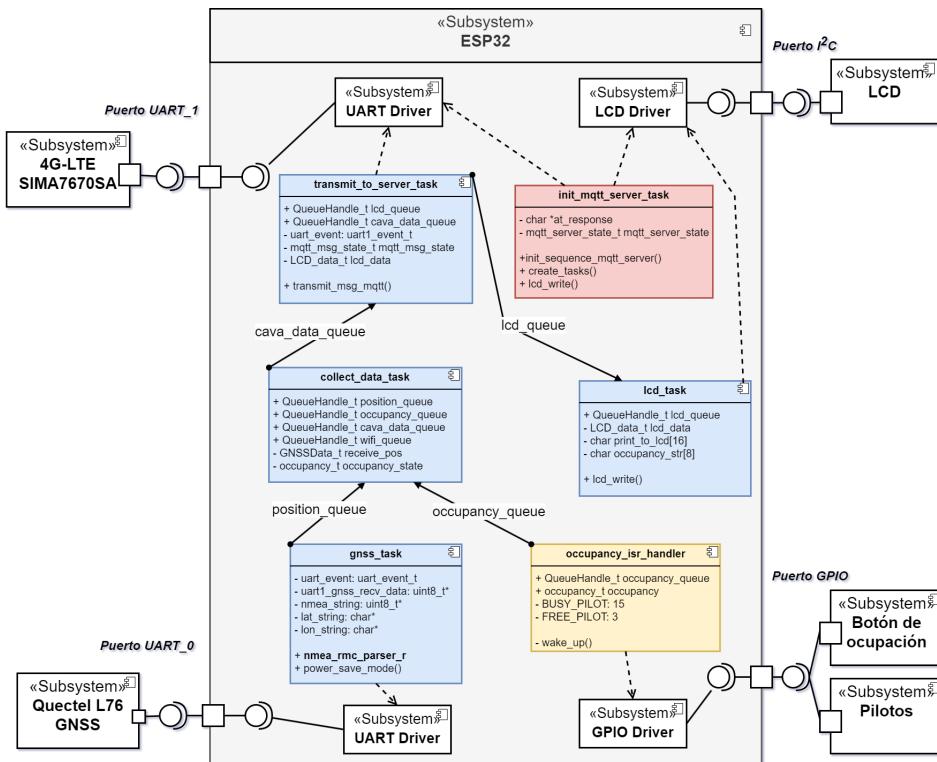


FIGURA 3.7. Arquitectura del firmware.

realizar la conexión con el *Broker MQTT* remoto y, si todo sale bien, desencadena la creación de las demás tareas del sistema. Esta tarea utiliza líneas de comandos AT para comunicarse con el módulo 4G.

2. Tarea *collect_data_task()*, se encarga de recibir a través de colas, la información relacionada con la ocupación y la posición de la cava, dicha información proveniente de la tarea *gnss_task()* y del manejador de interrupciones *occupancy_isr_handler()*, respectivamente. Cuando esta tarea obtiene la información mencionada, las transmite a la tarea *transmit_to_server_task()*, a través de la cola *cava_data_queue*.
3. Manejador de interrupción *occupancy_isr_handler()*, resaltada con color amarillo, es la función que se ejecuta siempre que ocurre una interrupción debida a los botones de ocupación de la cava, a saber, ocupado-desocupado. Este manejador de interrupción detecta qué botón se presionó y comunica a la tarea *collect_data_task()* el estado de ocupación, a través de la cola *occupancy_queue*. Además, esta tarea genera una acción de *wakeup* del sistema cuando se encuentra en modo ahorro de energía.
4. Tarea *gnss_task()*, se encarga primordialmente de detectar la recepción de una trama NMEA del tipo GNRMC, lo que asegura que la posición ha sido calculada con señales de dos o más constelaciones satelitales; en este caso, GPS y BeiDou. La detección de tramas se realiza a través de interrupciones del módulo UART de la ESP32. Esta tarea descarta todas las tramas que no sean compatibles con el estándar NMEA y todas las que no sean del tipo GNRMC. Además, se encarga de la extracción de la información de las tramas RMC a través de la función *nmea_rmc_parser_r()*. Esta última es una implementación de tipo *thread-safe* para asegurar la compatibilidad

con FreeRTOS, ya que permite la multitarea. Además, comunica la posición actual a la tarea `collect_data_task()`, a través de la cola `position_queue`. Por último, esta tarea verifica si la posición no ha cambiado en los últimos 5 minutos, en caso de no cambiar, colocará al sistema completo en modo ahorro de energía.

5. Tarea `transmit_to_server_task()`, se encarga de ejecutar la secuencia de transmisión de datos a través del protocolo MQTT, con el módulo 4G. La comunicación entre microcontrolador y el módulo se realiza a través de comandos AT. Esta tarea recibe los datos provenientes de la tarea `collect_data_task()`, los transmite con la función `transmit_msg_mqtt()`. Al finalizar, construye un paquete de información con los datos de ocupación, posición y resultado de la transmisión MQTT, para ser transferido a través de la cola `lcd_queue` hacia la tarea `lcd_task()`.
6. Tarea `lcd_task()` se encarga de administrar adecuadamente el acceso a los recursos de la pantalla LCD, para evitar sobrescritura o pérdida de información relevante que deba ser mostrada a través de la pantalla LCD.

3.3.2. Implementación del firmware

El firmware se implementó en el *framework* ESP-IDF desarrollado por la empresa ESPRESSIF, para las series de SoCs ESP32, ESP32-S y ESP32-C. Este *framework* proporciona un SDK apto para el desarrollo de cualquier aplicación genérica en estas plataformas, a partir de lenguajes de programación C y C++. Además, está optimizado para aplicaciones de Internet de las Cosas (IoT, del inglés *Internet of Things*).

También se utilizó el entorno de desarrollo (IDE) *PlatformIO*, que se ejecuta como una extensión nativa en el IDE de *Visual Studio Code*. *PlatformIO* permite usar todas las funcionalidades de ESP-IDF de una manera fiable y ágil, sin enmascarar funcionalidades o configuraciones avanzadas del *framework*. Además, permite ejecutar las versiones mas estables de ESP-IDF, proporciona soporte por la comunidad de desarrolladores, es multiplataforma y de software libre.

Para la implementación del firmware se utilizó el lenguaje de programación C y se implementó el sistema operativo en tiempo real FreeRTOS. Una ventaja importante que ofrece ESP-IDF es que está desarrollado sobre FreeRTOS, por tanto, el firmware se ejecuta nativamente en este.

Por petición del cliente, la codificación correspondiente al firmware no podrá ser mostrada, sin embargo, se empleará pseudocódigo para sustentarla y no comprometer la calidad del trabajo desarrollado.

Flujo general del firmware

Gracias a la implementación en FreeRTOS, el firmware tiene la capacidad de ejecutar multitarea. El flujo general del programa se muestra en las figuras 3.8, 3.9 y 3.10. Como se puede ver en dicho diagrama y en el diagrama de la arquitectura (ver figura 3.7), las diferentes tareas se ejecutan una independiente de la otra. Sin embargo, existe un flujo de información que se sucede de una tarea a otra. Como mecanismo de comunicación entre tareas, se utilizan las colas. Este comportamiento del firmware es relevante desde el punto de vista en que el sistema

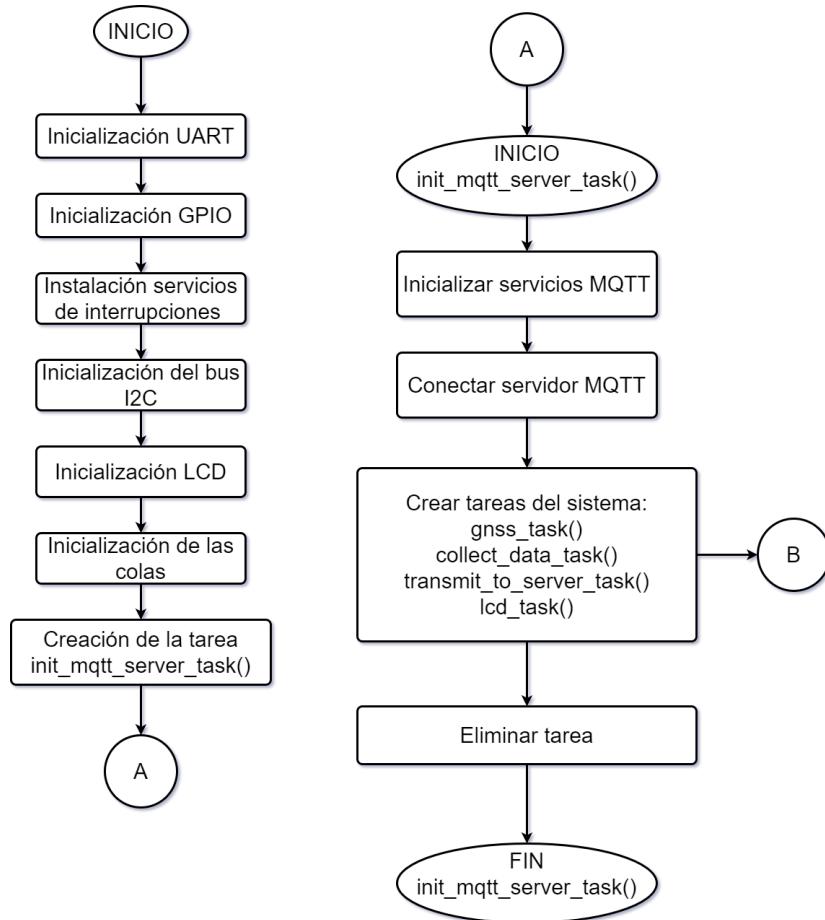


FIGURA 3.8. Diagrama de flujo del firmware (parte 1).

puede estar recibiendo, procesando y transmitiendo datos de posicionamiento de manera casi concurrente.

Implementación de las tareas del sistema

En los códigos, 3.1, 3.2, 3.3, 3.4, 3.5 y 3.6 se presenta la implementación en pseudocódigo de cada una de las tareas principales del firmware.

CÓDIGO 3.1. Pseudocódigo de la función init_mqtt_server_task().

```

1  init_mqtt_server_task()
2  {
3      while ("READY" == SIM_A7670SA)
4          Esperar
5      Inicializar servicios MQTT en SIM_A7670SA
6      while (4 == intentos)
7          Conectar con Broker MQTT en la nube
8          if ("OK" == Conexion_Servidor)
9              Crear tareas:
10                 gnss_task()
11                 collect_data_task()
12                 transmit_to_server_task()
13                 lcd_task()
14             Eliminar: init_mqtt_server_task
  
```

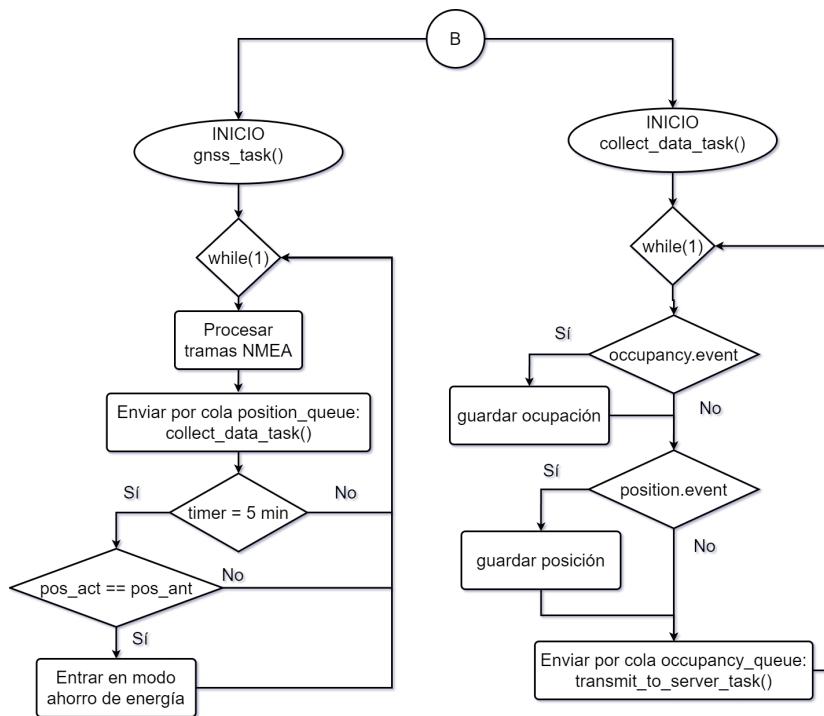


FIGURA 3.9. Diagrama de flujo del firmware (parte 2).

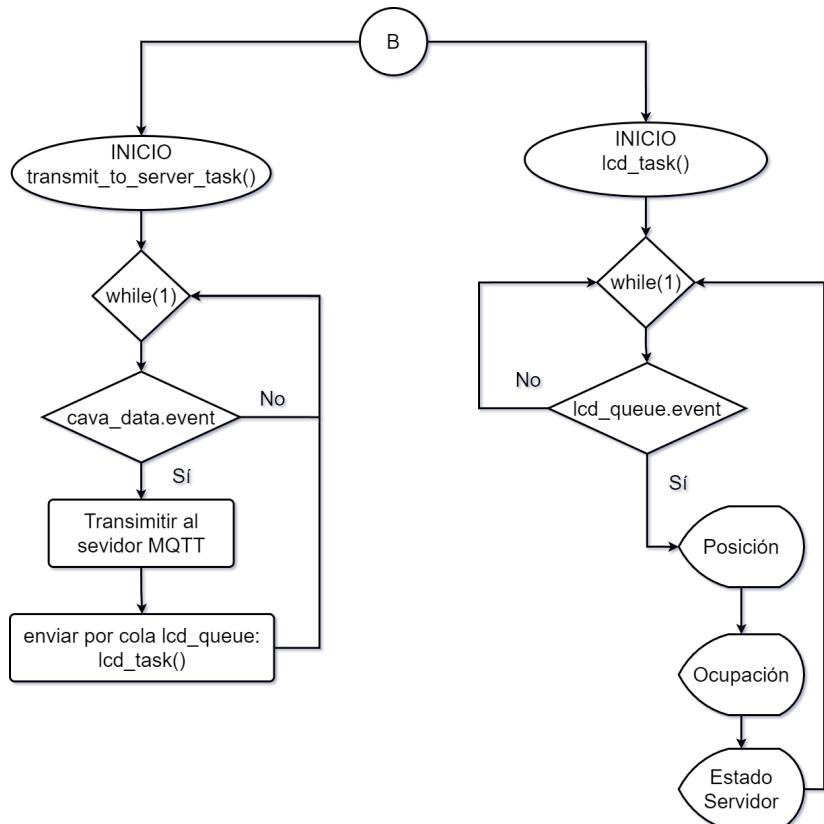


FIGURA 3.10. Diagrama de flujo del firmware (parte 3).

```

16           break
17     else
18       Reportar por LCD: Falla conexión Broker
19       Eliminar: init_mqtt_server_task
20   }

```

CÓDIGO 3.2. Pseudocódigo de la función gnss_task().

```

1 gnss_task()
2 {
3   while (1)
4     if (DATA == evento_UART_0)
5       Procesar trama:
6         posicion_actual =
7           ↪ nmea_rmc_parser_r()
8       Enviar por cola position_queue:
9         ↪ posicion_actual
10      if (Temporizar 5)
11        if (posicion_anterior ==
12          ↪ posicion_actual)
13          sleep_mode:
14            ↪ Modulo_celular
15          sleep_mode: Modulo_GNSS
16          sleep_mode: ESP32
17
18        posicion_anterior = posicion_actual
19   }

```

CÓDIGO 3.3. Pseudocódigo de la función occupancy_isr_handler().

```

1 occupancy_isr_handler()
2 {
3   Capturar botón presionado
4   switch (bonton)
5     case boton_libre:
6       Setear los pilotos como libre
7       Enviar por cola occupancy_queue: "Libre"
8
9     case boton_ocupado:
10      Setear los pilotos como ocupado
11      Enviar por cola occupancy_queue: "
12        ↪ Ocupado"
13   }

```

CÓDIGO 3.4. Pseudocódigo de la función collect_data_task().

```

1 collect_data_task()
2 {
3   while (1)
4     if (DATA == occupancy_queue.event)
5       datos_cava.occupancy = occupancy
6
7     if (DATA == position_queue.event)
8       datos_cava.position = position
9

```

```

10
11         Enviar por cola cava_data_queue: datos_cava
12 }
```

CÓDIGO 3.5. Pseudocódigo de la función `transmit_to_server_task()`.

```

1 transmit_to_server_task()
2 {
3     posicion_anterior
4     while (1)
5         if (DATA == cava_data_queue.event)
6             Transmitir al Broker MQTT:
7                 transmission_state =
8                     → transmit_msg_mqtt ()
9             Enviar por cola lcd_queue: cava_data+
10                → transmission_state
11 }
```

CÓDIGO 3.6. Pseudocódigo de la función `lcd_task()`.

```

1 lcd_task()
2 {
3     while (1)
4         if (DATA == lcd_queue.event)
5             Escribir_LCD: position.lat
6             Escribir_LCD: position.long
7             Escribir_LCD: position.time
8             Escribir_LCD: position.val // Validez de
9                 → la posición
10            Escribir_LCD: occupancy
11            Escribir_LCD: server_state
12 }
```

Otros componentes del firmware

Además de las tareas y funciones descritas en la sección 3.7, fue necesario implementar varios componentes de firmware. Sin embargo, por ser los más relevantes, se mencionan los siguientes:

1. Un driver para el manejo de la pantalla *LCD RGB Backlight* de 16x2 que se llamó *lcd_i2c_grove.h*. Esta pantalla presenta interfaz de comunicación I2C, para reducir la cantidad de pines utilizados por el microcontrolador. Este driver presenta las funciones descritas en el código 3.7.
2. Una función para inicializar los servicios MQTT en el módulo SIM A7670SA y conectarse al *Broker MQTT*, que se llamó *init_sequence_mqtt_server()*. Esta función implementa comandos AT específicos para el módulo en mención. Retorna el estado de conexión con el *Broker MQTT* o en su defecto los errores encontrados en el proceso. En el código 3.8 se muestra el pseudocódigo que describe su funcionamiento.

3. Una función para encapsular la funcionalidad de transmitir al *Broker MQTT* una vez se ha establecido la conexión. Esta es *transmit_msg_mqtt()*. Esta función implementa comandos AT específicos del módulo SIM A7670SA y retorna el estado de transmisión del mensaje. En el código 3.9 se muestra el pseudocódigo que describe su funcionamiento.
4. Una función que permite extraer la información de las cadenas tipo GNRMC recibidas por el módulo Quectel L76. Esta es una función de tipo *thread-safe* para facilitar la multitarea, que se nombró como *nmea_rmc_parser_r()*. En el código 3.10 se muestra el pseudocódigo que describe su funcionamiento.

CÓDIGO 3.7. Funciones principales del driver LCD.

```

1 /**
2  * @brief Esta función inicializa las funcionalidades I2C y la
3  *        configuración de escritura de la pantalla
4  */
5 void lcd_init();
6
7 /**
8  * @brief Esta función escribe en la pantalla LCD el texto
9  *        deseado.
10 *
11 * @param row: Columna desde la cual se inicia la escritura
12 * @param column: Fila sobre la cual se desea escribir
13 * @param str: Texto que se desea escribir
14 */
15 void lcd_write(uint8_t row, uint8_t column, char *str);
16
17 /**
18  * @brief Esta función limpia o borra todos los caracteres de la
19  *        pantalla LCD.
20  */
21 void lcd_clear();
22
23 /**
24  * @brief Esta función cambia el color de fondo de la pantalla
25  *        LCD según el modelo de color RGB.
26  *
27 * @param r: valor para cantidad de color rojo.
28 * @param g: valor para cantidad de color verde.
29 * @param b: valor para cantidad de color azul.
30 */
31 void lcd_set_RGB(unsigned char r, unsigned char g, unsigned char
32                  b);

```

CÓDIGO 3.8. Pseudocódigo de la función init_sequence_mqtt_server().

```

1 init_sequence_mqtt_server(uart_event_t uart1_event, char *
2                           at_response)
3 {
4     Enviar AT: iniciar el servicio MQTT
5     if ("OK" == respuesta)
6         Enviar AT: adquirir cliente MQTT
7         if ("OK" == respuesta)

```

```

8     Enviar AT: conectar con Broker MQTT
9         if ("OK" == respuesta)
10            return: MQTT_SERVER_OK
11        else
12            return: MQTT_FAIL_INIT_SERVER
13    else
14        return: MQTT_FAIL_ADD_CLIENT
15 else
16    return: MQTT_FAIL_INIT_SERVICE
17 }
```

CÓDIGO 3.9. Pseudocódigo de la función init_sequence_mqtt_server().

```

1 transmit_msg_mqtt(char * mqtt_payload, char * topic,
2                     ↪ uart_event_t uart1_event, char * at_response)
3 {
4     Enviar AT: configurar topic
5     if ("OK" == respuesta)
6         Enviar AT: cargar payload
7         if ("OK" == respuesta)
8             Enviar AT: publicar
9             if ("OK" == respuesta)
10                return: MQTT_MSG_OK
11            else
12                return: MQTT_MSG_FAIL
13        else
14            return: MQTT_TOPIC_FAIL
15    else
16        return: MQTT_TOPIC_FAIL
17 }
```

CÓDIGO 3.10. Pseudocódigo de la función nmea_rmc_parser_r().

```

1 nmea_rmc_parser_r(char *nmeaString, GNSSData_t *gnssData)
2 {
3     Verificar trama inicia con "$"
4     if ("$" == token_0)
5         Dividir trama en tokens por ","
6         Verificar que el primer token sea "GNRMC"
7         if("GNRMC" == token_1)
8             time = token_2
9             if("A" == token_3)
10                 latitud = convertir_DMS_to_DD(
11                     ↪ token_4, token_5)
12                 longitud = convertir_DMS_to_DD(
13                     ↪ token_6, token_7)
14             else
15                 return: NMEA_NO_VALID
16             else
17                 return: NMEA_NO_RMC
18         else
19             return: NMEA_NO_VALID
20 }
```

3.4. Diseño e implementación de la interfaz web

En esta sección se presentan las consideraciones de diseño y la metodología de implementación de la interfaz web.

3.4.1. Diseño de la interfaz web

Para el diseño de la interfaz web, se tuvo en cuenta satisfacer la historia de usuario descrita en la planificación de este trabajo y que motiva al desarrollo de este componente:

Como usuario final quiero ver una página web con un mapa que me muestre la posición en tiempo real de todas las cavas y su estado de ocupación para realizar un seguimiento de estas y organizar la logística del reparto de productos en la cadena de frío.

De acuerdo con lo anterior y con el análisis de la sección 2.3.5, se escogió la biblioteca *leaflet.js*¹ para la creación de mapas en la web, debido a que la empresa Cloud Technologies SAS, integra sus desarrollos web con tecnologías web con JavaScript y a que en la etapa de desarrollo del producto no se requerirá incurrir en gastos para el despliegue en pruebas o en entornos relevantes.

El desarrollo de la aplicación web se llevó a cabo siguiendo la arquitectura mostrada en la figura 3.11. En ella se puede apreciar

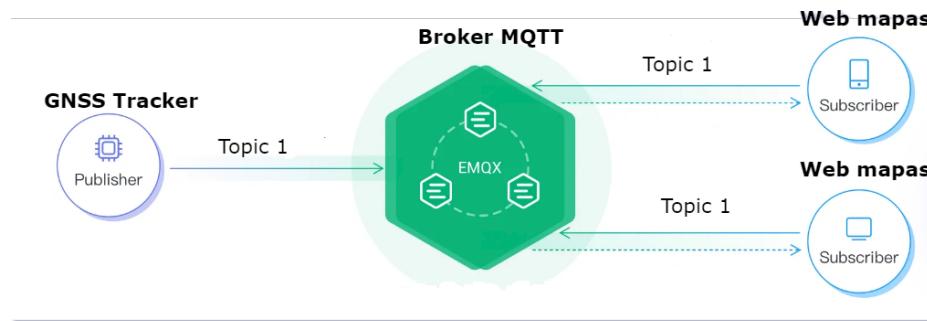


FIGURA 3.11. Arquitectura MQTT².

En esta arquitectura, el centro de la comunicación es el *Broker MQTT*, donde se centralizan los mensajes de los *publishers* para ser transmitidos hacia los dispositivos *subscribers*, a través de canales específicos, llamados tópicos. El *Broker MQTT* es responsable de recibir conexiones iniciadas por el cliente y reenviar mensajes enviados por el cliente a otros clientes elegibles [MasteringMQTT].

Para este trabajo, se utilizó el *Broker EMQX*³, que es utilizado para aplicaciones de IoT [MasteringMQTT].

La aplicación web al igual que el hardware desarrollado, son clientes del *Broker EMQX*. El hardware es el encargado de publicar mensajes con los datos de posicionamiento, estado ocupación, validez de la posición, la hora y la referencia de la cava. Por otro lado, la aplicación web recibe dichos datos y los hace visibles al

¹Documentación en: <https://leafletjs.com/index.html>

²Imagen adaptada de: <https://www.emqx.com/en/blog/mqtt-5-introduction-to-publish-subscribe-model>

³Documentación en: <https://www.emqx.io/>

usuario a través un mapa y otros elementos de visualización web. El usuario final puede visualizar

3.4.2. Implementación de la interfaz web

Para la implementación de la interfaz web, se utilizaron las siguientes tecnologías:

- HTML: Se utiliza para estructurar la página web.
- CSS (Bootstrap): Se usa para dar estilo a la página, para crear un diseño *responsive* y facilitar la implementación ya que cuenta con componentes predefinidos.
- lenguaje de programación JavaScript, características y bibliotecas de éste. A continuación, se detallan las más relevantes para este trabajo:
 - jQuery: Biblioteca de JavaScript utilizada para facilitar la manipulación del DOM, eventos y AJAX.
 - Leaflet.js: Biblioteca de JavaScript para mapas interactivos.
 - mqtt.js: Biblioteca de JavaScript para conectarse a un broker MQTT, permitiendo la comunicación en tiempo real entre el navegador y el broker.
- *IDE Visual Studio Code* para realizar la codificación.
- Docker como plataforma para la etapa de desarrollo, donde se ejecutó el EMQX.
- *Broker* EMQX.

3.4.3. Metodología de la implementación

A continuación, se presenta el proceso llevado a cabo para la implementación de la interfaz web.

1. Configuración Inicial:

- Se diseñó la estructura de la página web usando HTML y Bootstrap. En este punto se utilizaron las bibliotecas necesarias para la funcionalidad del mapa y la conexión MQTT. En la figura ?? se observa la versión desarrollada por el autor. Además, en la figura ?? se muestra la versión implementada en el servidor de la empresa Cloud Technologies SAS, la cual fue migrada y ajustada por su equipo de desarrolladores web.

2. Configuración del Mapa:

- Con Leaflet.js se inicializa un mapa centrado en una posición predeterminada y se añade una capa de OpenStreetMap para mostrar el mapa base.

3. Configuración de la Conexión MQTT:

- Se configuran las opciones de conexión al broker MQTT, a saber URL del broker, credenciales de usuario, parámetro *keepalive* y el parámetro *connectTimeout*. Se establece la conexión con el *Broker* utilizando mqtt.js. Por último, se realiza la suscripción al tópico que recibe los datos de ubicación del dispositivo GNSS.

4. Gestión de Mensajes:

- Se define la función `get_cava_data()` para procesar los mensajes recibidos del broker. Se convierten los mensajes a formato JSON y se extraen los datos de la cava. Se actualiza el mapa con la nueva posición de la cava, los demás datos actualizados y se muestra la ruta seguida.

5. Actualización del Mapa:

- Se elimina el marcador anterior y se añade un nuevo marcador en la nueva posición. Se actualiza la vista del mapa para centrarse en la nueva posición y se muestra la ruta recorrida por la cava utilizando una polilínea.

En el código 3.11 se muestra la implementación HTML de la interfaz web desarrollada por el autor, la cual sirvió de maqueta para que el equipo de desarrollo web de la empresa Cloud Technologys SAS lo implementara en sus servidores. Sin embargo, por petición del cliente, no es posible compartir el código implementado en sus servidores.

Además, en el código 3.12, se presenta la implementación de la dinámica de conexión del dispositivo y las funcionalidades necesarias para la inicialización del mapa y las conexiones

CÓDIGO 3.11. Pseudocódigo para la implementación HTML de la página web.

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Cava mapa</title>
5
6     <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" crossorigin="anonymous">
7     <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js" crossorigin="anonymous"></script>
8     <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.5.3/dist/umd/popper.min.js" crossorigin="anonymous"></script>
9     <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js" crossorigin="anonymous"></script>
10
11    <link rel="stylesheet" type="text/css" href="leaflet.css" />
12
13  </head>
14  <body>
15
16    <h1> GNSS Cava: </h1> <h2> Cava001 </h2>
17
18    <div class="container mt-4">
19      <div class="row">
20        <div class="col-md-4">
21          <div class="card mb-3">
22            <div class="card-body">
```

```

24         <h5 class="card-title">Dispositivo: </h5>
25             ↵ >
26             <p class="card-text" id="cava_name">h</p>
27             ↵ >
28         </div>
29     </div>
30     <div class="card_mb-3">
31         <div class="card-body">
32             <h5 class="card-title">Posicion: </h5>
33             <p class="card-text" id="lat"></p>
34             <p class="card-text" id="lng"></p>
35         </div>
36     </div>
37     <div class="card_mb-3">
38         <div class="card-body">
39             <h5 class="card-title">Estado: </h5>
40             <p class="card-text" id="estado"></p>
41         </div>
42         <div class="form-group">
43             <label for="exampleSelect">Seleccionar cava
44                 ↵ </label>
45             <select class="form-control" id="cava_select"
46                 ↵ ">
47                 <option>Cava_001</option>
48                 <option>Cava_002</option>
49                 <option>Cava_003</option>
50             </select>
51         </div>
52     </div>
53
54     <div class="col-md-8">
55         <div id="map"></div>
56     </div>
57
58     </div>
59
60     <script src="leaflet.js"></script>
61     <script src="mqtt.min.js"></script>
62     <script type="module" src="mapa.js"></script>
63 </body>
64
65 </html>

```

CÓDIGO 3.12. Implementación en javascript de la parte dinámica de la página web.

```

1 let map;
2 let marker;
3 let path = [];
4 let polyline
5 const home = { lat: 10.918982886682658, lng: -74.87194240611939
6   ↵ };
7 let current_cava_position;
8
9 const options = {

```

```

10    clean: true, // retain session
11    connectTimeout: 4000, // Timeout period
12    clientId: 'web_app_cava_position',
13    username: '*****',
14    password: '*****',
15    Keepalive: 60,
16  };
17
18 const connectUrl = '*****';
19 const client = mqtt.connect(connectUrl, options);
20 const topic_cava = 'proyectoLuis/cava/datos';
21 let cava_data;
22
23 function init_map(){
24   //map = L.map('cava_map').setView([home.lat, home.lng], 13);
25   map = L.map('map').setView([home.lat, home.lng], 13);
26   //OpenStreetMap
27   var tile_map = L.tileLayer('https://tile.openstreetmap.org/
28     ↪ {z}/{x}/{y}.png', {
29     maxZoom: 19,
30     attribution: '&copy; ↪ <a href="http://www.openstreetmap.
31       ↪ org/copyright">OpenStreetMap</a>'
32   });
33
34   tile_map.addTo(map)
35
36   marker = L.marker([home.lat, home.lng]).addTo(map);
37   marker.bindPopup("<b>Posicion</b>"+"<br>"+"lat:_"+home.lat.
38     ↪ toString()+
39       ↪ "<br>_lng:_"+home.lng.toString()+
40         ↪ "<br>_CD"
41   ).openPopup();
42 }
43
44 function get_cava_data(cava_data) {
45
46   let latitude = parseFloat(cava_data.lat);
47   let longitude = parseFloat(cava_data.long);
48
49   return { lat: latitude, lng: longitude };
50 }
51
52 function set_cava_position_map(position){
53
54   document.getElementById('lat').textContent = "Lat:_"+
55     ↪ position.lat.toString();
56   document.getElementById('lng').textContent = "Lng:_"+
57     ↪ position.lng.toString();
58
59   if (marker) {
60     map.removeLayer(marker);
61   }
62
63   marker = L.marker([position.lat, position.lng]).addTo(map);
64   var popup_info = "<b>Posicion</b>"+"<br>" +position.lat.
65     ↪ toString() +", _"+position.lng.toString();
66   marker.bindPopup(popup_info).openPopup();
67 }
```

```
61     path.push([position.lat, position.lng]);
62     /*
63      if (path.length > 1) {
64          map.removeLayer(rastro);
65      }
66      */
67
68     var rastro = L.polyline(path, {color: 'blue'}).addTo(map);
69
70     map.setView([position.lat, position.lng], 15);
71 }
72
73 init_map();
74
75
76 client.on('connect', function () {
77     console.log('Conectado')
78     // Subscribe to a topic
79     client.subscribe(topic_cava, { qos: 0 }, function (err) {
80         if (!err) {
81             console.log('Suscripto')
82         }
83     })
84 });
85
86 client.on('reconnect', (error) => {
87     console.log('reconectando:', error)
88 });
89
90 client.on('error', (error) => {
91     console.log('Conexion_fallida:', error)
92 });
93
94 client.on('message', (topic, message) => {
95
96     //console.log('mensaje recibido', topic,
97     //            ↪ current_cava_position)
98     current_cava_position = get_cava_data(JSON.parse(message.
99     ↪ toString()));
100    set_cava_position_map(current_cava_position);
101 });
102 }
```

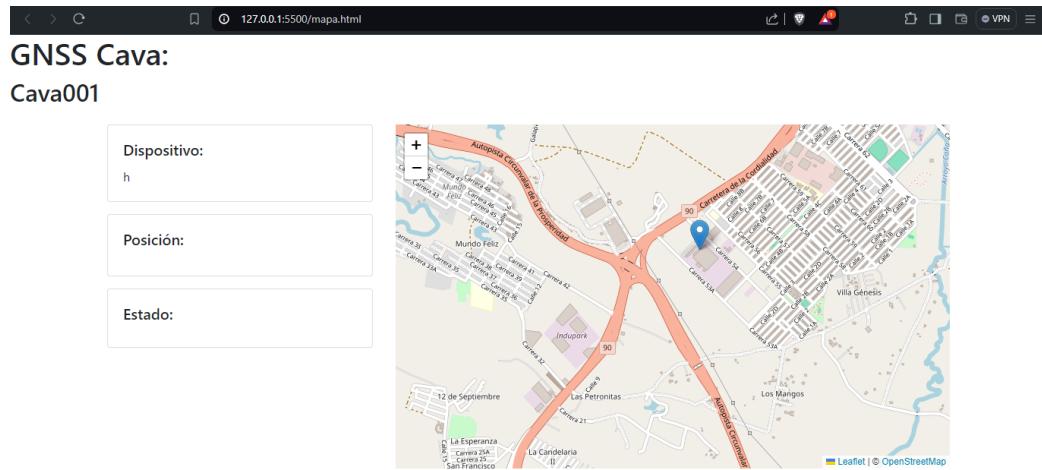


FIGURA 3.12. Maqueta de la interfaz web.

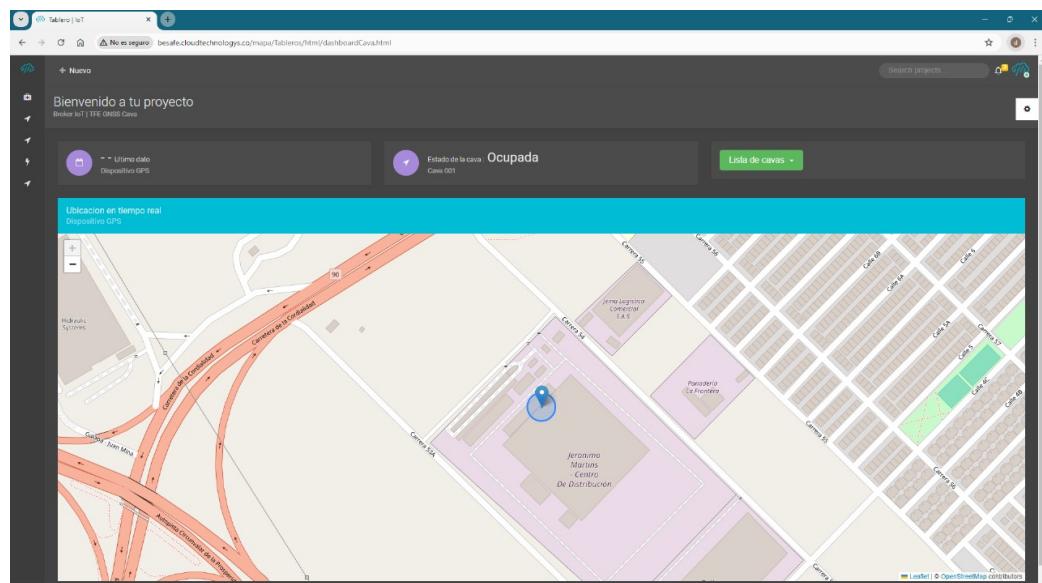


FIGURA 3.13. Maqueta de la interfaz web.

Capítulo 4

Ensayos y resultados

4.1. Pruebas funcionales del hardware

La idea de esta sección es explicar cómo se hicieron los ensayos, qué resultados se obtuvieron y analizarlos.

Capítulo 5

Conclusiones

5.1. Conclusiones generales

La idea de esta sección es resaltar cuáles son los principales aportes del trabajo realizado y cómo se podría continuar. Debe ser especialmente breve y concisa. Es buena idea usar un listado para enumerar los logros obtenidos.

Algunas preguntas que pueden servir para completar este capítulo:

- ¿Cuál es el grado de cumplimiento de los requerimientos?
- ¿Cuán fielmente se pudo seguir la planificación original (cronograma incluido)?
- ¿Se manifestó algunos de los riesgos identificados en la planificación? ¿Fue efectivo el plan de mitigación? ¿Se debió aplicar alguna otra acción no contemplada previamente?
- Si se debieron hacer modificaciones a lo planificado ¿Cuáles fueron las causas y los efectos?
- ¿Qué técnicas resultaron útiles para el desarrollo del proyecto y cuáles no tanto?

5.2. Próximos pasos

Acá se indica cómo se podría continuar el trabajo más adelante.

Apéndice A

Esquemáctico de la placa base del sistema

