


Smart FIng

Implementación y despliegue de un prototipo de campus inteligente en la Facultad de Ingeniería

Joaquín Veirana

Programa de Grado de Ingeniería en Computación
Facultad de Ingeniería
Universidad de la República

Montevideo – Uruguay
Febrero de 2023

Smart FIng

Implementación y despliegue de un prototipo de campus inteligente en la Facultad de Ingeniería

Joaquín Veirana

Tesis de Grado presentada al Programa de Grado de Ingeniería en Computación, Facultad de Ingeniería de la Universidad de la República, como parte de los requisitos necesarios para la obtención del título de Grado de Ingeniero en Computación.

Directores:
Prof. Dr. Ing. Eduardo Grampín
Prof. Dr. Ing. Matías Richart

Montevideo – Uruguay
Febrero de 2023

Veirana, Joaquín

Smart FIng / Joaquín Veirana. - Montevideo: Universidad de la República, Facultad de Ingeniería, 2023.

xv, 142 p.: il.; 29, 7cm.

Directores:

Eduardo Grampín

Matías Richart

Tesis de Grado – Universidad de la República, Programa de Ingeniería en Computación, 2023.

Referencias bibliográficas: p. 97 – 106.

1. Internet de las Cosas, 2. Campus Inteligente, 3. Sistemas ciberfisicos, 4. Sistemas embebidos. I. Grampín, Eduardo, Richart, Matías, . II. Universidad de la República, Proyecto de grado de Ingeniería en Computación. III. Título.

INTEGRANTES DEL TRIBUNAL DE DEFENSA DE TESIS

Prof. Dra. Ing. Laura González

Prof. Ing. Alejandro Blanco

Prof. MSc. Ing. Martín Giachino

Montevideo – Uruguay
Febrero de 2023

Agradecimientos

Quisiera agradecer a mis directores Eduardo y Matías, por darme la oportunidad de trabajar en este fascinante proyecto así como también guiarme de forma constante a lo largo del mismo. A Miguel, por su importante colaboración brindada en la parte electrónica y a Adrián y funcionarios de la biblioteca de la Facultad por su atenta cooperación.

Finalmente le quiero agradecer a mi familia, no hubiera sido posible alcanzar esta meta sin la ayuda y apoyo que siempre me dan.

RESUMEN

El presente documento muestra la memoria del trabajo realizado dentro del marco del proyecto de grado: *Smart FIng: Implementación y despliegue de un prototipo de campus inteligente en la Facultad de Ingeniería*.

Este es un proyecto centrado en las aplicaciones del Internet de las Cosas (IoT) y aborda un caso de estudio completo en el diseño e implementación de un primer prototipo de un campus inteligente a pequeña escala, pero funcional, localizado en la Facultad de Ingeniería. Al ser un proyecto de punta a punta, se tratan diversas temáticas que van desde la programación de los microcontroladores y fabricación de sensores hasta el despliegue del software encargado de recolectar y procesar en los servidores los datos generados por los sensores. Más precisamente se abarca el estudio y el desarrollo de las posibles infraestructuras a utilizar, las distintas arquitecturas disponibles para el software que oficia de plataforma IoT y la creación de los diversos sensores definidos como casos de uso, principalmente volcados a medir variables del ambiente y recolectar distintos datos. Se pretende dejar una fundación sólida y escalable sobre la que se pueda continuar trabajando en siguientes iteraciones. Sumado a lo anteriormente nombrado, se busca también la generación de conocimiento en las distintas tecnologías y softwares utilizados durante el proyecto, mostrando sus características, documentando los procedimientos realizados y presentando los resultados obtenidos.

Palabras claves:
Internet de las Cosas, Campus Inteligente, Sistemas ciberfisicos, Sistemas embebidos.

Lista de figuras

Lista de tablas

Tabla de contenidos

Capítulo 1

Introducción

1.1. Contexto y motivación

Durante la pandemia de COVID-19, la Facultad de Ingeniería impulsó el desarrollo de numerosas iniciativas para colaborar, combatir y adaptarse a la situación generada por el virus (Facultad de Ingeniería, 2020, [36]). Una de estas iniciativas fue desarrollada en el marco del curso *Taller de Sistemas Ciber Físicos* del Instituto de Computación, abordando el desarrollo de un sensor de medición de la calidad del aire utilizando sistemas embebidos económicos y de fácil acceso (Veirana, 2021, [99]). El objetivo del proyecto citado fue utilizar dichos dispositivos para realizar mediciones en distintos salones de clase de la facultad y así poder estudiar tanto la calidad del aire en ambientes con distintos niveles de ventilación durante el transcurso de las clases así como las capacidades de conectividad inalámbrica que estos dispositivos de IoT tienen en distintos sectores del edificio de la Facultad.

Sabiendo que el Internet de las Cosas cobra cada vez más relevancia tanto en el mundo académico como en el de distintas industrias y frente al satisfactorio resultado del proyecto anteriormente mencionado, se presentó de forma natural este nuevo desafío en el que se busca ahondar en la materia por medio de la investigación, la generación de conocimiento y el desarrollo de un prototipo funcional. Sobre este último se trabajó teniendo como objetivo que sea lo más sólido, robusto y cercano a un ambiente de producción posible, dejando así una base firme y el camino allanado para en el futuro poder construir un ecosistema completo en la Facultad donde se tenga una gran cantidad de sensores intercomunicados entre sí a la vez que generan y envían datos que puedan ser usados para estudiar el entorno así como influir sobre él.

Este ecosistema IoT a nivel de Facultad, al cual de ahora en más llamaremos *Campus Inteligente* o *Smart Campus* tiene múltiples motivaciones detrás, la principal es el poder explotar las posibilidades que estas tecnologías nos brindan y aplicarlas con el fin de ayudar a los estudiantes, docentes y funcionarios a mejorar su experiencia dentro de la Facultad. Sumado a esto, este campus busca ser escalable y modular para permitir a estudiantes y docentes, por un lado crear y agregar nuevos dispositivos al mismo que se integren y den solución a problemáticas que puedan surgir en el futuro, como también permitirles acceder a los datos que el campus genera para poder desarrollar nuevas aplicaciones e iniciativas a partir de estos.

Con respecto a la organización de este documento, al ser este un proyecto que busca ir de punta a punta, o dicho de otra forma que abarca aspectos en todo el espectro del campus, en ocasiones se tratan disciplinas de la computación que pueden ser un tanto distantes entre sí. Es por esto que se decidió alterar ligeramente el formato usual en este tipo de documentos y fragmentar el marco teórico en partes pequeñas que se hallarán previas a cada sección del documento donde se vayan a tratar temas distintos a los nombrados en el resto de capítulos. Por otra parte, dentro de los apéndices al final de este documento se brindan los procedimientos realizados con el fin de dejarlos documentados y explicados para todo aquel que desee en el futuro trabajar en este proyecto.

Por último resta destacar que para la realización de este proyecto se utilizaron múltiples recursos tanto de hardware y software como bibliográficos que serán citados a los largo de los capítulos, entre los bibliográficos hallamos particularmente una serie de trabajos previos realizados por estudiantes de la Facultad de Ingeniería que fueron de mucha utilidad para en algunos casos tomarlos de referencia y en otros para implementar directamente sobre el proyecto.

1.2. Objetivos y Etapas de trabajo

A partir de las motivaciones listadas en los últimos tres párrafos de la sección anterior donde se explica el porqué de un campus inteligente, es que podemos definir tres objetivos claros que engloban estas motivaciones de forma concreta. Sumado a esto y como veremos a continuación, estos objetivos guiarán de forma directa las etapas de trabajo del proyecto.

Listando los objetivos del proyecto:

- Diseñar y desplegar una plataforma propia de la Facultad que permita a estu-

diantes y docentes agregar nuevos dispositivos IoT.

- Desarrollar y desplegar la mayor cantidad posible de sensores, aplicados a distintas problemáticas dentro de la Facultad.
- Generar conocimiento y documentación de los procedimientos realizados durante el proyecto.

Para cumplir con dichos objetivos, el proyecto se conformó por dos etapas principales y una etapa final. La primera etapa principal se destinó a trabajar en todos los aspectos relacionados a la plataforma IoT encargada de recibir, procesar y almacenar los datos generados por los sensores. Se comenzó trabajando sobre la investigación y comparación de las plataformas IoT a utilizar, para que una vez fuera tomada esa decisión poder proseguir con la evaluación de las posibles infraestructuras sobre las cuales desplegar dicha plataforma junto con las arquitecturas de software posibles para cada una de ellas. Finalmente se realizaron y documentaron las instalaciones probadas para la plataforma y guías de uso de la misma.

En la segunda etapa se trabajó sobre la creación de sensores, en donde se realizó una investigación para la generación de ideas de posibles tipos de sensores y sus casos de uso. De este proceso se obtuvieron seis ideas de posibles sensores, de los cuales se tomaron cuatro para prototipar y probar, posteriormente en dos de estos cuatro prototipos se produjeron sensores físicos preparados para ser desplegados en la Facultad. Luego de producidos, se instalaron algunos de estos sensores en la Facultad y se recopilaron los datos generados con los mismos para analizar los desempeños de los sensores y para depurar el software.

Finalmente la última etapa fue dedicada a finalizar la documentación del proyecto y realizar las últimas pruebas de mediciones restantes.

1.3. Materiales entregados en el proyecto

El material correspondiente a este proyecto consiste del presente informe y del repositorio https://gitlab.fing.edu.uy/joaquin.veirana/proygrado-smartfing en donde se hallan todos los archivos, los proyectos con el código fuente y otros materiales adicionales relacionados con el proyecto. El repositorio tiene el material ordenado en diferentes directorios según el capítulo en el que son tratados o referidos.

1.4. Marco teórico inicial

Como se explica en la introducción, se presentará el marco teórico necesario antes de cada sección con los conceptos que se consideran necesarios conocer. Sin embargo en esta primera instancia se explicarán las ideas más generales para crear una base con la que abordar el resto de capítulos.

1.4.1. Internet y sus capas

Antes de comenzar a pensar en el Internet de las Cosas y sus distintas tecnologías, se considera necesario poder definir de forma concisa qué es internet y brindar una explicación superficial de las distintas capas que conforman su arquitectura. Como se define en (Kurose y Ross, 2017, Pag. 2-4, [53]):

> «Internet es una red de computadoras que interconecta miles de millones de dispositivos informáticos a lo largo de todo el mundo. (...) En la jerga de Internet, todos estos dispositivos se denominan hosts o sistemas terminales. (...) Los sistemas terminales se conectan entre sí mediante una red de enlaces de comunicaciones y conmutadores de paquetes. (...) Los conmutadores de paquetes se suministran en muchas formas y modelos, pero los dos tipos más utilizados actualmente en Internet son los routers y los switches de la capa de enlace. Ambos tipos de conmutadores reenvían paquetes hacia sus destinos finales. (...) Los sistemas terminales acceden a Internet a través de los ISP (Internet Service Provider, Proveedor de servicios de Internet), incluyendo los ISP residenciales, como son las compañías telefónicas o de cable locales; los ISP corporativos; los ISP universitarios; los ISP que proporcionan acceso inalámbrico (WiFi) en aeropuertos, hoteles, cafeterías y otros lugares públicos; y los ISP de datos móviles, que proporcionan acceso móvil a nuestros teléfonos inteligentes y otros dispositivos. Cada ISP es en sí mismo una red de conmutadores de paquetes y enlaces de comunicaciones. (...) El objetivo de Internet no es otro que conectar los sistemas terminales entre sí, por lo que los ISP que proporcionan el acceso a los sistemas terminales también tienen que estar interconectados entre ellos. Estos ISP de nivel inferior se interconectan a través de ISP de nivel superior nacionales e internacionales (...)»

Como se deja en claro en la cita anterior, Internet es una red de dispositivos

informáticos (nombrados hosts) conectados por distintos medios físicos (cables de cobre, líneas de fibra óptica, ondas de radio) y por dispositivos especiales (dispositivos de conmutación de paquetes) para dirigir los mensajes que son enviados entre estos hosts. Los hosts obtienen acceso a esta gran red por medio de un ISP (o proveedor de servicios de Internet), los cuales a su vez están conectados entre ellos por otros ISP más grandes. Es por esto que se suele nombrar a la Internet como *la red de redes*.

Por otra parte, también es importante saber que para poder establecer esta comunicación entre los dispositivos conectados a la red se tienen que seguir un conjunto de reglas. Estas reglas son llamadas **protocolos** y definen cada aspecto de la comunicación, por ejemplo la estructura que debe tener el mensaje, la secuencia que estos deben seguir, entre otros tantos aspectos. Si bien hay una gran cantidad de protocolos definidos, nos limitaremos a mencionar los más importantes en cada capa.

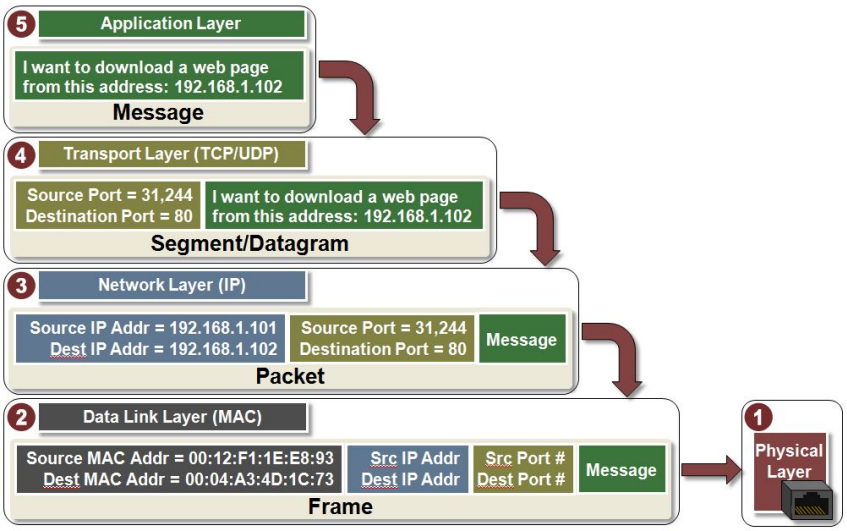

Figura 1.1: ***Esquema ilustrativo de las cinco capas de la red.*** Fuente: Microchip Technology Inc., 2021, [62]

En la parte que se nombran los conmutadores de paquetes, se hace referencia a los switch de capa de enlace, esto nos adelanta el hecho de que la red se puede comprender por capas para facilitar su comprensión. Respecto a esto se pueden definir cinco capas que pueden ser visualizadas una encima de la otra y en donde cada una de ellas brinda servicios a la inmediatamente superior. Las capas desde la inferior a la superior son: la capa física, capa de enlace, capa de red, capa de transporte y capa de aplicación. Como se puede observar en la Figura 1.1, cuando un mensaje pasa de una capa a otra es envuelto en una serie de datos extra que varían según el protocolo

por el cual está siendo manipulado. Estos datos extra son llamados cabezales del mensaje y contienen gran cantidad de datos, entre ellos usualmente se encuentra información de control y de direccionamiento del mensaje.

Pasando a dar un muy breve repaso de las capas y nuevamente basándonos en (Kurose y Ross, 2017, Pag. 42-44, [53]), definimos primero la capa superior de las cinco que es la **capa de aplicación**. En la capa de aplicación podemos encontrar aplicaciones o software que comúnmente utilizamos como por ejemplo el navegador, esta capa suele ser donde se crea inicialmente el mensaje cuando un host quiere enviar datos a otro. Como en toda capa de la red, se implementan protocolos que permitirán enviar estos mensajes, algunos de los más utilizados son HTTP para la transferencia de páginas web, SMTP para el envío y recepción de correos electrónicos o DNS para la traducción de los nombres de sitios web legibles para los humanos a su correspondiente número de dirección IP.

A la capa de aplicación le sigue la **capa de transporte**, esta capa se encarga de brindar dos protocolos en forma de servicio a la capa de aplicación para enviar sus mensajes entre dos hosts. Estos protocolos son UDP y TCP, mientras que UDP es un servicio básico y sin fiabilidad de envío de segmentos (así son llamados los mensajes en esta capa), TCP proporciona un servicio de envío que garantiza el suministro de los mensajes y un mecanismo de autorregulación de las velocidades según las capacidades del emisor y el receptor, así como para también adaptarse a redes congestionadas.

Luego está la **capa de red** encargada de enviar un segmento de capa de transporte al host de destino por la red a través de enrutadores utilizando el protocolo IP (con direcciones IP) y protocolos de enrutamiento, a los segmentos de capa de transporte se los encapsula en un nuevo paquete llamado datagrama.

Por último están la **capa de enlace** y la **capa física**. La capa de enlace se encarga de dirigir los datagramas de capa de red entre cada par de enrutadores y/o hosts presentes en camino conformado en la red, encapsulándolos en un nuevo paquete de capa de enlace llamado trama y usando una nueva dirección llamada MAC. Esta capa usa distintos protocolos según las características del enlace, Ethernet y WiFi son dos ejemplos de estos. Finalmente la capa física se encarga de mover los bits individuales de una trama desde un nodo de capa de enlace a otro, existen múltiples protocolos para esta capa según el medio de transmisión.

1.4.2. Internet de las Cosas

El Internet de las Cosas (Internet of Things o IoT en inglés) hace referencia a los sistemas conformados por objetos físicos conectados a la red, principalmente con algún tipo de conectividad inalámbrica y con un poder de cómputo suficiente como para medir, generar, enviar y recibir datos. Estos datos pueden ir y venir entre los objetos físicos y servidores de datos, la nube o incluso a otros dispositivos de la red IoT. El Internet de las Cosas busca conectar la mayor cantidad de dispositivos a la red con la característica de ser lo más independientemente posible de la intervención humana, permitiendo que los dispositivos observen el entorno que los rodea, recopilen datos y, de ser capaces de actuar, tomar decisiones autónomas en base a estos.

Relacionando las tecnologías del mundo IoT con las capas de internet vistas en la sección previa, encontramos dispositivos IoT que para su comunicación implementan protocolos de capa de aplicación como HTTP, MQTT o CoAP, otros que implementan protocolos en la capa de red y la de transporte como Zigbee y otros en la capa de enlace y red como LoRaWAN. También hay otros tipos de dispositivos como bridges y gateways que implementan distintos protocolos IoT según la tecnología para la que estén hechos y los conectan con otro protocolo de red. Un ejemplo de estos últimos son los gateways LoRa, que se encargan de recibir los datos enviados por los objetos IoT a través del protocolo LoRaWAN el cuál transmite la información por el aire usando las ondas de radio en el espectro definido por LoRA (siendo LoRa la capa física) y luego de recibir estos datos los transmite por medio de algún otro protocolo de Internet usando otra interfaz de red existente en el gateway, logrando así una integración y comunicación entre ambas tecnologías.

Si bien las posibles aplicaciones de estas tecnologías son muchísimas, en este proyecto se hará uso de un caso de uso típico en IoT que es el censado de distintas variables del ambiente y la potencial actuación en base al estado de este. Este caso de uso guarda ciertas similitudes con la domótica o los hogares inteligentes, en donde es posible lograr que el hogar funcione de acuerdo a los lineamientos que determine la persona que lo habita pero a la vez ser lo más auto gestionado posible con la ayuda de los datos que recolectan los sensores de la casa. El alumbrado o el termostato inteligente son algunos ejemplos de esto. Dentro de las aplicaciones del IoT ya popularmente adoptadas en la sociedad tenemos: los accesorios de ropa inteligentes como por ejemplo los smartwatch, las gafas inteligentes (aunque aún están en una fase temprana) o los hoy en día ya comunes smartphones. Estos

son dispositivos IoT que generan una vasta cantidad de datos diariamente, algunos complementos incluso miden nuestro ritmo cardíaco al hacer deporte o cuentan la cantidad de pasos en tiempo real, además pueden determinar la ubicación geográfica en todo momento incluso sin estar conectados a internet y junto con sistemas de mapas en la web son capaces de detectar cuando concurrimos a comercios o distintas localizaciones públicas. Existen otros campos de aplicación como los automóviles inteligentes con conducción autónoma, o los variados sensores usados para el monitoreo de cultivos en la agricultura (Friha et al., 2021, [41]), por nombrar algunos ejemplos de los tantos que se hallan en la actualidad.

Como toda tecnología es importante saber que tiene sus ventajas y desventajas, y es el desarrollador de la misma el responsable de conocerlas para poder trabajar sobre los puntos fuertes para potenciarlos como en los débiles para poder reducirlos. Sobre las ventajas podemos afirmar que estas tecnologías abarcan un amplio espectro de aplicaciones e impactan en todo tipo de ámbitos yendo desde el individual o personal, por ejemplo mejorando el confort de las personas (véase el caso de los dispositivos inteligentes o la domótica), hasta el ámbito empresarial e industrial permitiendo optimizar los recursos, mejorar procesos y aumentar la productividad. Otra ventaja fundamental que ya fue descrita anteriormente es la flexibilidad y la capacidad de adaptación de estas tecnologías que, al ser tan diversas, son capaces de ser aplicadas a prácticamente en muchas verticales de negocio en el mundo actual (Malik et al., 2021, [59]). Sumado a esto, el Internet de las Cosas es un paradigma de trabajo en pleno desarrollo que tiene una proyección de crecimiento notoria (Dahlqvist et al., 2019, [23]) y es un área de la tecnología en el que empresas e instituciones académicas están constantemente invirtiendo para su mejora. Finalmente también es de destacar que es una tecnología con tanta variedad de opciones en el mercado que es posible hallar una extensa cantidad de productos de bajo costo, los cuales son asequibles para estudiantes, entusiastas, pequeñas empresas y todo el que desee adentrarse en este mundo.

Pasando ahora a las desventajas, dentro de las consideradas las más críticas hallamos primero el riesgo de seguridad que este paradigma conlleva. Existiendo tantos dispositivos y fabricantes distintos, en conjunto con que no siempre estos productos están completamente probados en materia de seguridad y que por su naturaleza no se suelen hacer actualizaciones de firmware ni aplicar parches de seguridad, es cuestión de tiempo que se hallen formas de vulnerar una buena parte de ellos con fines maliciosos. Sumado a esto también hay una incertidumbre respecto a la privacidad de los usuarios ya que como se describe en párrafos anteriores, la

cantidad de datos generados a partir de las personas es enorme, incluyendo datos potencialmente sensibles por ejemplo relacionados a la salud. Yendo por un lado puramente técnico también se encuentran otras problemáticas. Entre las nombradas en (Farhan et al., 2018, [37]) encontramos la dificultad de trabajar con volúmenes tan masivos de datos cuando son generados de forma constante en un despliegue con una gran cantidad de dispositivos, junto con los problemas de heterogeneidad e integración que conlleva trabajar con tal variedad de dispositivos (ver la Figura 1.2 para visualizar un listado de algunas tecnologías de transmisión usadas en la actualidad clasificadas según su rango de acción), o por otro lado los problemas de escalabilidad que implica una cantidad tan masiva de dispositivos en las redes.

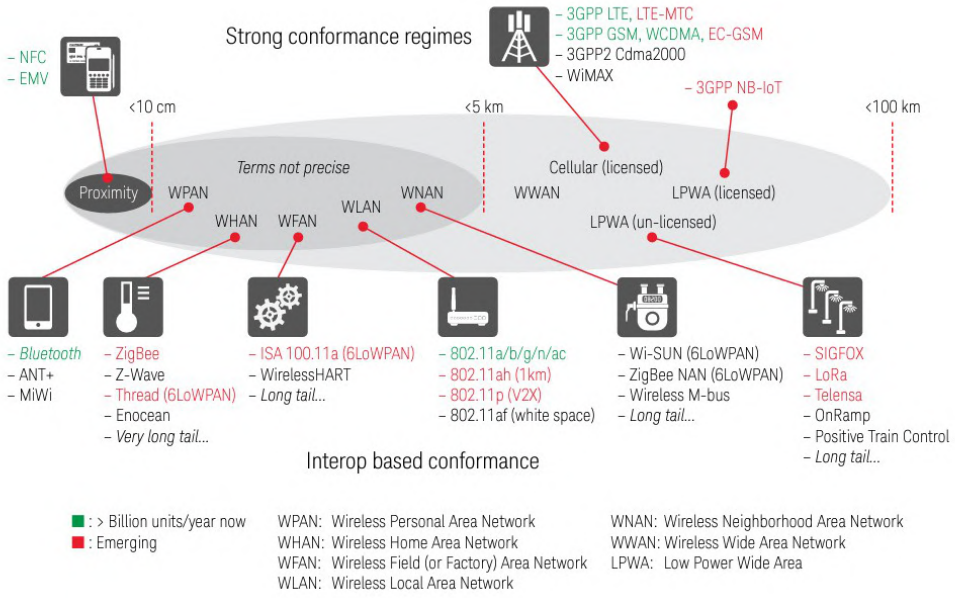

Figura 1.2: ***Esquema ilustrativo de las distintas tecnologías de transmisión inalámbrica y sus rangos operativos.*** Fuente: Keysight Technologies, 2017, [49]

1.4.3. Sistemas embebidos y sistemas ciber-físicos

Los sistemas embebidos son por definición sistemas computacionales que se encargan del manejo o control de otro sistema que lo comprende o abarca, una característica distintiva con un sistema computacional de propósito general es que los sistemas embebidos acostumbran ser diseñados para resolver unas pocas tareas específicas. En el Internet de las Cosas se encuentran dentro de la mayoría de los dispositivos y se podría decir que son el componente que controla el funcionamiento dentro de cada objeto o nodo de la red.

Los sistemas ciber-físicos están relacionados a los sistemas embebidos pero agregan sobre estos un componente físico con el cual está estrechamente integrado. Este componente físico posee la capacidad de interactuar de forma directa con el ambiente que lo rodea y en ocasiones con los humanos.

Citando la definición provista en (Lee y Seshia, 2017, [55]) (traducida con Google Translate):

> « Un sistema ciberfísico (SCF) es una integración de computación con procesos físicos. Las computadoras y redes integradas monitorean y controlan los procesos físicos, generalmente con bucles de retroalimentación donde los procesos físicos afectan los cálculos y viceversa. Como desafío intelectual, SCF se trata de la intersección, no la unión, de lo físico y lo cibernético. No es suficiente comprender por separado los componentes físicos y los componentes computacionales. En cambio, debemos entender su interacción. »

Los SCF cobran cada vez mayor importancia en la industria así como en el ámbito académico, también la conjunción de estos con otras tecnologías como el machine learning pueden llegar a ser aún más potentes. Las aplicaciones de los SCF van desde la agricultura, las cadenas de producción, la salud, la logística, la arquitectura y más. En Montevideo se ven cada vez más iniciativas relacionadas principalmente a proyectos de smart city, otra de las aplicaciones abarcadas por los SCF, las cámaras y sensores para el flujo inteligente de transito [20][63] son ejemplos de estos. En el presente proyecto se aborda el desarrollo de un smart campus que también es otra de las aplicaciones de los SCF. Se puede decir que en términos generales, los trabajos que tratan entornos físicos para volverlos inteligentes, son abarcados por los SCF.

1.5. Proyectos relacionados

Si bien el estado del arte en proyectos de Internet de las Cosas a nivel global es inmenso e imposible de abarcar en su totalidad, dentro de la Facultad de Ingeniería es posible encontrar trabajos que se centran en distintos apartados de la disciplina gracias a la diversidad tecnológica y las numerosas capas de abstracción que el mundo del IoT posee. Entre estos apartados es posible encontrar proyectos que tratan la creación de sensores nuevos con los que medir distintas variables del ambiente: (Veirana, 2021, [99]) y (Detta, 2020, [25]), otros proyectos que se centran

en la investigación de plataformas de IoT y formas de utilizarlas: (Arriola García y Wynants Lombardini, 2018, [8]), (Alles Conde, 2022, [4]) y (Abellá, Machado y Susviela, 2021, [1]), otros orientados en la búsqueda de crear nuevas formas de aplicar las tecnologías de transmisión inalámbrica presentes en los dispositivos IoT para obtener un resultado nuevo: (Crizul y Gómez, 2021, [22]), (Bracco, Grunwald y Navcevich, 2019, [14]) y (Acevedo, Coduri y Perera, 2018, [2]), luego otros que se dirigen al desarrollo en general de aplicaciones directas del Internet de las Cosas: (Osimani y Stecanella, 2018, [66]) y (Pacheco, 2020, [67]), y finalmente hay también proyectos centrados en la ciberseguridad en el mundo del IoT, una problemática central del paradigma en el presente: (Passaro y Pacheco, 2020, [68]) y (Márquez y Rodríguez, 2020, [60]).

En lo que respecta específicamente a campus inteligente se puede ver que están surgiendo nuevas iniciativas como (Escolar Díaz, 2021, [32]) en la Universidad de Castilla La Mancha en donde se busca transformar las instalaciones de la universidad brindando en su primera fase sensores de parking, de flujo de personas, calidad de aire y meteorología.

Como complemento se brinda la referencia a los proyectos de Internet de las Cosas del MIT media lab research[61], una institución con desarrollos punteros de aplicaciones del IoT, desarrollo de dispositivos sensores y proyectos de smart cities.

Capítulo 2

Implementación y despliegue de la plataforma

2.1. Introducción

En este capítulo se presenta lo llevado a cabo en la primera etapa del proyecto en donde se abarca el estudio de la plataforma IoT seleccionada y la infraestructura sobre la cual dicha plataforma debe ser instalada. La plataforma utilizada en este proyecto es ThingsBoard[1] [92], un software open source con buenas prestaciones y características de las cuales se brindará un análisis en siguientes secciones.

Seguido de esto, se presentarán las opciones de infraestructura evaluadas en base a un análisis que contempla múltiples variables, y en donde se termina definiendo: por un lado cual se considera que es la mejor opción para una implementación completa en miras de un proyecto a largo plazo y por otro la opción alternativa que se utilizó durante el proyecto para implementar el prototipo funcional.

Como complemento a este capítulo, en el Apéndice 1 se provee material extra con manuales de despliegue de las infraestructuras así como manuales de instalación de la plataforma IoT para las distintas variantes que se presentarán de la misma. Además se encuentran también algunas guías útiles para la gestión de la plataforma ThingsBoard.

[1] En su versión 3.4.0-1

2.2. Marco Teórico

2.2.1. Plataforma de Internet de las Cosas

Una plataforma IoT es un software especializado dedicado a la gestión y orquestación de una red de objetos de Internet de las Cosas. Generalmente brinda a los usuarios funcionalidades como: la administración de los dispositivos, implementación de aspectos de seguridad como la autenticación y autorización, disponibilización a los usuarios de la información generada por medio de web services, implementación de distintos protocolos de comunicación usualmente usados en IoT, procesamiento de los datos recibidos mediante motores de reglas, proveer una interfaz de usuario para administradores e incluso en algunos casos para clientes, entre otros. Se pueden encontrar en distintas arquitecturas, con distintas características funcionales y no funcionales, y en formatos de comercialización de software libre, de software propietario o en plataformas de nube pública (Arriola García y Wynants Lombardini, 2018, [8]).

2.2.2. Proveedores de servicios en la nube

Comúnmente escuchamos el término *La Nube* como una forma genérica y simplificada de denominar lo que en el fondo es una red de servidores y datacenters conectados a internet a escala mundial. Muchas empresas de tecnología ofrecen servicios de su propia nube, proveyendo software y hardware alojado en sus centros de cómputo disponibles para sus clientes por medio de internet. Usualmente hay distintas formas de brindar estos servicios, por un lado está la clase más común llamada nube pública, pero también se existen servicios de nube privada. En la nube pública toda la infraestructura (hardware y software) es propiedad del proveedor y es ofrecida a cualquier individuo u organización que desee contratar dichos servicios. Sumado a esto, todos estos recursos que conforman la nube son compartidos para todos los clientes que la utilizan, algunos ejemplos de plataformas con gran popularidad que proveen servicios de nube pública son Google Cloud[21], Microsoft Azure[12], IBM Cloud[47], AWS[78] o DigitalOcean[27].

Por otro lado la nube privada se basa en el mismo concepto de nube pero dedicado exclusivamente a una única entidad u organización, siendo accedida por una red privada. En este caso todos los recursos de esta nube son privados y usados únicamente por el dueño, ya sea que dichos recursos estén físicamente bajo propiedad de la entidad o bien que estos sean dados por un proveedor externo. Una gran par-

te de las empresas proveedoras de nube pública también brindan servicios de nube privada.

2.2.3. Software utilizado

2.2.3.1. Contenedores

Un contenedor en el mundo de la computación es un paquete de software que agrupa el código fuente, las bibliotecas necesarias, las dependencias de ejecución y los archivos de configuración. Debido a sus características de virtualización, posibilitan su ejecución en cualquier ambiente independientemente del hardware o del sistema operativo del host sobre el cual ejecutan. Además, son más ligeros y rápidos que una máquina virtual puesto que no requieren un sistema operativo completo para ejecutar. Existen múltiples softwares que permiten trabajar con contenedores, sin embargo el más popular en la actualidad es *Docker*[29]. Los contenedores serán una parte fundamental a la hora de instalar la plataforma IoT en una de sus posibles variantes tratadas más adelante.

2.2.3.2. Kubernetes

En esta sección se busca dar una breve explicación sobre Kubernetes, una herramienta que será de utilidad a la hora de abordar el segundo caso tratado en la Sección 2.4, donde se detallan las arquitecturas propuestas para la plataforma. Más precisamente, Kubernetes permitirá el despliegue de la plataforma IoT en su versión contenerizada y ayudará a alcanzar una serie de requerimientos no funcionales deseados entre los que se encuentran la alta disponibilidad, la escalabilidad y la performance. No se pretende ir a cada detalle de la tecnología sino brindar la información necesaria para comprender los componentes que conforman un cluster junto con las características distintivas del funcionamiento de cada uno.

Kubernetes es un software de código abierto desarrollado inicialmente por Google. Para comprender de forma clara el concepto de Kubernetes, lo mejor es comenzar con la definición brindada en *Google Cloud*[43]:

«“Kubernetes (a veces acortado a K8s; el 8 representa la cantidad de letras entre la “K” y la “s”) es un sistema de código abierto para implementar, escalar y administrar aplicaciones alojadas en contenedores en cualquier lugar.”»

Se complementan junto con Docker para brindar ayuda al desarrollador a trabajar sobre las tareas requeridas para poder crear y desplegar aplicaciones en conte-

nedores. Mientras que Docker ayuda en la generación de las imágenes a partir de las cuales se crean los contenedores de aplicaciones, Kubernetes utiliza estas imágenes para poder crear los despliegues y ayudar al desarrollador a gestionar estas aplicaciones. La interacción entre Docker y Kubernetes suele ser mediante un repositorio de imágenes, en donde Docker las almacena luego de crearlas y Kubernetes las descarga para su utilización. El desarrollador podrá interactuar con Kubernetes mediante comandos especiales ejecutados en un CLI cliente.

A continuación se brinda una breve descripción de la arquitectura y el funcionamiento de Kubernetes. Al instalar o contratar un servicio de Kubernetes se obtiene una agrupación de nodos de cómputo conocida como cluster, estos nodos pueden ser físicos o virtuales y están conformados por dos tipos componentes principales que abarcan a su vez otros subcomponentes dentro de ellos. El primer componente es el plano de control o nodo maestro (ver Figura 2.1), como su nombre indica es el encargado de controlar el cluster y tomar decisiones globales o llevar a cabo acciones sobre él en base a su estado. Dentro del plano de control se encuentran múltiples subcomponentes, cada uno con funciones específicas:

- **kube-apiserver:** Es un API server que expone las APIs que el desarrollador utiliza para interactuar con el cluster.
- **etcd:** Es un almacenamiento clave-valor que genera persistencia de toda la información del cluster (no de las aplicaciones que se ejecutan en él).
- **kube-scheduler:** Componente encargado de la asignación de pods que no están asignados a algún nodo worker. Los pods son una agrupación de contenedores en ejecución, es la estructura mínima de cómputo de Kubernetes y siempre se ejecutan dentro de estos nodos worker, en próximos párrafos se abordan ambos componentes con mayor detalle.
- **kube-controller-manager:** Es un proceso que ejecuta múltiples controladores de Kubernetes. Estos controllers se encargan de detectar y responder ante fallos en un nodo, controlan los números de réplicas que deben haber de pods, unifican los servicios que permiten el acceso a las aplicaciones a los pods que contienen estas aplicaciones, entre otros.

Para que un cluster no tenga punto de fallo, o dicho de otra forma que sea capaz de proporcionar alta disponibilidad, debe tener como mínimo tres nodos maestro replicados y preferentemente estando cada uno en distintos hosts. Como se nombraba anteriormente, estos nodos maestro se encargan de gestionar el cluster tomando decisiones sobre él. Cuando hay más de un nodo maestro estos trabajan en

conjunto y toman decisiones realizando un quórum en el cual se requiere que haya $floor(n/2) + 1$ maestros para poder funcionar, donde n es el número inicial de maestros. Es por esto que en un cluster de tres nodos maestro, se puede tolerar el fallo total de uno de ellos y el cluster seguiría funcionando con normalidad.

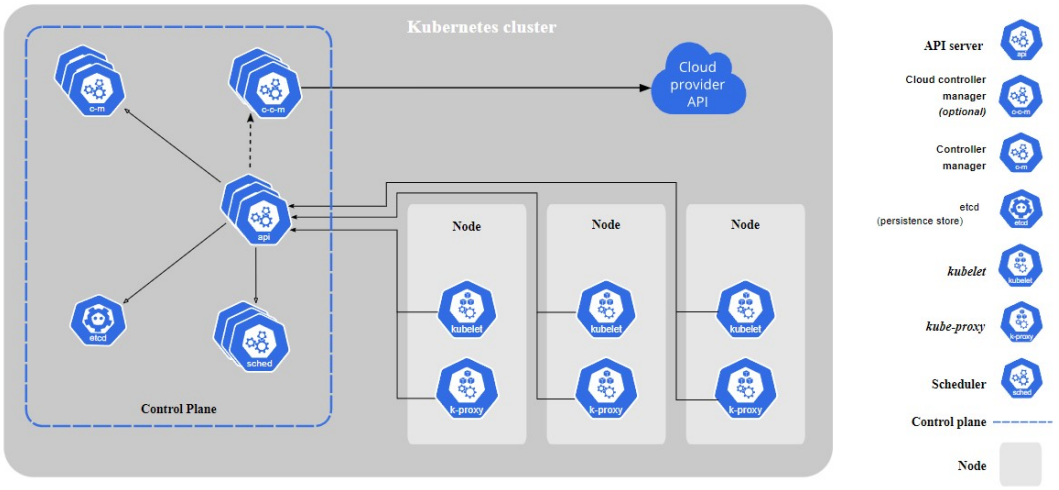

Figura 2.1: ***Esquema ilustrativo de la arquitectura de Kubernetes***. Fuente: Kubernetes Authors, 2022, [51]

El segundo componente que conforma un cluster es el nodo trabajador o worker node. El worker node es el componente donde se ejecutan los contenedores con las aplicaciones creadas por los desarrolladores, cada cluster debe poseer al menos un worker node y nuevamente en los clusters que proporcionan alta disponibilidad debe haber más de uno. Es de notar sin embargo, que los worker node pueden ser creados y borrados con relativa rapidez y más allá de generar una indisponibilización de los aplicativos, no involucran un potencial fallo en el cluster como si puede suceder cuando los nodos maestro fallan. Los contenedores se ejecutan dentro de unas estructuras llamadas pods, los pods son las unidades mínimas de cómputo en Kubernetes, estos engloban a uno o más contenedores en ejecución y tienen la particularidad de ser efímeros, esto significa que no poseen persistencia y pueden caerse o ser borrados en cualquier momento para automáticamente ser levantado uno nuevo e idéntico en su lugar. Volviendo a los worker node, dentro de ellos también se hallan otros subcomponentes, los cuales son:

- **kubelet:** Se asegura de que los contenedores estén ejecutando correctamente dentro de los pods del nodo.
- **kube-proxy:** Es un proxy de red que se ejecuta en cada nodo y se encarga de establecer reglas para lograr comunicación hacia los pods.

Para ver en mayor detalle cada uno de estos subcomponentes tanto del master como del worker, se recomienda referirse a la documentación oficial [50].

2.2.3.3. HAProxy

HAProxy es el software que será utilizado como API gateway para exponer los servicios provistos por la plataforma IoT, a continuación se cita la definición dada en su web oficial [46] (traducida con Google Translate).

> « HAProxy es un proxy reverso gratuito, muy rápido y fiable que ofrece alta disponibilidad, equilibrio de carga y proxy para aplicaciones basadas en TCP y HTTP. Es particularmente adecuado para sitios web de muy alto tráfico y alimenta una parte significativa de los más visitados del mundo. A lo largo de los años, se ha convertido en el equilibrador de carga de código abierto estándar de facto, ahora se envía con la mayoría de las distribuciones principales de Linux y, a menudo, se implementa de forma predeterminada en plataformas en la nube. (...) »

HAProxy posee una abundante cantidad de configuraciones posibles lo que lo hace un software versátil y adaptable, sumado a esto posee una extensa documentación oficial y al ser tan popular también tiene una gran comunidad. Todas las configuraciones que determinan el comportamiento de HAProxy se llevan a cabo en un único archivo de configuración el cual está organizado en cuatro secciones principales desde donde se define todo, estas secciones son: global, defaults, frontend, y backend. [45] En la Sección 1.3 del Apéndice 1 se muestra un poco más en detalle estas secciones junto con las configuraciones realizadas para los casos propuestos.

2.2.3.4. Keepalived

Keepalived es el software que será utilizado para alternar el tráfico entrante hacia los distintos hosts destinados al rol de API gateway en las implementaciones de la plataforma que proveen alta disponibilidad y tolerancia a fallos. A continuación se cita la definición dada en su web oficial [3] (traducida con Google Translate).

> « Keepalived es un software de enrutamiento escrito en C. El objetivo principal de este proyecto es proporcionar instalaciones simples y sólidas para el equilibrio de carga y alta disponibilidad para el sistema

> Linux y las infraestructuras basadas en Linux. El marco de equilibrio de carga se basa en el conocido y ampliamente utilizado módulo kernel de Linux Virtual Server (IPVS) que proporciona equilibrio de carga Layer4. (...) »

En el caso implementado, se utiliza Keepalived para garantizar la disponibilidad de los servicios mediante una IP virtual compartida por los hosts configurados como API gateways. El manejo de conexiones entrantes es redirigido al nodo de mayor jerarquía que se encuentre en línea, como se muestra en la Figura 2.2. De esta forma, las entidades externas a la solución manejan una única dirección IP para llegar a ella, abstrayéndose del estado de la misma.

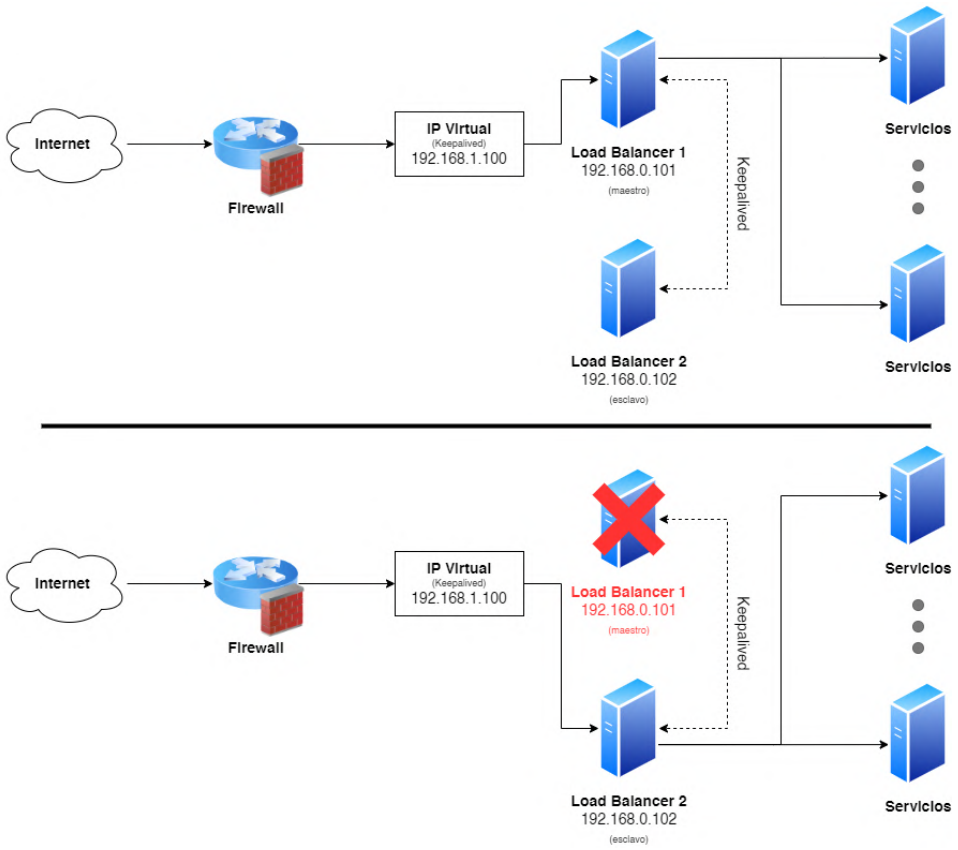

Figura 2.2: ***Esquema ilustrativo del funcionamiento de Keepalived respondiendo ante fallos en una API gateway.***

2.2.4. Arquitecturas de aplicaciones utilizadas

En esta sección se brinda una muy breve definición de dos estilos arquitectónicos para soluciones de software que son usadas por los casos de uso llevados a cabo

en el proyecto detallados en próximas secciones.

2.2.4.1. Arquitectura de microservicios

Los microservicios son un estilo arquitectónico que divide una aplicación en componentes donde cada uno es a su vez una aplicación completa pero en miniatura, encargada de producir una única tarea de negocio y acoplándose al principio de responsabilidad única. Cada microservicio tiene una interfaz y dependencias bien definidas, de forma que pueda correr y ser integrado en soluciones de manera independiente.

2.2.4.2. Arquitectura monolítica

En una arquitectura monolítica las distintas capas que conforman una aplicación se encuentran unidas en un mismo componente. La aplicación es independiente y capaz de realizar todas las tareas de negocio. En aplicaciones medianas o grandes puede resultar problemático su mantenimiento debido al crecimiento del código fuente y al hecho de no tener modularización de las distintas capas o componentes. Se puede entender como la arquitectura opuesta a los microservicios.

2.3. Plataforma IoT seleccionada

Esta sección se centra de lleno en la plataforma IoT seleccionada y todo lo relacionado a la misma.

ThingsBoard es un software de plataforma IoT open source desarrollado por la empresa ucraniana *ThingsBoard, Inc.*. ThingsBoard tiene una versión gratuita completamente funcional que permite administrar usuarios administradores, usuarios proveedores y usuarios finales, permite también crear y administrar dispositivos, consultar los datos por medio de APIs, crear reglas para procesar los mensajes, crear tableros con gráficos y widgets (Figura 2.3) y además provee soporte de distintos protocolos como HTTP, MQTT y CoAP, entre otras funcionalidades extra.

ThingsBoard posee además una edición profesional que tiene algunas funcionalidades extra, como por ejemplo la integración con otras plataformas IoT, la generación de reportes, descarga de los datos de telemetría en formato CSV (este punto fue resuelto manualmente, ver la Sección 1.5), un manejo más avanzado de roles de usuarios, white-labeling de la UI, entre otros.

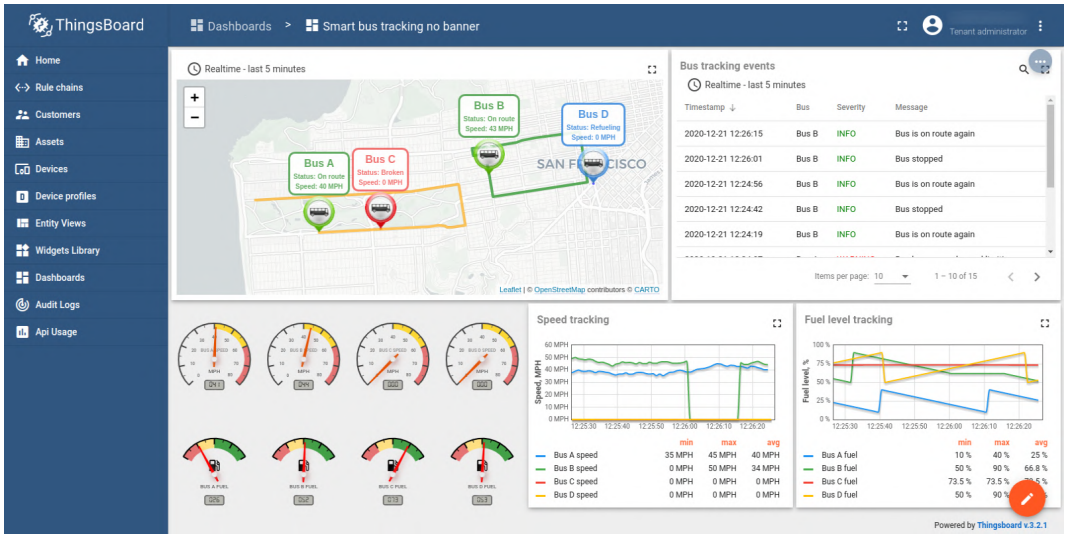

Figura 2.3: ***Captura de pantalla de un dashboard en ThingsBoard ejemplificando una aplicación de monitoreo de flota.*** Fuente: The ThingsBoard Authors, 2023, [92]

Entre las ventajas de ThingsBoard encontramos: su gran capacidad de adaptación a los distintos casos de uso, la escalabilidad que posee gracias a sus distintas arquitecturas permitiendo utilizar la que más se adecue al escenario, y también sus funcionalidades integradas de seguridad. Esto sumado a que tiene una interfaz sencilla y amigable, una comunidad relativamente grande y una documentación abundante y bien explicada, la hacen una excelente alternativa para ser usada en nuesto proyecto (Alles Conde, 2022, Pag. 25-27, [4]).

Una vez seleccionada la plataforma IoT a utilizar para el proyecto, es momento de evaluar las opciones de instalación disponibles, las cuales afortunadamente con ThingsBoard son numerosas. Pero antes de pasar a las posibles formas de instalación se presenta una rápida revisión de la arquitectura de ThingsBoard, arquitectura que, citando a sus creadores vemos que está diseñada para ser escalable, tolerante a fallos, robusta, eficiente, durable y customizable[93].

2.3.1. Arquitectura de componentes de ThingsBoard

A nivel de arquitectura la plataforma cuenta con los cinco componentes relevantes mostrados en la Figura 2.4: los *ThingsBoard Transports* que se encargan de proveer APIs basadas en protocolos HTTP, MQTT, CoAP y LwM2M para recibir los mensajes de los dispositivos. Una vez que reciben el mensaje, lo procesan realizando un parsing y lo envían a las *colas de mensajes* que proveen durabilidad en los mensajes en caso de fallos o caídas de los nodos. ThingsBoard permite que las colas

de mensajes sean implementadas con diferentes softwares de terceros, en particular para este proyecto se utiliza *Apache Kafka*.

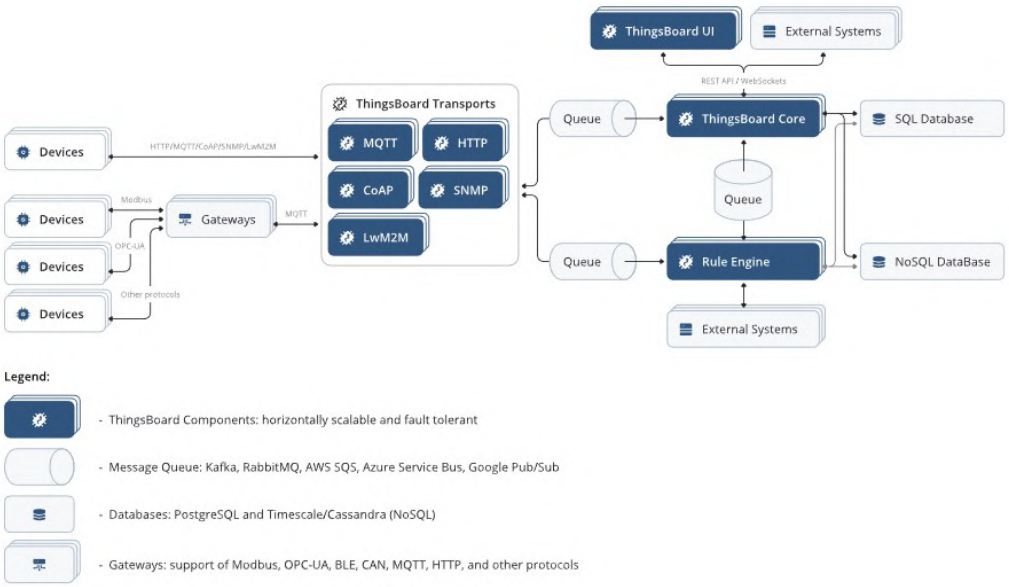

Figura 2.4: ***Esquema ilustrativo de la arquitectura de ThingsBoard.*** Fuente: The ThingsBoard Authors, 2023, [92]

Al ser los mensajes procesados en las colas de mensajes, son recibidos por el *motor de reglas*, otro componente fundamental de ThingsBoard. El motor de reglas se encarga de procesar los mensajes recibidos suscribiéndose a la cola de mensajes, es altamente configurable por medio de la interfaz gráfica usando un esquema similar a Pipes and Filters para definir los procesamientos y desencadenantes que se deben realizar al recibir cada tipo de mensaje. Finalmente al completar su pasaje por el motor de reglas, el mensaje es almacenado en la base de datos. Los otros dos componentes restantes son el *ThingsBoard Core* el cual se encarga de manejar los request REST enviados a la API de consulta de datos así como también los WebSockets que mantienen sesiones para obtener estos datos en tiempo real. Entre otras cosas también se encarga de monitorear los dispositivos y sus estados de conectividad. El componente final a nombrar es el aplicativo web *ThingsBoard UI*.

2.4. Arquitecturas de despliegue propuestas

Como fue explicado anteriormente, ThingsBoard brinda una serie de opciones de instalación que permiten a los usuarios adecuar la aplicación a sus necesidades.

En esta sección haremos un repaso de esas opciones para luego presentar los dos escenarios contemplados en el proyecto: un escenario creado para una posible implementación productiva del campus inteligente, llamada de ahora en más *arquitectura de cluster* (Subsección 2.4.3), en donde se busca alta disponibilidad, escalabilidad y robustez en la solución, y por otro lado, el escenario creado para un prototipo (Subsección 2.4.2) en donde se crea una alternativa reducida de la arquitectura de cluster que fuese viable para validar las otras partes del proyecto durante el transcurso del mismo. Esta última arquitectura será llamada de ahora en más *Arquitectura monolítica*.

Las opciones que ofrece ThingsBoard al momento de instalarse son:

- **Instalación on-premise o despliegue en la nube:** Refiere al sitio físico donde se encuentran los hosts sobre los que se instala la plataforma. En la siguiente sección se tratará este tema.
- **Instalación standalone o en modo de cluster:** La primera refiere a una instalación con un único nodo conteniendo el aplicativo y la segunda a un cluster de nodos replicados conteniendo el aplicativo. Respecto a este ítem fueron probadas ambas opciones en los casos de estudio a presentar más adelante.
- **Instalación con arquitectura monolítica o con microservicios:** ThingsBoard provee estos dos tipos de arquitectura que difieren en cómo se organiza la estructura interna de los componentes de software y cómo estos interactúan entre sí, en la Subsección 2.2.4 del marco teórico se brindan definiciones genéricas para comprender mejor cada una. A nivel de ThingsBoard las diferencias técnicas entre ambas se presentan en la documentación oficial[94][95].
- **Instalación con una base de datos SQL o híbrida SQL junto con NoSQL:** ThingsBoard ofrece para la base de datos la utilización de únicamente PosrgreSQL o PostgreSQL en conjunto con Apache Cassandra[40], en función de la cantidad de mensajes y el volumen de información generada por los dispositivos del sistema. En ambos casos de estudio presentados se detalla la decisión tomada en cuanto a este ítem.

2.4.1. ¿Instalación on-premise o en la nube?

Un aspecto que vale la pena estudiar más en detalle sobre la plataforma tiene relación con el lugar físico en donde se pretende instalar. Tiene mucho sentido pensar en este aspecto de la solución antes que otros puesto que en base a esta decisión se

impactará directamente sobre una serie de variables fundamentales de la solución que serán abordadas en este análisis, estas variables son: el costo, la soberanía de los datos, la robustez, la ciberseguridad y la dificultad del mantenimiento.

El *costo monetario de la infraestructura* sobre la que se instalará la plataforma es considerada la variable más preponderante a la hora de evaluar la viabilidad de una alternativa para ser implementada en el proyecto. En relación a esto, es importante notar que una ventaja fundamental al realizar una instalación on-premise en Facultad es que gracias a los recursos de computo disponibles, el despliegue de un prototipo para la solución podría ser considerado de costo cero. Sin embargo, para un despliegue productivo de la solución será necesario realizar un detallado análisis del costo operativo y de hardware que esta involucra. Por otra parte, durante el proyecto se hizo un relevamiento de los costos mínimos que implicaría tener en algunos de los proveedores de nube más conocidos: por un lado una única máquina virtual para una instalación monolítica de la plataforma y por otro un cluster de Kubernetes para una instalación clusterizada de la plataforma, a continuación se muestran los resultados obtenidos.

Tabla 2.1: Relevamiento de costos: máquinas virtuales en cloud (ene 2023)

Proveedor	Modelo	Cores	RAM	Storage	Precio mensual[a]
Azure	B2ms	2	8 GiB	16 GiB	US$ 61
Digital Ocean	GPD	2	8 GB	50 GB	US$ 68
AWS	m5d.large	2	8 GiB	75 GiB	US$ 75
Google Cloud	n1-standard-2	2	7.5 GiB	NA	US$ 49
IBM Cloud	B1.2x4	2	4 GB	25 GB	US$ 56

[a] El precio es el valor **redondeado** del costo de un uso total durante el mes, hay que considerar que el proveedor cobra por hora de uso.

Tabla 2.2: Relevamiento de costos: clusters Kubernetes de tres nodos en cloud (ene 2023)

Proveedor	Modelo	Cores	RAM	Precio mensual[a]
Azure	D2 v3	2	8 GB	US$ 256
Digital Ocean	GPN dcpu[b]	1	4 GiB	US$ 189
AWS	EKS + 3 EC2 (a1.large)	2	4GB	US$ 182
Google Cloud	n2-standard-2	2	8 GB	US$ 170
IBM Cloud	u3c.2x4 mz[c]	2	4 GB	US$ 261

[a] El precio es el valor **redondeado** del costo de un uso total durante el mes, hay que considerar que el proveedor cobra por hora de uso.

[b] GPN = General purpose node. dcpu = Dedicated CPU.

[c] mz = Multizona, cada nodo esta en un datacenter de una zona distinta.

En la Tabla 2.1 se pueden ver los precios mensuales estimados para una máquina virtual que provee lo necesario para resolver la propuesta de plataforma con instalación monolítica.

En la Tabla 2.2 se pueden ver los precios mensuales estimados para un cluster de Kubernetes de tres nodos que provee lo necesario para resolver la propuesta de plataforma con instalación de cluster, el tener tres nodos en el cluster es con el fin de proveer alta disponibilidad.

Es importante notar que en todos los casos vistos se está omitiendo el potencial costo extra de un servicio de base de datos en la nube, que puede llegar a ser importante para tener más robustez en la solución. En cualquier caso, si bien los servicios de nube son confiables, relativamente fáciles de usar y brindan una ventaja notoria en el hecho de no tener que administrar ni instalar prácticamente nada en materia de infraestructura[2], es claro que los costos pueden llegar a ser notorios según las cualidades que queramos darle a nuestra solución. Todo esto sumado al hecho de ser una propuesta pensada para el mediano y largo plazo, hace que estas alternativas de nube sean poco llamativas.

La segunda variable listada para la comparativa es la *soberanía de los datos* de la solución. La soberanía de datos busca plantear y dar respuesta a la siguiente lista de interrogantes: ¿dónde están guardados mis datos?, ¿quién tiene acceso a ellos?, ¿soy el único dueño de mis datos? o, ¿qué transparencia hay en el manejo de mis datos en caso de ser administrados por un tercero? Para poder evaluar este aspecto con lujo de detalle sería necesario una revisión detallada de todas las cláusulas legales de soberanía de datos presentes en los términos y condiciones de uso de las distintas plataformas de nube. Sumado a esto se debe hacer una revisión legal detallada de las leyes informáticas de protección de datos de los países o estados en los que se encuentran los datacenters del proveedor elegido, usualmente dichos datacenters pueden ser seleccionados por el usuario entre los disponibles según la región que más le convenga. Como esta revisión requiere de una gran cantidad de tiempo y conocimientos legales, queda por fuera del alcance de este proyecto. Sin embargo lo que sí se puede afirmar es que la alternativa on-premise siempre tendrá las de ganar en cuanto a soberanía de datos por razones obvias. Al ser estos datos alojados en los servidores de la Facultad, nunca serán expuestos (en circunstancias normales) a agentes externos a ella, esto hace que la alternativa on-premise obtenga un fuerte

[2]Este último aspecto se hace mucho más notorio en el caso de la plataforma con instalación de cluster ya que como se verá en secciones posteriores, la instalación y gestión de un cluster de Kubernetes implican un trabajo extra.

punto a favor.

Pasando ahora a la *robustez* de la plataforma, se puede decir que el análisis y conclusión que se obtiene vale de la misma forma para la siguiente variable en la lista que es la de *seguridad*. En este sentido se entiende que la alternativa de nube provee ventajas sustanciales dado que son aspectos que ya vienen prácticamente resueltos e integrados en la solución. Mientras que en la alternativa on-premise se tienen que resolver manualmente y, si bien es de reconocer que tener el control total sobre la infraestructura es un punto a favor ya que permite tomar todas las decisiones que se deseen respecto a la robustez y seguridad, para poder hacerlo de una forma correcta hay que invertir una cantidad sustancial de tiempo de investigación, obtención de conocimiento y trabajo. Dentro de algunos de los apartados considerados indispensables para la seguridad de la solución se encuentran: la correcta utilización de certificados TLS válidos y firmados por una CA avalada, la correcta configuración del firewall a nivel de sistema operativo en las máquinas virtuales (por ejemplo con ufw/iptables en Ubuntu) como también a nivel de las aplicaciones (esto en caso de usar Kubernetes utilizando el software Calico), la correcta configuración de las reglas de los enrutadores para exponer únicamente los hosts necesarios, el asegurar el correcto manejo de los permisos a nivel de los usuarios administradores de la infraestructura, la instalación y configuración de sistemas de detección de intrusos (IDPS), asegurar una comunicación cifrada incluso entre los componentes internos de la solución, la configuración de generaciones de backup de todos los sistemas regularmente y la instalación de sistemas de monitoreo/logueo de la actividad en cada nodo de la infraestructura.

Finalmente en cuanto a la dificultad de mantenimiento ambas opciones implican una cantidad de esfuerzo similar a nivel de aplicación, pero es a nivel de infraestructura que la situación cambia y la nube nuevamente tiene una ventaja notoria ya que es el proveedor quien se encarga de ello. Si bien ocurren las caídas en los sistemas de la nube, son raras de ver y usualmente son solventadas en períodos cortos de tiempo ya que disponen de un equipo de respuesta especializado para dichas situaciones. Por otro lado cabe recordar que los proveedores de nube también ofrecen a sus clientes algún tipo de servicio de soporte técnico por el cual se pueden crear tickets con consultas o incidencias para obtener ayuda del personal técnico del proveedor, un feature que on-premise obviamente no se tiene.

Haciendo entonces un repaso final de lo discutido anteriormente, podemos concluir que la opción deseada para este proyecto es una instalación on-premise. Principalmente por los beneficios de costo cero y del control total sobre las decisiones

de diseño e implementación, así como la soberanía sobre los datos. Pero cabe destacar que si bien se ahorra costos monetarios, es imperativo que se lleve a cabo la investigación correspondiente para poder resolver de manera sólida todos esos requerimientos que la nube resuelve por nosotros al contratar sus servicios y que son necesarios para un ambiente de producción expuesto a Internet, haciendo principal énfasis en la ciberseguridad y la robustez de la infraestructura.

En las dos siguientes secciones se presentarán los dos casos de estudio desarrollados para el proyecto.

2.4.2. Primer caso: Arquitectura del prototipo

El primer caso a tratar muestra el despliegue e instalación que fue llevado a cabo para el prototipo funcional de la plataforma en el proyecto. En este caso la instalación de ThingsBoard fue de tipo standalone y monolítica, llevada a cabo en una virtual private server en la nube de Antel llamada *Minube*[1]. El hecho de haberse realizado la instalación en la nube luego de haber dedicado una sección entera a discutir los motivos por los cuales era preferible la instalación on-premise podría sonar contradictorio. Sin embargo esto fue así ya que por un lado, esta máquina virtual en particular fue cedida por Antel a la Facultad de Ingeniería, por lo que no fueron consideradas las desventajas del costo. Por otro lado, como es una solución de la cual se desea acceder a sus datos desde Internet, la implementación on-premise habría tomado una cantidad de tiempo sustancialmente mayor respecto a la máquina virtual al ser necesario tomar en cuenta todas las medidas de seguridad que son mandatorias para un despliegue en la infraestructura de Facultad, algunas de estas medidas de seguridad fueron listadas en la misma Subsección 2.4.1.

Habiendo dicho eso, el motivo de haber llevado a cabo una instalación standalone y monolítica sabiendo que es la más básica de todas y que no ofrece alta disponibilidad ni tolerancia a fallos viene dado por el tipo de máquina virtual disponible en la nube de Antel. Esta máquina virtual está creada con una tecnología de virtualización llamada OpenVZ, que en particular no permite la instalación de software del gestor de paquetes Snap el cual es necesario para llevar a cabo la instalación de cluster usando Kubernetes. Aún de haberse podido llevar a cabo la instalación de Snap, el hecho de que sea un único host alojando todos los nodos replicados del cluster implica que a nivel de Kubernetes si se considere que el sistema tiene hay

[1] Las especificaciones del VPS de Minube consisten en un procesador con ocho núcleos Intel(R) Xeon(R) CPU E5-2660 v2 @ 2.20GHz y 16 GB de memoria RAM.

alta disponibilidad, pero en la realidad nunca llegaría a entrar en acción puesto que si el host se cae todos los nodos fallarían a la vez y se perdería el acceso al servicio.

Respecto a la seguridad, debido al mismo problema de instalación dado por OpenVZ no fue posible instalar el software *Certbot* de *Let's Encrypt* para poder obtener de forma sencilla certificados TLS firmados por dicha autoridad de certificación. Si bien hay alternativas provistas por Let's Encrypt que no requieren instalación, fueron descartadas del alcance del proyecto. Por lo que finalmente sí se dispone de un cifrado TLS para las conexiones HTTP pero autofirmado. Por motivos de alcance del prototipo tampoco se tiene un firewall configurado a nivel de sistema operativo, característica que como se nombra anteriormente es de vital importancia para un despliegue productivo.

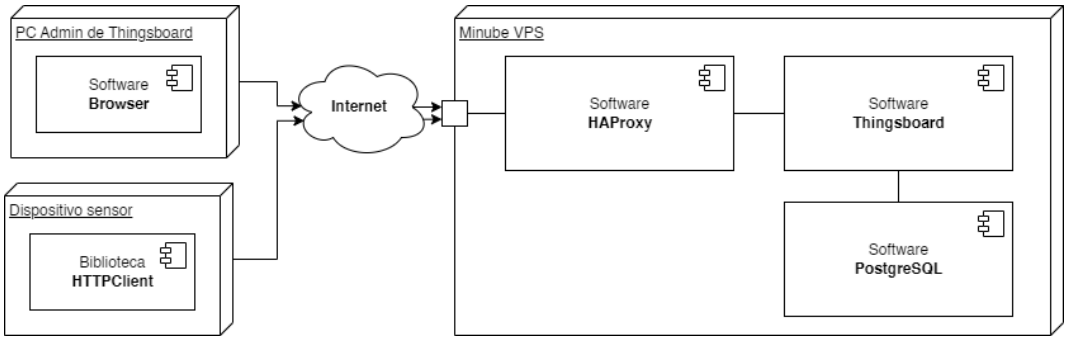

Figura 2.5: ***Esquema ilustrativo de la arquitectura monolítica presentada en el primer caso de estudio.***

Como se puede ver en el esquema de arquitectura de la solución (Figura 2.5) se muestra con UML las partes que la componen. Tanto el VPS y los dispositivos externos como las distintas piezas de software utilizadas son modeladas mediante bloques y componentes UML respectivamente. En dicho esquema se observa el componente llamado HAProxy el cual funciona como API Gateway para exponer los puertos 80 (HTTP) y 443 (HTTPS), para redireccionar los distintos requests a ThingsBoard y también para proporcionar el cifrado TLS en las conexiones por HTTPS.

Respecto a la base de datos, como se sabía de antemano que el volumen de los mensajes en el prototipo iba a ser reducido, se optó por una base de datos PostgreSQL única en lugar del modelo híbrido.

A pesar de ser una instalación monolítica los resultados obtenidos fueron muy satisfactorios, la aplicación web de ThingsBoard fue usada regularmente durante ráfagas esporádicas de varios días y no se detectaron caídas ni fallas en la misma, por otra parte los datos de los sensores fueron recibidos con normalidad y sin pro-

blemas. Las APIs de ThingsBoard fueron consumidas solicitando los datos de una semana entera de mediciones y los resultados fueron obtenidos también sin problemas. El buen funcionamiento de este prototipo de instalación permitió trabajar en las demás etapas del proyecto realizando distintas pruebas de manera correcta.

En la Sección 1.1 del Apéndice 1 se detalla el proceso de instalación de este prototipo.

2.4.3. Segundo caso: Arquitectura de cluster

Se presenta a continuación una arquitectura más avanzada que la que se llevó a cabo para el prototipo funcional del primer caso. En esta oportunidad se tiene como fin modelar una posible implementación de la plataforma en los servidores de Facultad, dicha plataforma sería instalada en su modalidad de cluster con el objetivo de lograr una solución con alta disponibilidad, robustez y que minimice los posibles puntos de falla.

Esta arquitectura hace uso de diez máquinas virtuales, cada una con distintos propósitos los cuales se irán abordando más adelante. Estas máquinas virtuales podrían estar en un mismo servidor físico o pueden estar en distintos equipos siempre y cuando estos se encuentren en la misma red interna. En la Figura 2.6 se muestra un esquema gráfico completo donde se observan las máquinas virtuales, nuevamente simbolizadas por bloques UML y dentro de ellas el software que ejecutan es modelado como componentes UML.

Para el diseño de esta solución se utilizó como referencia la arquitectura de red *screened host*, tomando como ejemplo la presentada en (Zwicky, Cooper y Chapman, 2000, Pag. 83, [103]). En esta arquitectura de red se utiliza uno o más hosts para exponer los servicios de la red interna a Internet y para redireccionar las solicitudes a los correspondientes hosts internos, estos hosts son llamados hosts bastion. Sumado a esto, por medio del filtrado de paquetes llevado a cabo por el firewall del enrutador se protege al resto de hosts en la red interna de ser alcanzados desde el exterior de forma directa por agentes externos. En materia de seguridad esta arquitectura es relativamente buena siempre y cuando se proteja de manera adecuada el host bastión, puesto que de ser vulnerado se pone en peligro a toda la red interna. Es de notar también que el otro punto de fallo posible es el enrutador, el cual en caso de este ser comprometido toda la red interna y sus servicios quedan inaccesibles.

En la Sección 1.2 del Apéndice 1 se detalla el proceso de instalación de todos

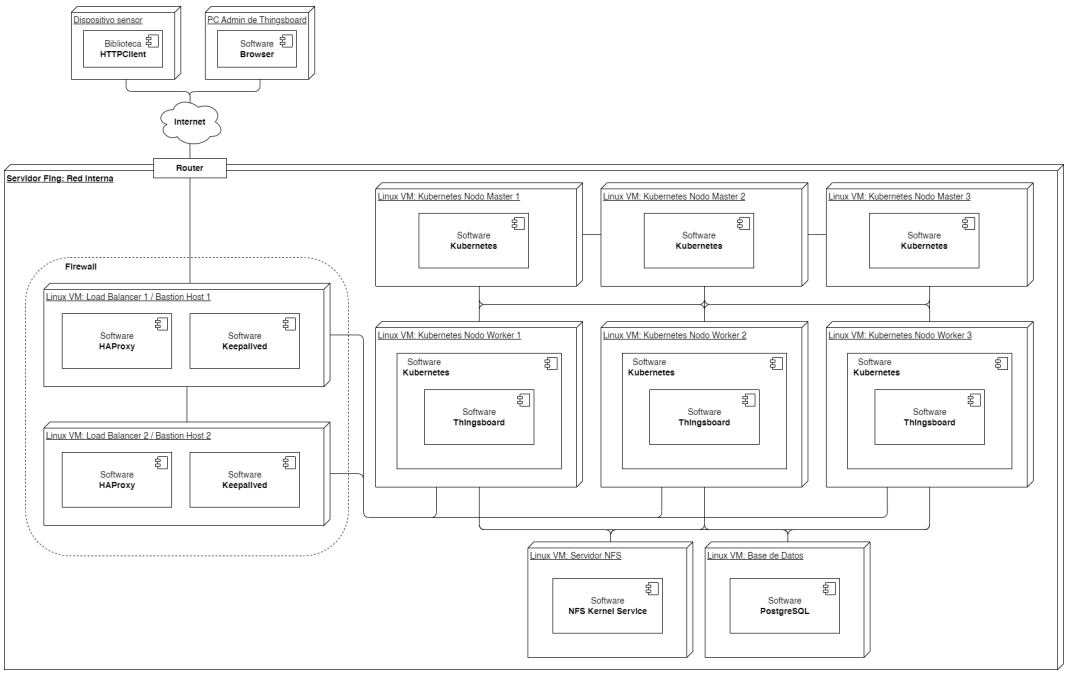

Figura 2.6: ***Esquema ilustrativo de la arquitectura de cluster presentada en el segundo caso de uso.***

los componentes.

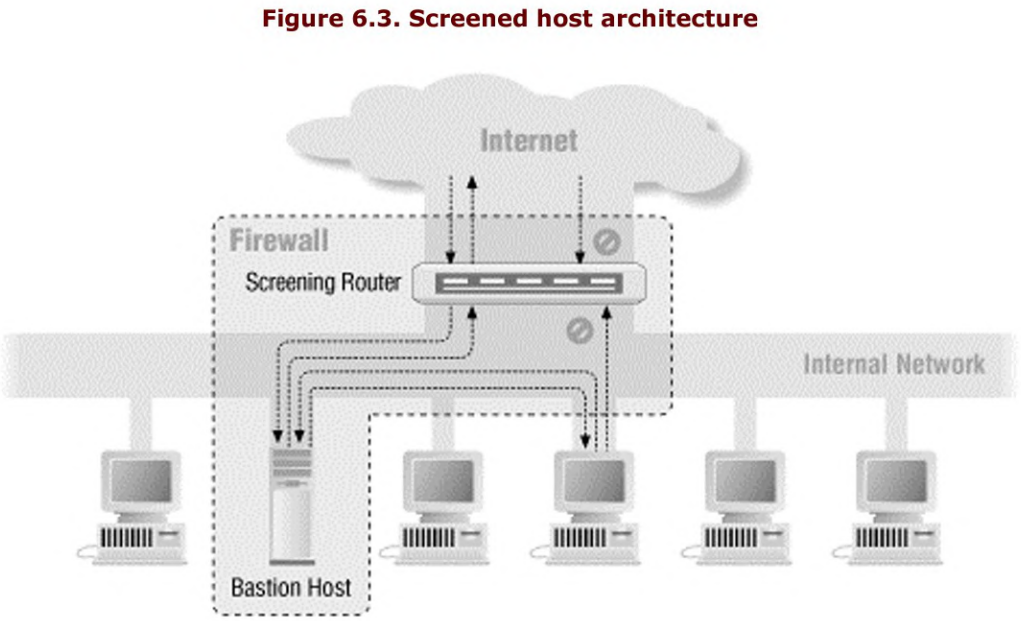

Figura 2.7: ***Esquema ilustrativo de la arquitectura de red "screened host".***
Fuente: Zwicky, Cooper y Chapman, 2000, [103]

2.4.3.1. Bastion host, load balancers y firewall

Retomando el esquema de la Figura 2.6, se procede a dividirlo en distintas secciones mostradas en la Figura 2.8 para su mejor comprensión. La primera sección a detallar es la sección verde en la que se encuentran los hosts que cumplen el rol de bastión, API gateway y balanceadores de carga. Los clientes se conectan a través de Internet a los servicios que se proveen en la red interna, estos servicios son tanto para los dispositivos sensores que envían datos de telemetría a la API REST de ThingsBoard, como para los administradores de la plataforma que acceden al dashboard web de la misma y también para las aplicaciones que consumen los datos almacenados por ThingsBoard por medio de su API REST. Al realizar dicha conexión, los hosts bastión que en este caso son dos máquinas virtuales se encargan de exponer por HTTPS una API REST por la que se redireccionará el tráfico hacia los hosts internos correspondientes. Para esto hace uso del software HAProxy con una configuración personalizada. Por otro lado se tiene Keepalived, software que como se explica en la Subsubsección 2.2.3.4, permite exponer los hosts (bastión) por medio de la implementación de una única IP virtual que puede ser tomada por cualquiera de las dos máquinas virtuales según el estado de disponibilidad de cada una y en caso de que una de ellas se vea comprometida, la IP virtual sea tomada automáticamente por la otra, proveyendo así alta disponibilidad del servicio.

Para lograr una solución segura también es necesario configurar el router que brinda acceso desde Internet a la red interna para ocultar a todos los hosts que no sean los bastiones. Si bien este es uno de los apartados más importantes en materia de seguridad, queda por fuera del alcance de este proyecto a nivel de implementación.

2.4.3.2. Cluster de Kubernetes

Los componentes en color celeste conforman el cluster de Kubernetes, como se puede apreciar se disponen de tres nodos maestros y tres nodos trabajadores que tal y como fue detallado en la sección teórica de la Subsubsección 2.2.3.2, es la cantidad mínima necesaria para tener tolerancia al fallo de un nodo maestro y dos nodos trabajadores. La instalación de dicho cluster fue realizado con la herramienta *Microk8s*[17], desarrollada por Canonical. Esta herramienta permite realizar una instalación de un cluster multi nodo de manera más sencilla que otras herramientas disponibles como por ejemplo *Kubeadm* y además posee distintos addons con funcionalidades de utilidad para el cluster[16].

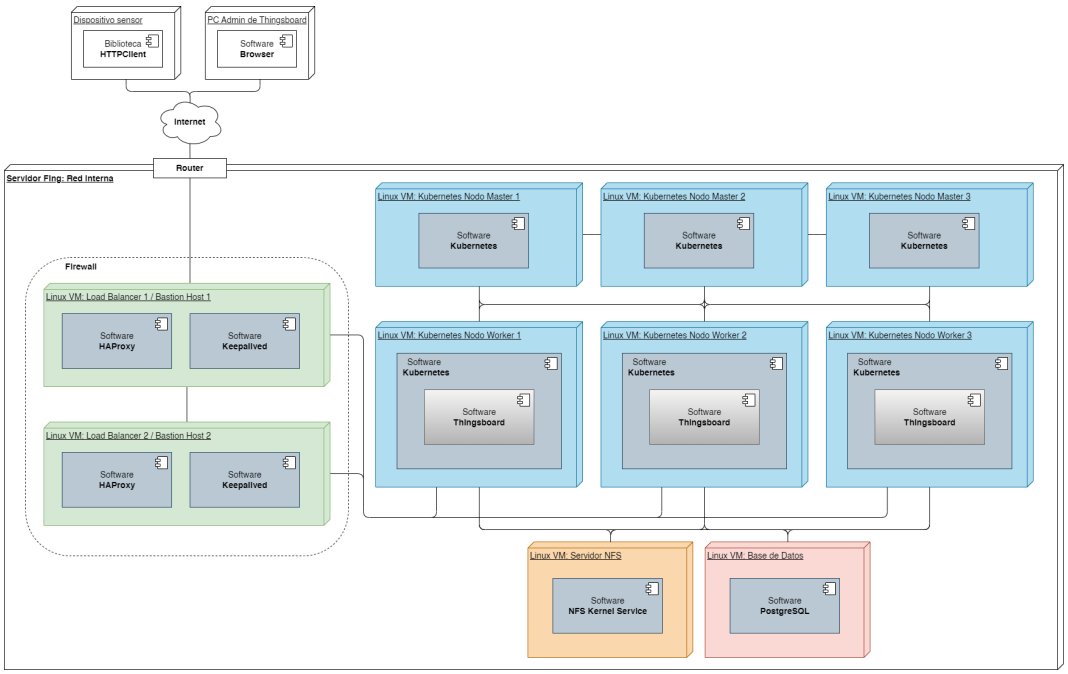

Figura 2.8: ***Esquema ilustrativo de la arquitectura de cluster presentada en el segundo caso de uso pintado por secciones.***

2.4.3.3. Servidor NFS

Como se describe nuevamente en el marco teórico de Kubernetes en la Subsubsección 2.2.3.2, las aplicaciones se despliegan en estructuras llamadas pods las cuales son efímeras, esto quiere decir que pueden eliminarse y recrearse en cualquier momento si el cluster lo decide y no poseen persistencia de datos. Es por esto que para ciertas aplicaciones como en el caso de ThingsBoard se necesita disponer de una conexión a un servicio que permita a los pods almacenar datos relacionados a la aplicación para que persistan luego de que estos sean recreados.

Para lograr la persistencia requerida se utiliza un servidor NFS, el cual está coloreado de anaranjado en la Figura 2.8. NFS (Network File System) es un protocolo de capa de aplicación para crear sistemas de archivos accesibles en una red local, este protocolo habilita a un host cliente acceder a una unidad de almacenamiento en un host servidor como si estuviera físicamente conectado a él. Este servidor debe ser instalado antes del despliegue de la aplicación ThingsBoard ya que al momento de realizar dicho despliegue se crean volúmenes dinámicos en el cluster para almacenar información de la aplicación, en caso de no tener conexión con el servidor NFS los pods comenzarán a fallar.

2.4.3.4. Base de datos

En el componente rojo encontramos la máquina virtual que aloja la base de datos que almacena la información generada por los dispositivos sensores. Al igual que en el primer caso presentado, la base de datos consiste en una instancia de PostgreSQL sin integración con Cassandra. Eventualmente es posible plantearse la opción de pasar a un modelo híbrido si es necesario. La migración puede llegar a ser costosa de llevar a cabo, una opción rudimentaria de llevarla a cabo consiste en la creación de la nueva instancia de base de datos híbrida paralelamente a la antigua e implementar un proceso que utilice la API REST de ThingsBoard para consultar toda la telemetría generada hasta el momento de cada dispositivo y la vaya insertando en la nueva base de datos. Debido al costo que podría suponer esta migración, es importante siempre conocer y estimar de antemano qué cantidad de dispositivos habrán y qué volumen de tráfico se espera de ellos. En nuestro caso se continúa con una base de datos SQL única ya que según la documentación oficial de ThingsBoard, el pasaje al modelo de base de datos híbrido es recomendado a partir de los cinco mil mensajes por segundo, lo cual es una cantidad extremadamente mayor a la esperada incluso en un despliegue productivo a gran escala dentro de la Facultad.

Otra práctica de seguridad considerada importante que no se abarca en este caso de uso es la creación de replicas para la base de datos o la creación de un proceso que genere respaldos regularmente de la misma con el fin de evitar tener un único punto de falla en el caso de la primera opción o para tener posibilidad de recuperación ante fallas en el caso de la segunda opción.

Capítulo 3

Diseño y desarrollo de los nodos sensores

3.1. Introducción

En este capítulo se presenta lo llevado a cabo en la segunda etapa del proyecto en donde se aborda el desarrollo de los distintos dispositivos sensores ideados para cuatro casos de uso propuestos. Se brinda una descripción de cada caso de uso planteado, junto con su motivación, el valor que aporta a la Facultad y al campus inteligente, las tecnologías utilizadas en su desarrollo y finalmente las pruebas junto con los resultados obtenidos a partir de ellas.

Para cada uno de los cuatro casos de uso que se plantearon para el campus inteligente se diseñó un dispositivo sensor para poder brindarles una solución. Más precisamente, dos de ellos fueron fabricados y finalizados completamente para ser desplegados en la Facultad, mientras que los otros dos quedaron en fase de prototipo, habiendo dejado un primer acercamiento de cómo estos podrían resolverse.

3.2. Marco teórico

3.2.1. Hardware utilizado

3.2.1.1. Plataforma ESP32

El chip ESP32 es una de las piezas fundamentales de este proyecto, siendo el hardware en el que se basan tres de los cuatro casos de uso que se presentarán en próximas secciones. Es por esto que se considera relevante dedicarle una sección

explicando sus características a fin de que el lector pueda comprender el hardware con el que se está trabajando así como también conocer el potencial que este tiene.

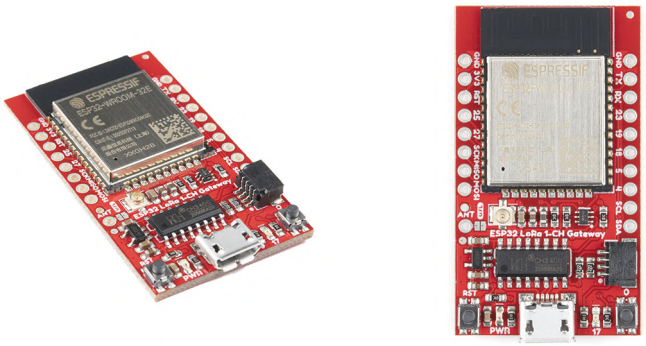

Figura 3.1: ***Fotografías de la placa de desarrollo ESP32 SparkFun LoRa Gateway 1-Channel usada en el proyecto.*** Fuente: sparkfun.com [84]

El ESP32 es una pieza de hardware denominada en inglés como *system on a chip*, o SOC. Los SOC son una familia de chips muy usados en los sistemas embebidos que tienen como principal característica el hecho de integrar todos sus módulos en un único chip físico. El procesador que utiliza es el *Tensilica Xtensa 32-bit LX6* que dispone de dos núcleos, el ESP32 es el sucesor del chip ESP8266 y en este sentido una de las características principales que mejora es el pasaje de un procesador mono núcleo a tener dos. Tanto el ESP32 como el ESP8266 son diseñados y fabricados por la empresa china *Espressif* [35].

El ESP32 cuenta con una amplio número de funcionalidades y características que lo vuelven extremadamente potente y versátil para ser usado en aplicaciones de Internet de las Cosas. Se listan a continuación algunas de las principales características que pueden observarse en la Figura 3.2.

- **Periféricos y sensores:** El chip dispone de 34 pines de entrada y salida de propósito general (GPIO) con distintas características. Algunas de las clases de pines usadas en este proyecto son:
  - **Pines digitales** que permiten la lectura y escritura de una señal digital booleana.
  - **Pines analog-to-digital** y **digital-to-analog** (ADC y DAC) que permiten leer y escribir señales analógicas.
  - **Pines I2C** que permiten la comunicación con periféricos por medio de este bus serial.

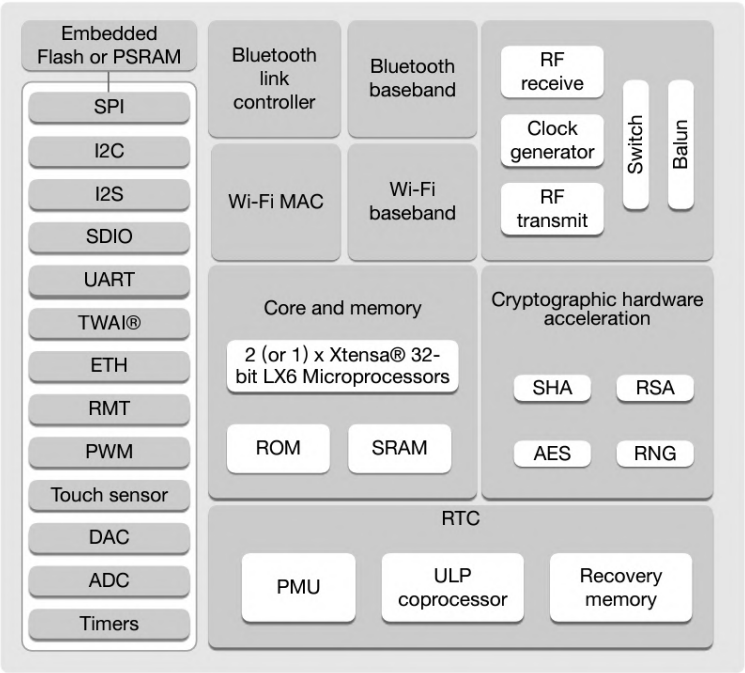

Figura 3.2: ***Diagrama de bloques funcionales que conforman el chip ESP32.***
Fuente: espressif.com [34]

Sumado a estos se dispone de otras clases de pines como PWM, I2S, TWAI, SPI, UART, GPIOs capacitivos, RMT, PCNT, entre otros [34] (Pag. 34-45).

- **Tecnologías y protocolos de comunicación:** El ESP32 dispone de un transceptor de 2.4GHz usado para dar soporte a WiFi e implementar los estándares de comunicación 802.11 b/g/n y Bluetooth Low Energy (BLE). Además de estos conocidos estándares de comunicación inalámbrica, se agrega a la lista *ESP-NOW*[30][33], un protocolo desarrollado por Espressif específicamente para esta familia de chips implementado por encima de la capa de enlace abarcando la capa de red, la capa de transporte y la capa de aplicación. Este protocolo permite la comunicación peer-to-peer unidireccional o bidireccional utilizando la antena WiFi pero sin necesidad de disponer de una red WiFi, lo que lo hace más eficiente en consumo energético. ESP-NOW brinda al ESP32 la capacidad enviar y recibir mensajes cortos de 250 bytes, con o sin encriptación (CCMP), a una tasa aproximada de 300 paquetes por segundo a través de distancias que pueden ir hasta los 300 metros en su modo estándar, o con una tasa aproximada de 95 paquetes por segundo a distancias que pueden ir hasta los 500 metros en su modo long range (LR)[9]. Aunque estos valores varíen según el entorno geográfico, el bitrate declarado por el fabricante es de

1 Mbps en su modo estándar.

- **Sistemas operativos en tiempo real:** Dentro de las bibliotecas presentes en el *Espressif IoT Development Framework* (ESP-IDF) para el ESP32, se encuentra integrado un port del sistema operativo en tiempo real FreeRTOS[6][31]. Este sistema operativo es rápido y ligero y se encarga de manejar los procesos administrando su ejecución de forma concurrente en función de su estado y sus prioridades, además controla la asignación de procesos a los distintos núcleos del ESP32.
- **Compatibilidad con frameworks de trabajo:** Gracias a su amplia popularidad, el ESP32 cuenta con distintos ambientes de desarrollo integrados (IDEs) para programar las aplicaciones, siendo los más conocidos el Arduino IDE y la extensión PlatformIO para VSCode. En (Veirana, 2021, Pag. 18-23, [99]) se halla una guía de configuración de PlatformIO de la forma en la que fue usada para este proyecto.
- **Bajo coste:** El costo de estos dispositivos es bajo en comparación a su poder y a las prestaciones que ofrecen en relación a otras piezas de hardware con las que compite. Si bien los precios pueden variar según el mercado y las características del modelo que se desee adquirir, al momento de realizar el proyecto las placas de desarrollo utilizadas se encontraban en USD 37.50 la unidad [84].
- **Bajo consumo:** El chip varía su consumo en relación al funcionamiento y la utilización de sus capacidades. Según el *ESP32 Series Datasheet* [34] (Pag. 32 y 48), el chip consume aproximadamente 240mA en su modo activo con la CPU funcionando y transmitiendo datos por WiFi, por otra parte consume 130mA en el mismo modo pero transmitiendo por Bluetooth. Además de esto el chip tiene otros modos de trabajo reducido en donde se pueden desactivar diferentes módulos y utilizar el co procesador de bajo poder (ULP)[1] que tiene integrado para reducir su consumo, algo que resulta de utilidad en proyectos que utilicen baterías. Estos modos de consumo reducido van desde los 68mA a los 5μA.

Como mención extra a todo lo anterior, es de destacar que existe software para los chips ESP32 provisto por el fabricante que dan soporte a *Matter*, el protocolo que busca crear un nuevo estándar de comunicación sobre el protocolo IP para los

[1] Este co procesador de bajo consumo es programado usando un set de instrucciones de largo fijo de 32 bits[24], similar al lenguaje Assembler.

dispositivos de domótica de distintos fabricantes. Este proyecto está llevado a cabo por la *Connectivity Standards Alliance* conformada por numerosas compañías de tecnología entre las cuales se encuentra Espressif. Si bien Matter aún se encuentra en una fase temprana, podría ser a futuro el nuevo estándar para tecnologías de domótica y quizás también de otras áreas del IoT.

3.2.1.2. Raspberry Pi

La Raspberry Pi es otro componente de hardware utilizado para uno de los casos de uso, consiste de una computadora monoplaca de bajo costo creada por la Raspberry Pi Foundation. Posee todas las características de un PC regular y el sistema operativo comúnmente utilizado es Linux Raspberry Pi OS, una distribución derivada de Debian. Sin embargo existen otras ditribuciones de propósito general así como una gran variedad de distribuciones para propósitos específicos las cuales son masivamente utilizadas, entre estas últimas encontramos por ejemplo la cartelería digital, la emulación de sistemas antiguos y videojuegos retro, las máquinas CNC y de impresión 3D, aplicaciones de home automation, NAS, media centers y más. Existen diferentes versiones, siendo la Raspberry Pi 4B el modelo más reciente lanzado en 2019.

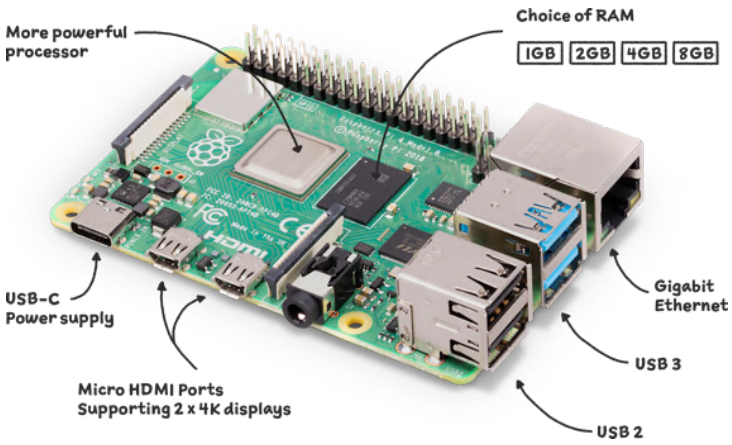

Figura 3.3: ***Fotografía de la Raspberry Pi modelo 4.*** Fuente: raspberrypi.com [73]

3.3. Primer caso de uso: Sensores de ambiente y calidad de aire

3.3.1. Introducción

Durante la pandemia de COVID-19, se llevó a cabo un proyecto realizado en la materia Taller de sistemas ciber-físicos (Veirana, 2021, [99]) para conocer el potencial que pueden llegar a tener este tipo de dispositivos al llevar a cabo una medición constante del aire. Habiendo obtenido resultados positivos, se propone este caso de uso que consiste en el diseño, programación y fabricación de nueve sensores de ambiente y calidad de aire para ser desplegados en distintos salones del piso 3 de la Facultad de Ingeniería. En el marco de este proyecto se llegaron a desplegar tres sensores en los salones 314 y 312.

3.3.2. Motivación y objetivos

La motivación de esta propuesta consiste en tener la capacidad de conocer la calidad de aire dentro de la Facultad. Se desea a partir de las mediciones realizadas para poder detectar los escenarios de mayor riesgo y estudiar sus causas para poder anticiparlos, prevenirlos y reaccionar a ellos con el fin a largo plazo de mejorar lo más posible las condiciones salubres de los concurrentes a la Facultad.

El objetivo es obtener una medición constante de la calidad de aire así como de otros datos como temperatura, humedad, cantidad de personas y nivel de sonido en distintos salones de la Facultad. Se desea generar estos datos de forma continua en el tiempo y tener la capacidad de poder consumirlos para analizarlos y buscar patrones. La generación de alertas y reacciones ante estas mediciones no está contemplada en este proyecto.

3.3.3. Implementación

Para llevar a cabo la implementación de este caso de uso se tomó como punto de partida el proyecto anteriormente mencionado de sensores de calidad de aire[99] el cual posee el código fuente y los procedimientos definidos para poner en funcionamiento un sensor de calidad de aire utilizando: una placa de desarrollo Sparkfun LoRa Gateway 1- Channel con chip ESP32, un sensor de calidad de aire SGP30, un sensor de clima Si7021 y un sensor de sonido. Este proyecto anterior también

aborda una investigación sobre las posibilidades de conectividad que brinda el dispositivo y provee el código fuente de funcionalidades de transmisión de datos por medio de LoRa y WiFi en conjunto con los resultados de una prueba de conectividad con LoRa en distintas zonas de la Facultad. En este análisis se puede observar que los salones del tercer piso tienen una conectividad LoRa muy mala pero en cambio tienen una señal de WiFi especialmente buena gracias a la red *wifing*, por lo que solo se utilizara WiFi para este caso de uso.

El software del dispositivo fue mejorado en varios aspectos. Por un lado fue añadida la funcionalidad de conteo de personas que utiliza las direcciones MACs obtenidas por medio del esnifeo de tramas de red que viajan por el aire. Esto se logra al setear la antena WiFi del ESP32 en modo promiscuo. La idea de la funcionalidad fue tomada del proyecto (Detta, 2020, [25]) y se adaptó para el dispositivo y framework utilizado en el proyecto. Esta función logra realizar un conteo estimado de la cantidad de dispositivos conectados a la red WiFi, por lo que asumiendo que cada persona carga consigo un dispositivo móvil conectado a esta red WiFi se termina obteniendo un número estimado de la cantidad de personas. La función también es capaz de filtrar estas tramas mediante la creación de una lista fija de MACs a ignorar seteada a nivel de código y también mediante un parámetro opcional llamado RSSI (Received Signal Strength Indicator) obtenido del cabezal de capa de enlace de las tramas recibidas. El RSSI indica la fuerza de la señal al ser recibida, tiene un valor que se encuentra entre -90 (muy mala señal) y -50 (excelente señal) por lo que se puede asignar un valor deseado a la variable de filtrado para ignorar todas las tramas que lleguen con menor RSSI, acotando de esta forma el rango de distancia a escanear.

Sumado a esto se realizó la adaptación de un nuevo sensor calidad de aire, el sensor Sparkfun SGP40. En el proyecto anterior se hizo uso del sensor Sparkfun SGP30, el cual es una iteración previa al SGP40 y utiliza un paradigma de funcionamiento completamente distinto a su contraparte más moderna. Es por esto que en la Subsubsección 3.3.3.2 se los ve en más detalle y se provee una comparativa entre estos dos sensores.

Otro cambio significativo fue la mejora en el algoritmo de medición del sensor, en donde anteriormente se realizaba una única medición cada un cierto intervalo de tiempo determinado por medio de todos sus sensores para posteriormente enviar esos datos puntuales. Se modificó dicho comportamiento para que durante el tiempo de espera entre envíos se realicen mediciones constantes, pudiendo así relevar una mayor cantidad de datos en cada envío en donde se incluyen promedios de medi-

ción, máximos y mínimos en cada variable. Todos los nuevos datos generados en cada envío son detallados en la Subsubsección 3.3.3.4.

En lo que respecta al hardware, para poder elaborar un dispositivo compacto y resistente se diseñó y fabricó una caja para almacenarlo. Esta caja está hecha en madera de eucalipto compensada de 3mm cortada por láser con una máquina CNC. Por otro lado, para la colocación de la placa Sparkfun ESP32 junto con el resto de sensores se diseñó y fabricó un circuito con las pistas necesarias sobre una placa de pertinax. Como se menciona en la introducción de este caso de uso, se fabricaron en total nueve sensores de ambiente y calidad de aire, sin embargo para poder obtener la mayor cantidad de sensores posible dentro de los recursos disponibles, se optó por crear variantes del mismo sensor las cuales poseen dos de los tres sensores citados anteriormente. Más precisamente se realizaron combinaciones que puedan cumplir su rol fundamental de monitoreo del aire junto con una de las variables de monitoreo de ambiente: nivel de sonido o temperatura y humedad. En la Tabla 3.1 se lista cada dispositivo fabricado junto con los sensores de los que dispone.

A continuación se muestran imágenes de algunos de los dispositivos finalizados y esquemas de los diagramas de conexiones de cada variante generada.

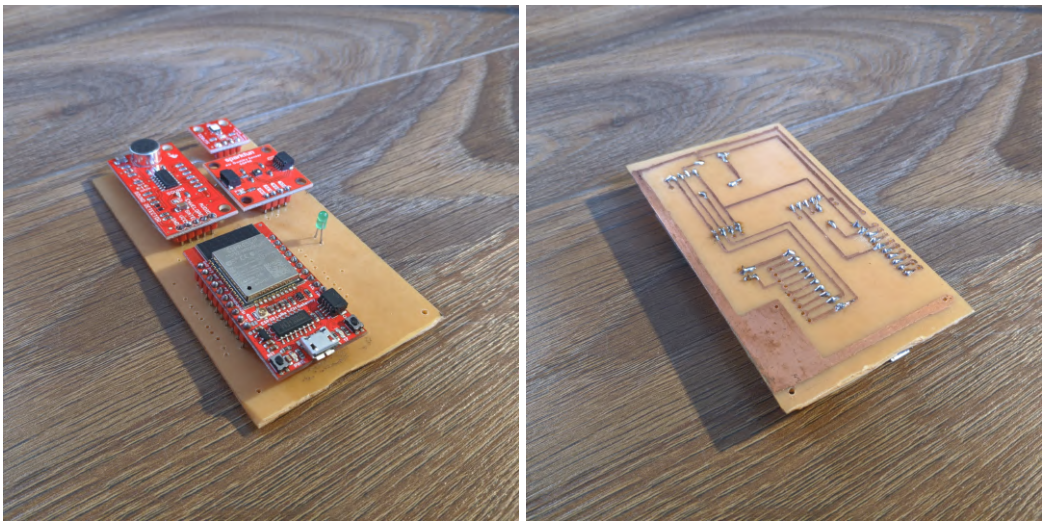

Figura 3.4: ***Fotografías de la placa principal de un sensor de ambiente y calidad de aire fabricado. Izquierda: parte superior de la placa con el ESP32 y sensores soldados. Derecha: parte inferior de la misma placa en la cual se aprecian las pistas.***

Como se puede apreciar en la imagen derecha de la Figura 3.4, todos los pines del ESP32 (incluyendo los que no se usan) están soldados su respectiva pista y sobre el otro extremo de la misma tienen otra perforación. Esto se hizo para que la placa

sea más reutilizable y flexible sin tener que desoldar los componentes, permitiendo utilizar a futuro estos pines actualmente inactivos del ESP32 de ser necesario, por ejemplo en el caso de querer agregarle un nuevo sensor al dispositivo.

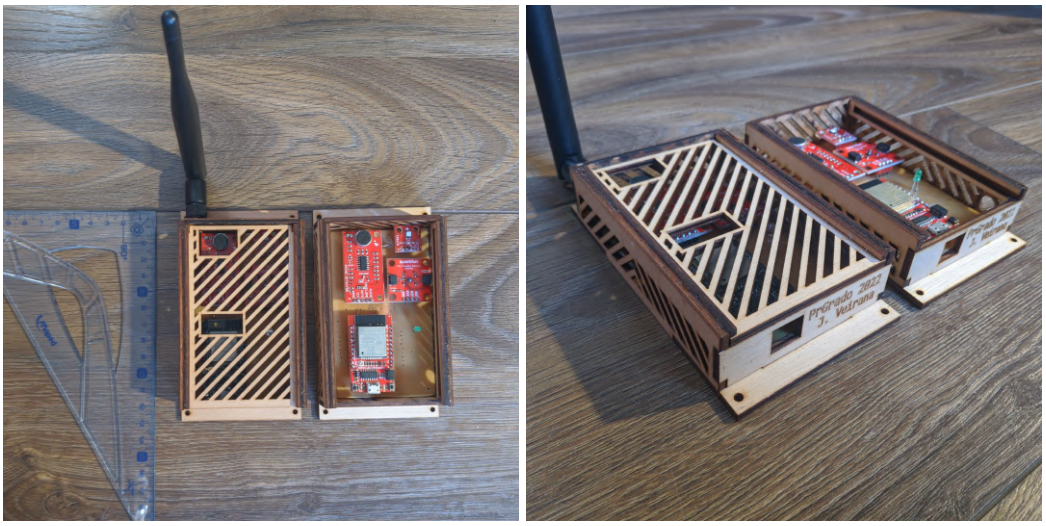

Figura 3.5: ***Fotografías de dos sensores de ambiente y calidad de aire terminados dentro de sus respectivas cajas, uno con la tapa retirada para mostrar su contenido y sin la antena LoRa.***

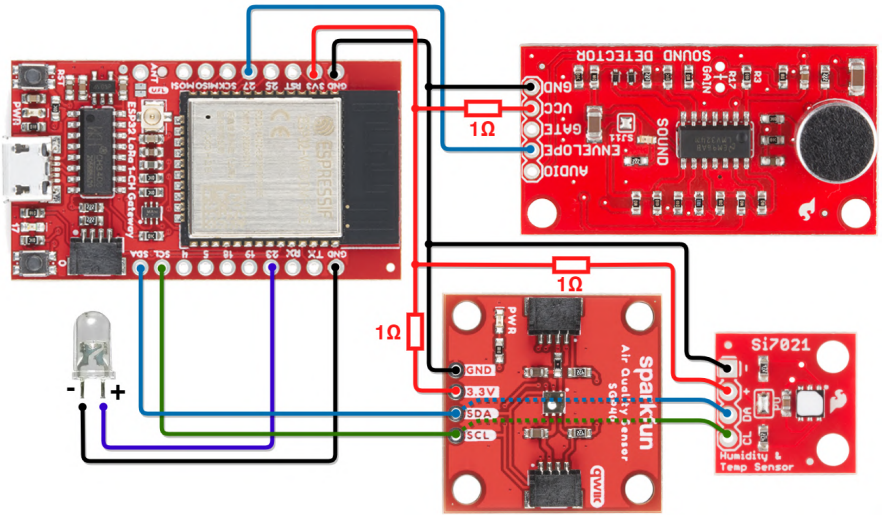

Figura 3.6: ***Diagrama de conexiones del sensor de ambiente y calidad de aire completo.***

[1] sca001 = Sensor de Calidad de Aire 001

[2] Sensor de calidad de aire SGP30 heredado del proyecto de TSCF[99], es el único de su tipo en existencia al momento de realizado el proyecto.

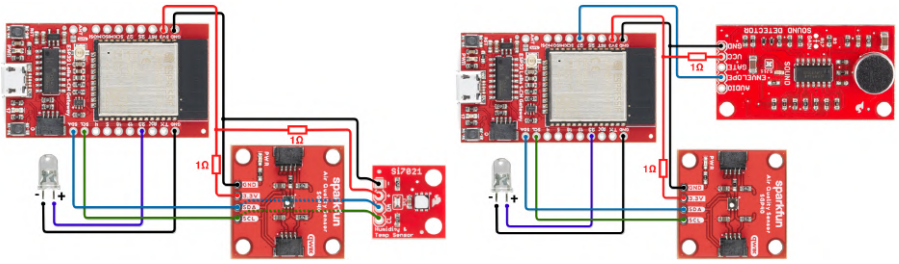

Figura 3.7: ***Diagrama de conexiones de las variantes del sensor de ambiente y calidad de aire. Izquierda: variante sin sensor de sonido. Derecha: variante sin sensor de temperatura y humedad.***

Id	Variante	Sensores Integrados
sca001[1]	1	SGP30[2] - Si7021 - Sonido
sca002	2	SGP40 - Si7021 - Sonido
sca003	2	SGP40 - Si7021 - Sonido
sca004	3	SGP40 - Sonido
sca005	3	SGP40 - Sonido
sca006	3	SGP40 - Sonido
sca007	4	SGP40 - Si7021
sca008	4	SGP40 - Si7021
sca009	4	SGP40 - Si7021

Tabla 3.1: Lista de sensores de ambiente y calidad de aire fabricados

3.3.3.1. Introducción a los compuestos orgánicos volátiles (VOCs)

Los compuestos orgánicos volátiles (VOCs por sus siglas en inglés), son compuestos orgánicos que contienen elementos como el carbono, hidrógeno, oxígeno, flúor, cloro, bromo, azufre o nitrógeno y cumplen que a temperatura y presión ambiente son altamente volátiles, librándose en dichas condiciones gases de los mismos.

Estos compuestos están fuertemente presentes en productos químicos como pinturas, disolventes, aerosoles, productos de limpieza, combustibles, materiales de construcción, maderas tratadas, textiles y otros. Sin embargo, además de existir en productos químicos artificiales también están presentes en la exhalación de los seres humanos, más precisamente los compuestos isopreno, acetona, etanol y metanol(Fenske y Paulson, 1999, [39]). Sumado a esto, se sabe que tanto los VOCs como el CO2 proveniente de la respiración humana llevan un comportamiento similar en interiores, por lo que en ambientes cerrados, con poca ventilación y gran densidad de personas, las concentraciones tienden a aumentar de forma similar y es posible lograr la disminución o control de las mismas mediante la ventilación del ambiente

o el recambio del aire (Liu et al., 2016, [57]) (Stönner, Edtbauer y Williams, 2018, [91]). Por esta razón es posible utilizar los VOCs como un indicador de la calidad del aire.

3.3.3.2. Hardware extra utilizado

Detalles del sensor SGP40: El SGP40 es un sensor de la marca Sparkfun que mide la calidad de aire (Figura 3.8), es el sucesor del sensor SGP30 [81]. El SGP40 utiliza el sensor *CMOSens* de la empresa suiza *Sensirion*[70] y difiere drásticamente en la forma de medir respecto a la de su antecesor. A diferencia del SGP30, el SGP40 no realiza estimaciones de concentraciones de CO2 en el aire de forma directa, sino que calcula un índice de calidad de aire relativo por medio de un algoritmo específico y basándose en las concentraciones actuales de VOCs (compuestos orgánicos volátiles) en el aire y el historial reciente de mediciones.

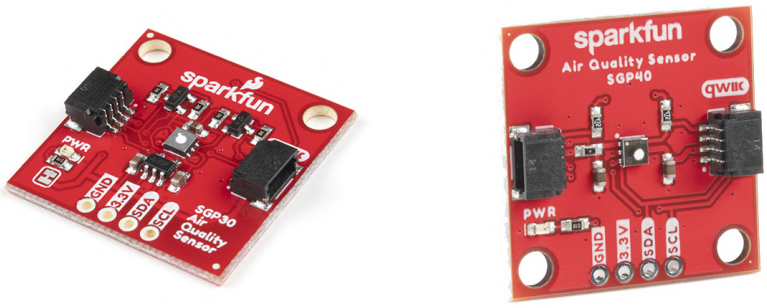

Figura 3.8: ***Fotografías del sensor Sparkfun SGP30 (izq) y SGP40 (der).*** Fuente: sparkfun.com [81][82]

El índice está definido con una escala relativa al ambiente que va de las 0 a las 500 unidades, el 100 indica el valor estándar de la calidad de aire y es calculado en base al promedio de las mediciones de las últimas veinticuatro horas. Los valores mayores a 100 indican que la calidad de aire está empeorando respecto a este promedio y los valores menores a 100 indican que la calidad de aire está mejorando, ver la Figura 3.9 en la que se muestra un ejemplo de una medición junto con su interpretación sobre el eje vertical y una recomendación de en que momento seria conveniente ventilar la habitación.

Gracias a la forma en la que está definido el índice, el sensor puede adaptarse a distintos ambientes independientemente de la buena o mala calidad de aire que en

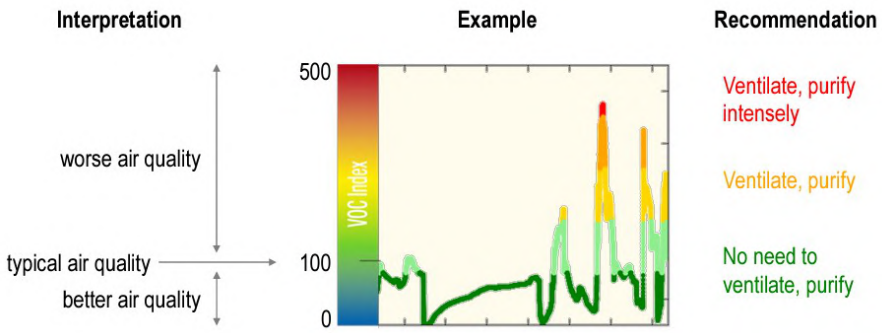

Figura 3.9: ***Ejemplo ilustrativo del índice del sensor SGP40 en funcionamiento.*** Fuente: Sensirion [77]

promedio tengan para luego comenzar respecto a ese punto a detectar los cambios en las concentraciones.

Detalles del sensor Si7021: Este sensor de temperatura y humedad fabricado por Sparkfun[87] (Figura 3.10) utiliza un sensor Si7021 fabricado por *Silicon Labs* que mide la temperatura y la humedad ambiente con una alta precisión y proporciona los datos por medio de pines del bus I2C.

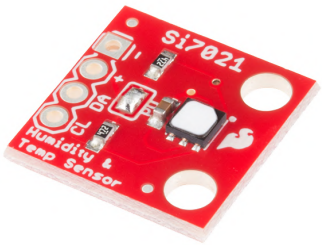

Figura 3.10: ***Fotografía del sensor Sparkfun Si7021 de temperatura y humedad.*** Fuente: sparkfun.com [87]

Detalles del sensor de sonido: Este sensor de sonido fabricado por Sparkfun[88] (Figura 3.11) utiliza un micrófono de electreto para medir las ondas de sonido en combinación con distintos chips fabricados por *Fairchild Semiconductor* para proveer tres salidas distintas por medio de sus pines:

- Salida de audio (pin Audio, digital)
- Salida de la onda envolvente superior (pin Envelope, analógico)

- Salida binaria indicativa de presencia de ruido (pin Gate, digital)

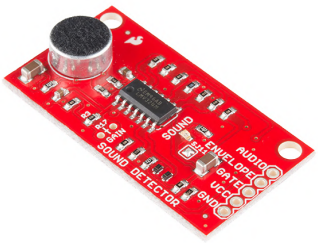

Figura 3.11: ***Fotografía del sensor Sparkfun de sonido.*** Fuente: sparkfun.com [88]

Para el dispositivo final se utilizó este pin de la envolvente para censar el nivel de ruido del ambiente, cabe aclarar que este dato no posee una unidad de medida directa sino que se puede entender como una medida relativa de la cual se podrían determinar intervalos que den una semántica a la medición, por ejemplo determinando que las mediciones superiores a cierto valor de la envolvente sean consideradas un sonido ambiente alto en un salón de clases.

La envolvente modela la evolución del sonido en el tiempo, esta no depende únicamente del volumen pero como se aprecia en la Figura 3.12, sí está directamente relacionada ya que el volumen es definido por la amplitud de la onda de sonido y a mayor amplitud mayor será la envolvente.

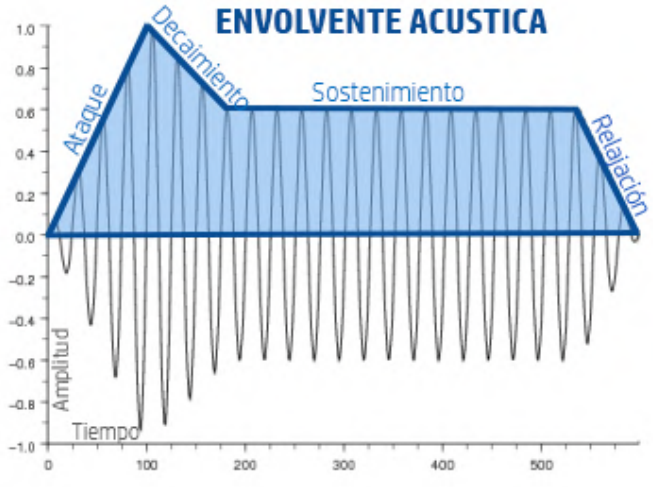

Figura 3.12: ***Función envolvente de una onda de sonido.*** Fuente: SoundSystems.es, 2019, [80]

3.3.3.3. Software extra utilizado

Bibliotecas de los sensores: Sparkfun provee bibliotecas escritas en C para poder utilizar los sensores de forma directa, estas se encuentran en la página web del sensor sobre sección de documentación del sensor, las bibliotecas usadas son:

- Biblioteca del sensor **SGP40:** https://github.com/sparkfun/SparkFun_SGP40_Arduino_Library
- Biblioteca del sensor **Si7021:** https://github.com/sparkfun/SparkFun_Si7021_Arduino_Library

Para el sensor de sonido no se necesito una biblioteca ya que fue suficiente con realizar una lectura analógica del pin para obtener los datos.

Código fuente del contador de personas: También se utilizó como referencia el código del proyecto de conteo de MACs: https://github.com/fdetta/Lua-RTOS-ESP32

3.3.3.4. Datos generados por el sensor

En esta sección se detallan todos los datos que se generan en cada envío hacia la plataforma. En general se pueden identificar cuatro datos correspondientes a cada una de las variables censadas: *avg*, *last*, *max* y *min*. *Avg* corresponde al valor promedio calculado de todas las mediciones realizadas de manera continua entre envío y envío. *Last* corresponde al último valor medido antes de realizar un envío y se toma como el valor 'instantáneo' relacionado al instante de tiempo en el que se reciben los datos en la plataforma. *Max* y *Min* corresponden al valor máximo y mínimo que se pudo llegar a medir entre envío y envío.

Es de notar que cuanto más distantes en el tiempo se hagan los envíos, más valor van a aportar estos cuatro campos medidos, puesto que cuando los envíos se realizan muy cercanos en el tiempo los cuatro valores tienden a converger a un único valor.

Los datos generados son:

- *id* [6 Bytes]: ID del dispositivo
- *aux* [2 Bytes]: Banderas auxiliares (no tienen ningún uso asignado)
- *avg_humidity* [8 Bytes]: Promedio de humedad (en %) medido entre envíos

- *last_humidity* [8 Bytes]: Valor de humedad (en %) medido al momento de enviar
- *max_humidity* [8 Bytes]: Valor máximo de humedad (en %) medido entre envíos
- *min_humidity* [8 Bytes]: Valor mínimo de humedad (en %) medido entre envíos
- *avg_temperature* [8 Bytes]: Promedio de temperatura (en °C) medido entre envíos
- *last_temperature* [8 Bytes]: Valor de temperatura (en °C) medido al momento de enviar
- *max_temperature* [8 Bytes]: Valor máximo de temperatura (en °C) medido entre envíos
- *min_temperature* [8 Bytes]: Valor mínimo de temperatura (en °C) medido entre envíos
- *avg_co2* [8 Bytes]: Promedio de CO2 (en ppm) medido entre envíos (Solo válido en dispositivos de variante 1 con SGP30)
- *last_co2* [8 Bytes]: Valor de CO2 (en ppm) medido al momento de enviar (Solo válido en dispositivos de variante 1 con SGP30)
- *max_co2* [8 Bytes]: Valor máximo de CO2 (en ppm) medido entre envíos (Solo válido en dispositivos de variante 1 con SGP30)
- *min_co2* [8 Bytes]: Valor mínimo de CO2 (en ppm) medido entre envíos (Solo válido en dispositivos de variante 1 con SGP30)
- *avg_tvoc* [8 Bytes]: Promedio de VOCs medido entre envíos, la unidad es ppm para dispositivos de variante 1 e índice SGP40 para el resto de variantes
- *last_tvoc* [8 Bytes]: Valor de VOCs medido al momento de enviar, la unidad es ppm para dispositivos de variante 1 e índice SGP40 para el resto de variantes
- *max_tvoc* [8 Bytes]: Valor máximo de VOCs medido entre envíos, la unidad es ppm para dispositivos de variante 1 e índice SGP40 para el resto de variantes
- *min_tvoc* [8 Bytes]: Valor mínimo de VOCs medido entre envíos, la unidad es ppm para dispositivos de variante 1 e índice SGP40 para el resto de variantes
- *avg_sound* [8 Bytes]: Promedio de sonido (unidad relativa) medido entre envíos
- *last_sound* [8 Bytes]: Valor de sonido (unidad relativa) medido al momento

de enviar

- *max_sound* [8 Bytes]: Valor máximo de sonido (unidad relativa) medido entre envíos
- *min_sound* [8 Bytes]: Valor mínimo de sonido (unidad relativa) medido entre envíos
- *rssi_filter_value* [8 Bytes]: Valor que se está considerando como el mínimo RSSI válido para las MACs contadas
- *avg_rssi_MACs_count* [8 Bytes]: Promedio de cantidad de MACs contadas entre envíos con RSSI por encima del *rssi_filter_value*
- *avg_total_MACs_count* [8 Bytes]: Promedio de cantidad de MACs contadas entre envíos sin aplicar el filtro
- *avg_filter_MACs_count* [8 Bytes]: Promedio de cantidad de MACs filtradas por estar por debajo del RSSI *rssi_filter_value* contadas entre envíos
- *last_rssi_MACs_count* [8 Bytes]: Cantidad de MACs con RSSI por encima del *rssi_filter_value* contadas al momento de enviar
- *last_total_MACs_count* [8 Bytes]: Cantidad de MACs sin aplicar el filtro contadas al momento de enviar
- *max_MACs_count* [8 Bytes]: Cantidad máxima de MACs sin aplicar el filtro contadas entre envíos
- *min_MACs_count* [8 Bytes]: Cantidad mínima de MACs sin aplicar el filtro contadas entre envíos

3.3.3.5. Despliegues realizados

Fueron desplegados tres dispositivos en dos salones de Facultad, más precisamente se desplegaron los dispositivos *sca001* y *sca002* conjuntamente en el salón 314 para poder comparar las mediciones del sensor SGP30 y SGP40 estando en un mismo ambiente (ver la Figura 3.13).

Por otra parte se desplegó el sensor *sca003* en el salón de PCs Windows 312 (ver Figura 3.14).

3.3.4. Resultados obtenidos

En esta sección se muestran los resultados obtenidos de las mediciones realizadas en los despliegues de los salones 314 y 312. Comenzando por el salón 314 el cual tiene instalado el dispositivo *sca001* con el sensor de aire SGP30 y el *sca002*

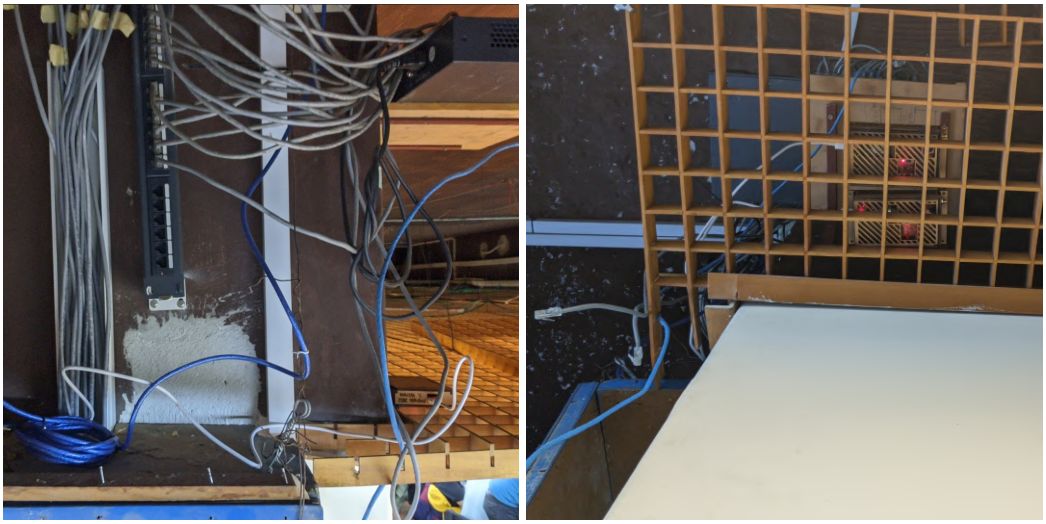

Figura 3.13: ***Sensores de calidad de aire instalados en el salón 314.***

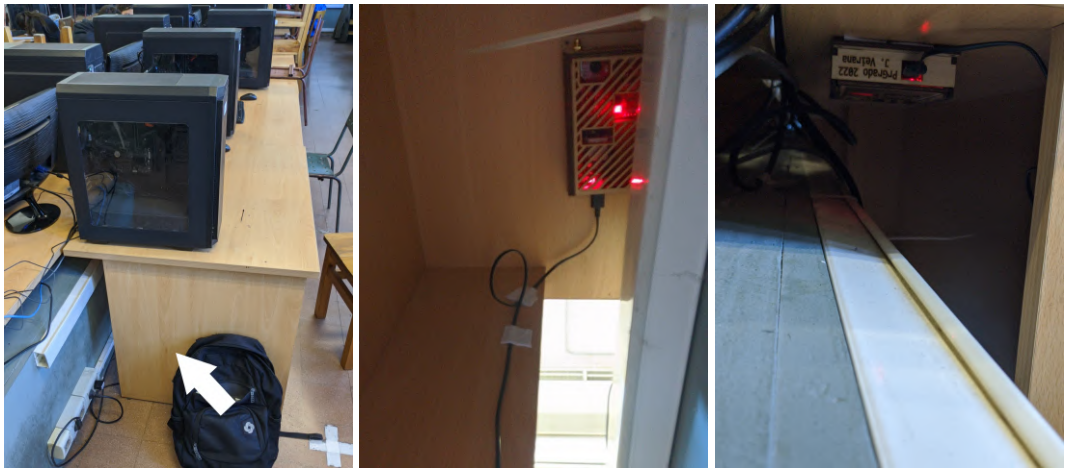

Figura 3.14: ***Sensor de calidad de aire instalado en el salón 312.***

con el sensor SGP40 para poder comprobar el funcionamiento de ambos y además realizar comparativas entre los distintos métodos con los que miden la calidad de aire. Luego se muestran los resultados del salón 312 con el dispositivo *sca003* con sensor SGP40. Todos los dispositivos estaban configurados para enviar datos aproximadamente cada un minuto.

Para ambos salones se tomaron los datos obtenidos de medir una semana de continuo para realizar el análisis, más precisamente la semana del 6 de noviembre al 12 de noviembre de 2022.

3.3.4.1. Observaciones generales sobre los resultados

Comenzaremos por el final nombrando las conclusiones y observaciones obtenidas de las mediciones que se presentarán de ahora en más.

La primera observación es que se obtuvo una respuesta positiva en el funcionamiento de los dispositivos, los cuales enviaron datos durante la semana completa con buena estabilidad. El mayor intervalo de desconexión detectado fue de poco más de una hora y media de no recibir datos de uno de los dispositivos. Más precisamente esto ocurrió en el dispositivo *sca001* el 8 de noviembre a partir de las 00:00hs hasta aproximadamente las 1:40hs. Luego se detectaron en el mismo dispositivo otros cinco intervalos esparcidos en la semana en los que no se recibieron datos, con una duración variable de entre 15 y 30 minutos. Por otro lado el dispositivo *sca002* solo tuvo dos cortes uno de 12 minutos aproximadamente el día 7 de noviembre a las 15:15hs y otro de 30 minutos el día 9 de noviembre a las 10:00hs. No se llegó a estudiar el motivo de estas desconexiones por lo que queda pendiente como trabajo futuro, lo que se puede deducir sin embargo es que los dispositivos *sca001* y *sca002* que se encontraban en la misma localización y estaban conectados a la misma red, experimentaron cortes en distintos momentos el uno del otro, por lo que en principio no se tomaría a la red WiFi como punto de falla. Un dato adicional que puede ser de ayuda para esta investigación es que los dispositivos se desconectan de la red WiFi antes de cada envío para poder realizar el conteo de MACs con su antena en modo promiscuo, luego se vuelven a conectar a la red WiFi para enviar los datos y se repite el ciclo. Esto está generando conexiones y desconexiones constantes a la red, separadas por un intervalo de aproximadamente un minuto.

Sobre los datos recolectados las observaciones obtenidas son que por un lado se pudo comprobar que como era de esperar, los horarios en donde la calidad de aire de los salones empeora son en aquellos en los que se acumula una mayor cantidad de personas, generalmente por las tardes a partir de las 14hs. Sobre esto último, si bien por experiencia propia es sabido que en las tardes es cuando más cantidad de personas hay en estos salones, se pudo ver reflejado en los datos también gracias al contador de MACs en los sensores.

Se observó también que el índice de VOCs empeora bruscamente todos los días entre semana en horarios matutinos, usualmente entre las 7 y 8 de la mañana. Dado que en estos horarios se presume que no suelen haber muchos estudiantes en dichos salones y que además se conoce que el personal de aseo de la Facultad se encarga de limpiar cada salón al inicio de la jornada, se presenta la hipótesis de que el índice

reacciona por los productos de limpieza usados cuando estos se evaporan. Como se explica en la Subsubsección 3.3.3.1, algunos químicos presentes en los productos de limpieza forman parte de estos compuestos y son captados por el sensor.

3.3.4.2. Datos obtenidos en el salón 314

Detección de VOCs en la mañana: Una de las observaciones nombradas fue la detección de un aumento en el índice de calidad de aire del detector SGP40 en la mañana entre las 7:00hs y las 8:00hs. En el gráfico de la Figura 3.15 se muestran las mediciones del domingo 6 de noviembre desde las 00:00hs hasta las 00:00hs del martes 8. Se puede apreciar en la parte del domingo que no se ve el pico en la mañana puesto que ese día no se realiza la limpieza, por el contrario el sensor detecta que el aire va mejorando en relación al promedio de las últimas 24hs. Luego el lunes puede verse un aumento drástico en la medición aproximadamente a las 7 de la mañana. También se puede observar la evolución del índice a partir del mediodía del lunes, en donde llega a marcar picos de 300 (el triple del promedio diario).

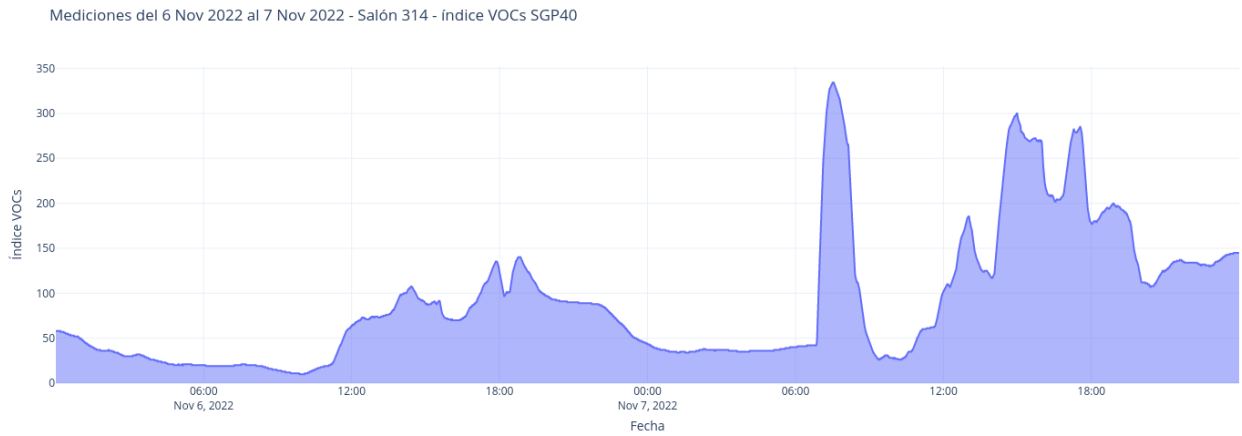

Figura 3.15: ***Gráfica de la medición del domingo 6/11/22 y el lunes 7/11/22 en salón 314.***

Comparación entre la medición del SGP30 con el SGP40: En la Figura 3.16 se muestran los datos obtenidos de la medición de CO2 con el sensor SGP30 en comparación directa con el índice de compuestos orgánicos volátiles calculado por el SGP40. Sobre la izquierda de la figura se observa la escala del nivel de CO2, las mediciones base por defecto de un ambiente al aire libre son de 400ppm. Por otro lado sobre la derecha se encuentra la escala del índice de compuestos orgánicos volátiles la cual como se mencionó anteriormente tiene su nivel base en 100 ante

un ambiente de calidad de aire estable. Se puede observar que a grandes rasgos los patrones en ambas funciones se correlacionan y aumentan simultáneamente, viendo así que el SGP30 y SGP40 pueden ser usados de la misma forma y se puede esperar que su funcionamiento sea a grandes rasgos similar.

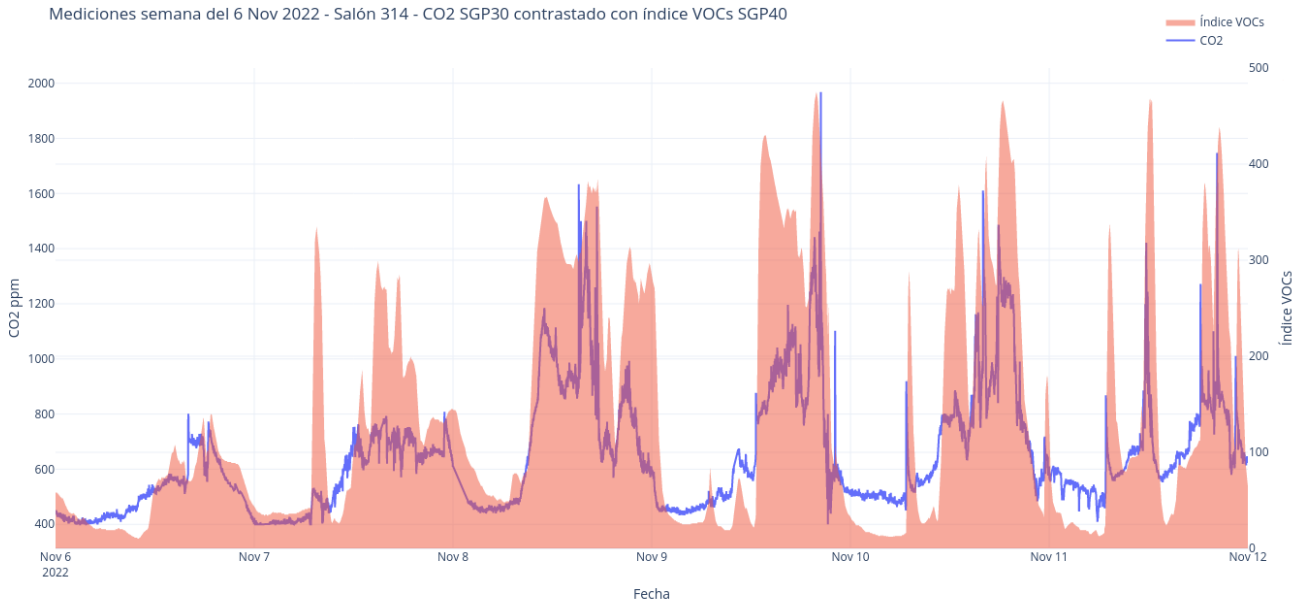

Figura 3.16: ***Comparación de la medición de CO2 del SGP30 contra el índice VOCs del SGP40 en el salón 314.***

Comparación entre las mediciones de aire y la cantidad de MACs: En esta sección se introducen los resultados del contador de MACs para contrastarlo con las mediciones de calidad de aire. Sobre la izquierda en el eje vertical se encuentran en dos gráficas distintas las escalas de calidad de aire tanto del SGP30 (en la Figura 3.17) como el SGP40 (en la Figura 3.18) y sobre la derecha de ambas gráficas nombradas se encuentra el eje vertical con la cantidad de MACs. Se puede observar que el contador de MACs sigue el mismo patrón que el de calidad de aire para ambos dispositivos y que el máximo número de direcciones contado es de 30 dispositivos.

Comparación entre la intensidad de sonido respecto a las MACs y VOCs: Se introducen en la Figura 3.19 los resultados del sensor de sonido para ser contrastado con las mediciones de calidad de aire. Sobre la izquierda en el eje vertical se encuentra la escala de la intensidad de sonido topeada a 1000 como valor máximo

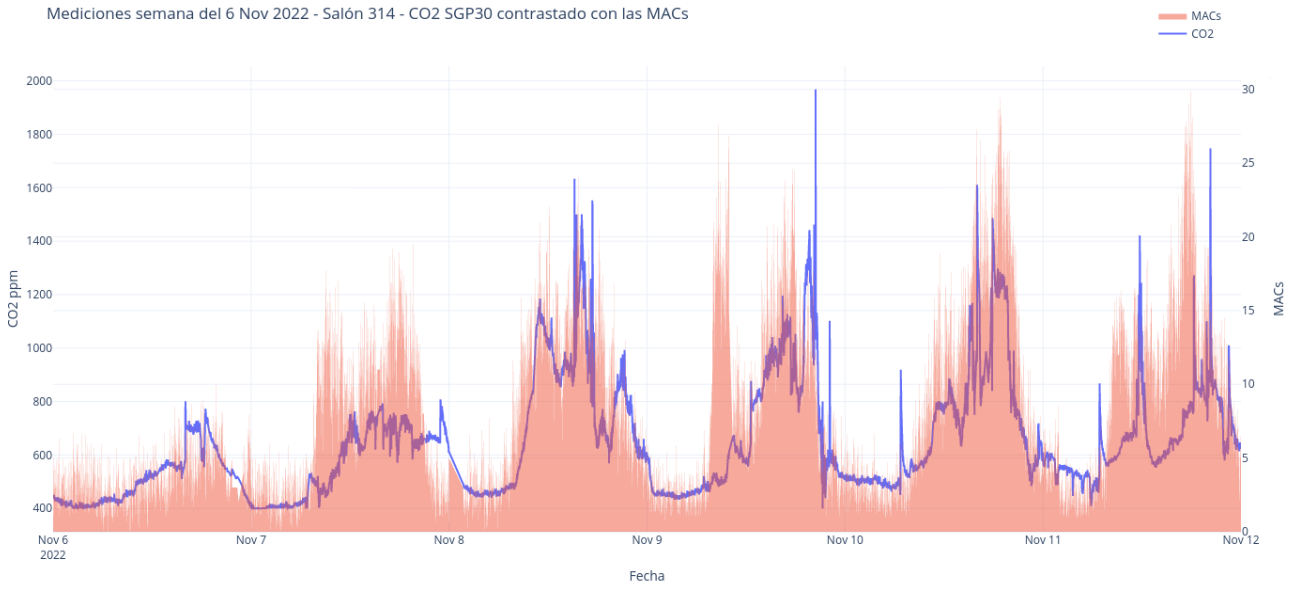

Figura 3.17: ***Comparación de la medición de CO2 del SGP30 contra la cantidad de MACs en el salón 314.***

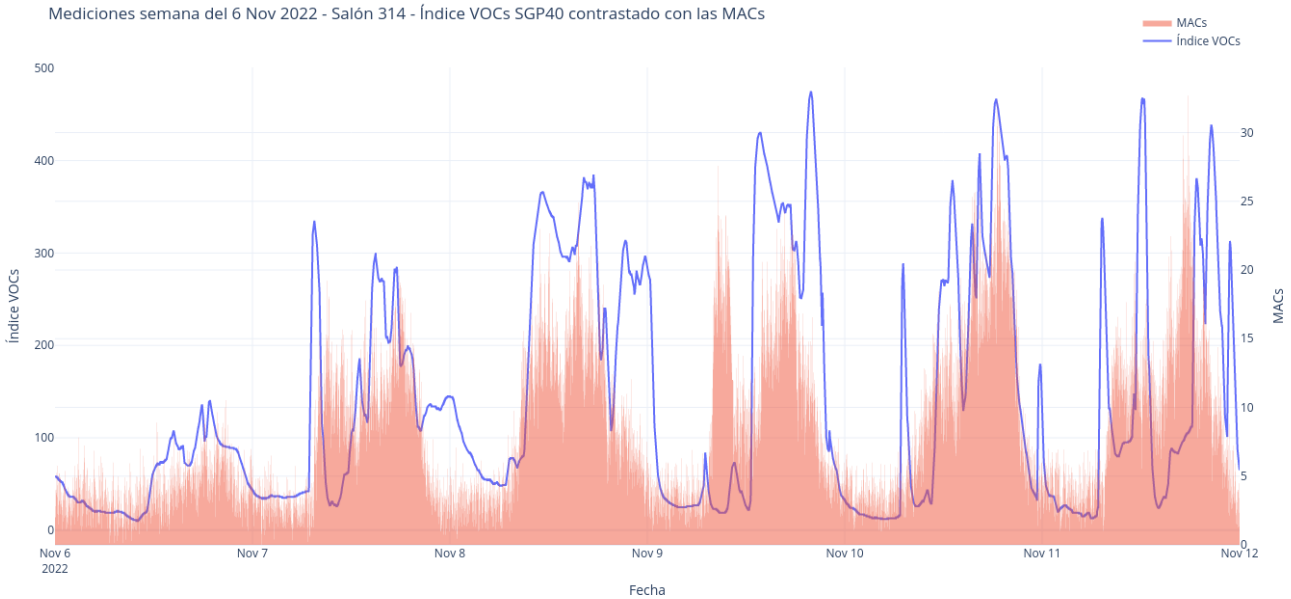

Figura 3.18: ***Comparación de la medición de VOCs del SGP40 contra la cantidad de MACs en el salón 314.***

para que facilite la observación y sobre la derecha se encuentra el eje vertical con el índice de calidad de aire del SGP40. Se puede observar que la intensidad de sonido si bien es totalmente cambiante en cortos periodos de tiempo, nuevamente sigue el

mismo patrón que la calidad de aire.

En la Figura 3.20 se muestran los resultados del sensor de sonido para contrastarlo con las mediciones del contador de MACs. Sobre la izquierda en el eje vertical se encuentra la escala de la intensidad de sonido topeada a 1000 como valor máximo para que facilite la observación y sobre la derecha se encuentra el eje vertical con la cantidad de MACs. La conclusión obtenida de esta relación es la misma que la obtenida en la comparativa de sonido-VOCs.

Para una mejor visualización se proveen también las mediciones solamente del día 7 de Noviembre en la Figura 3.21, mostrando el contraste entre sonido y VOCs/MACs.

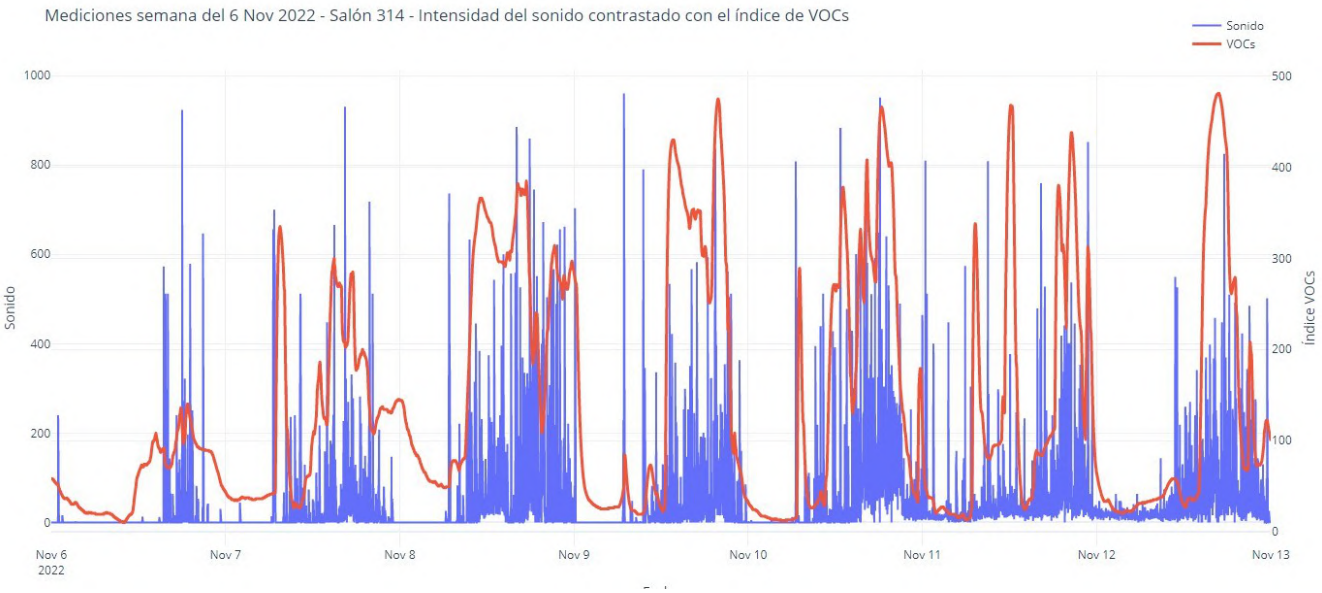

Figura 3.19: ***Comparación de la medición de VOCs del SGP40 contra la intensidad de sonido en el salón 314.***

Comparación entre el conteo de MACs de ambos dispositivos usados: Finalmente se presenta en la Figura 3.22 las mediciones del contador de MACs de ambos dispositivos, los cuales como se mostró en las fotografías del despliegue están en la misma ubicación física. Se puede apreciar que si bien los conteos pueden fluctuar bastante de una medición a la siguiente, a grandes rasgos siguen el mismo patrón por lo que se puede afirmar que el conteo funciona de manera similar en los distintos dispositivos.

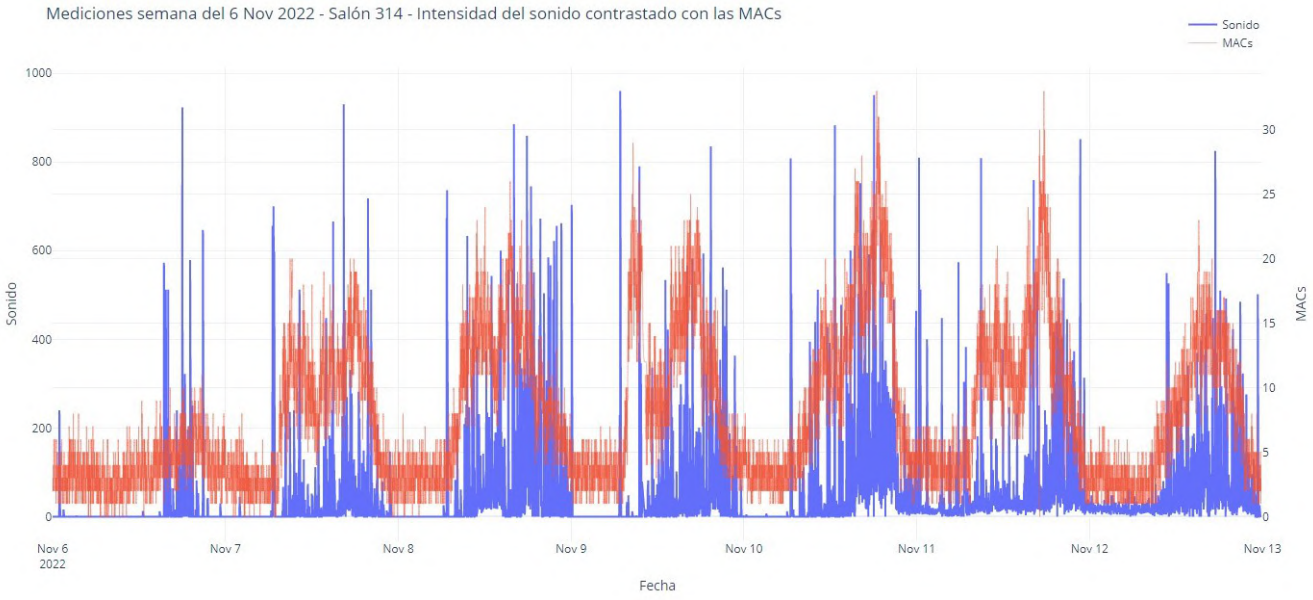

Figura 3.20: ***Comparación de la medición de cantidad de MACs contra la intensidad de sonido en el salón 314.***

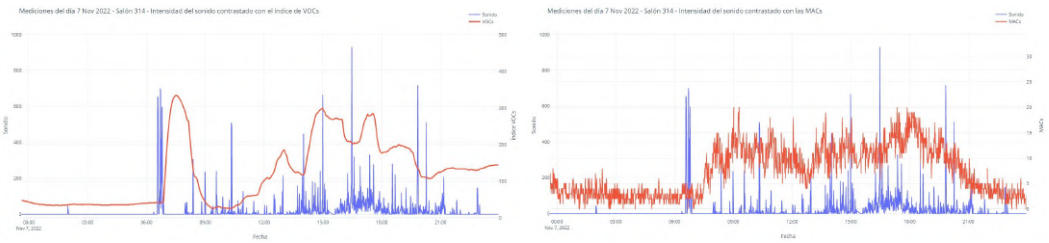

Figura 3.21: ***Izquierda: Comparación de la intensidad de sonido contra la calidad de aire. Derecha: Comparación de la intensidad de sonido contra la cantidad de MACs. Ambas medidas el 7/11/2022 en el salon 314.***

3.3.4.3. Datos obtenidos en el salón 312

Comparación entre las mediciones de aire y la cantidad de MACs: Haciendo la misma comparación que en el salón 314 entre la calidad de aire y la cantidad de MACs se reconocen en la Figura 3.23 patrones similares pero con fluctuaciones más violentas durante las tardes. Es de notar que la ubicación física de este sensor puede no ser la mejor puesto que está debajo de un escritorio rodeado de paredes tanto del muro bajo que divide los escritorios en el centro del salón como de las distintas partes del escritorio donde esta instalado.

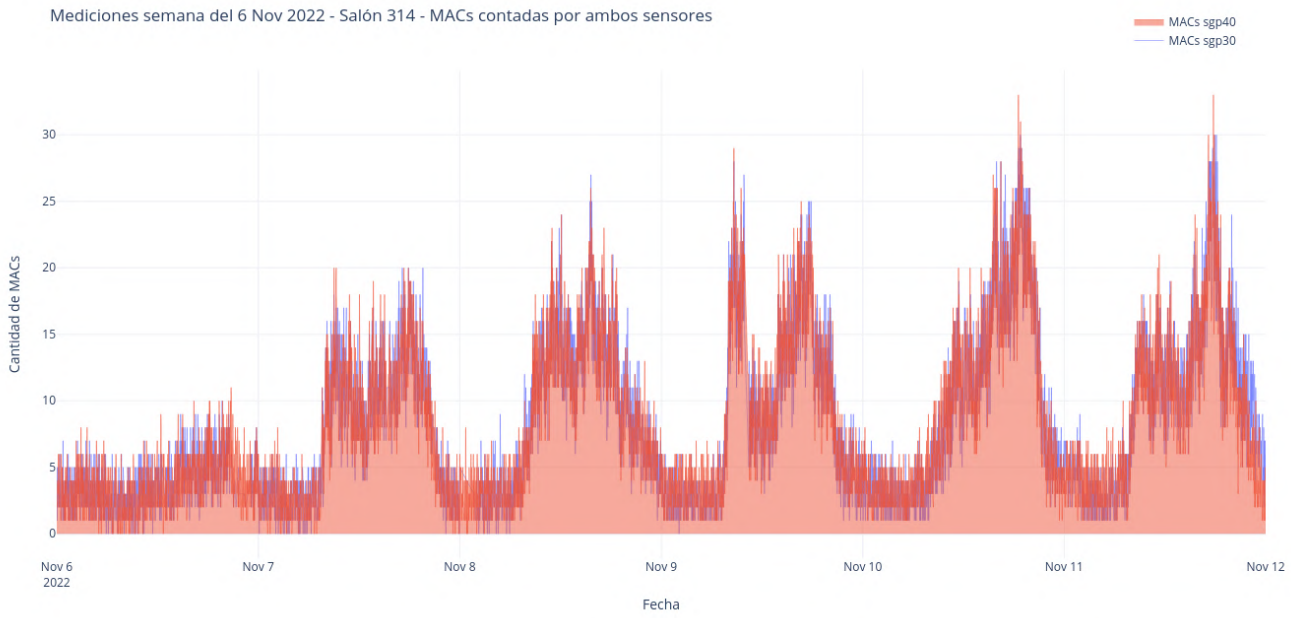

Figura 3.22: ***Comparación de la cantidad de MACs medidas por ambos dispositivos en el salón 314.***

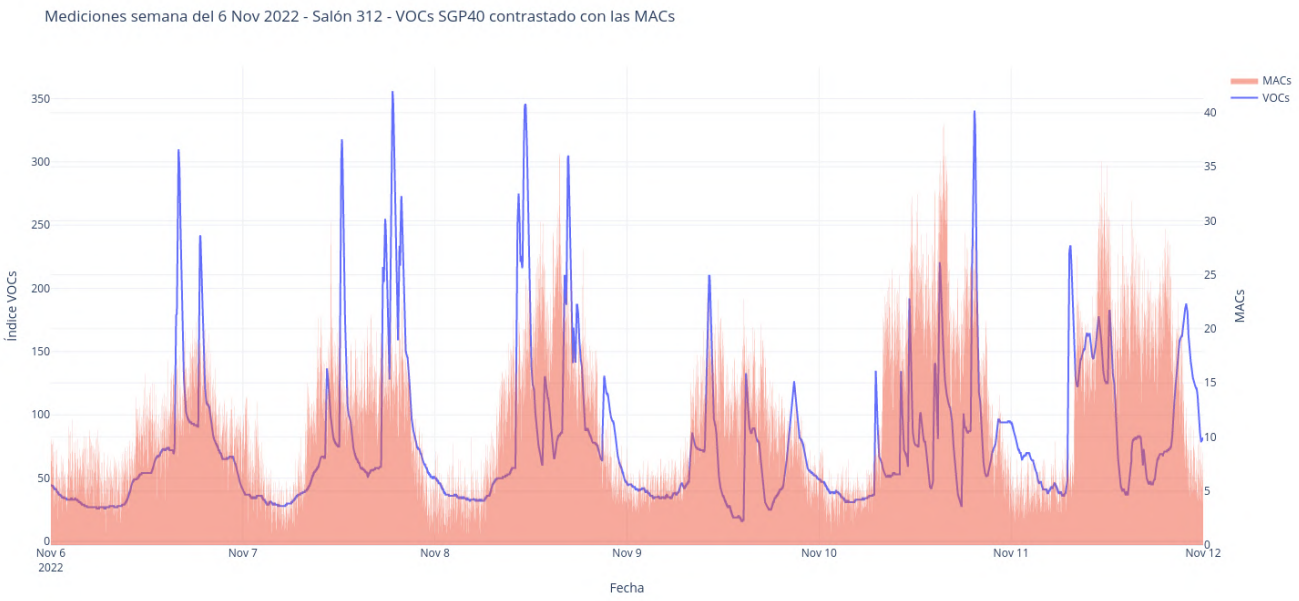

Figura 3.23: ***Comparación de la medición de VOCs del SGP40 contra la cantidad de MACs en el salón 312.***

3.3.5. Mejoras sugeridas y trabajo futuro

En esta sección se discuten algunos apartados mejorables de estos sensores y posibles caminos por los que continuar el trabajo demostrado.

Comenzando por la antena WiFi presente en el chip ESP32 utilizado para el proyecto, se tiene que dicha antena trabaja con una frecuencia de 2.4GHz[34](Pág. 16) por lo que algunos teléfonos celulares modernos con antenas de 5GHz no serían captados por el modo promiscuo del ESP32 para el conteo de direcciones MACs. Nuevos modelos de Espressif como el ESP32-C5 y ESP32-C6 vienen equipados con una antena WiFi dual 2.4GHz-5GHz pero pierden el módulo LoRa integrado que sí tienen las placas que se usaron en el proyecto.

Otro posible camino por el cual seguir avanzando es la posibilidad de conectarlos o transformarlos en sensores actuadores con el fin de poder reaccionar ante eventos. Algunos ejemplos de posibles eventos a los cuales reaccionar pueden ser situaciones de aglomeración de personas, mediciones de mala calidad de aire o mediciones de alta temperatura, sin embargo cabe resaltar que la creación de la infraestructura que tenga la capacidad de poder actuar ante estos tipos de eventos dentro de la Facultad de Ingeniería (dispositivos de filtrado del aire, refrigeración o incluso abrir y cerrar ventanas de forma automática) puede ser extremadamente desafiante y costoso.

Finalmente también es necesario poder avanzar en el diagnóstico de los cortes repentinos experimentados en las mediciones.

3.4. Segundo caso de uso: Conteo de estacionamientos con reconocimiento de imágenes

3.4.1. Introducción

El conteo de lugares libres de estacionamiento es una problemática con un amplio historial de estudio principalmente impulsado por la ineficiencia del gasto de combustible y tiempo empleado en los parkings ser recorridos en busca de un lugar de estacionamiento. Gracias al avance de las redes neuronales que permiten llevar a cabo técnicas de reconocimiento y clasificación de imágenes fue que se abrieron nuevas posibilidades para abordar la solución, en ocasiones de forma más barata a los enfoques tradicionales de los que se hablará en las siguientes secciones. En particular para la Facultad de Ingeniería fue propuesta la idea de realizar un análisis

y un prototipo que brinde una primera aproximación a una solución adaptable al campus inteligente.

3.4.2. Motivación y objetivos

Dentro del predio del edificio de la Facultad existen tres parques de estacionamientos y dependiendo del horario es posible que la gran parte de estos se encuentren ocupados, haciendo necesario en estos casos que cualquiera que esté llegando a la Facultad tenga que recorrer detenidamente múltiples áreas antes de hallar un lugar. Es debido a esto que surge la idea de poder contabilizar los lugares libres de las distintas zonas de parking para saber en tiempo real la cantidad de lugares libres hay en cada una de ellas y poder brindar ese dato de forma fácilmente accesible para que las personas que ingresan a Facultad ahorren tiempo y combustible, así como mejorar la experiencia en general.

3.4.3. Implementación

Esta sección será dividida en dos subsecciones, en la primera comenzaremos brindando un repaso general de las distintas formas que actualmente existen para abordar esta tarea y los pros y contras que estas tienen al aplicarlas en nuestra situación. En la segunda parte se presenta una breve descripción de las características actuales de la Facultad en materia de infraestructura junto con un breve análisis de los desafíos que se deben sortear para lograr una solución lo más óptima posible mediante el enfoque utilizado para el proyecto.

3.4.3.1. Técnicas relevadas de conteo de parking

Las alternativas que fueron relevadas para abordar el conteo de espacios de parking consistieron en tres enfoques que se reiteran en múltiples artículos académicos así como en soluciones brindadas en la industria. Dentro de estos tres enfoques se pueden hallar múltiples variantes en distintos aspectos de la solución, formando así un grupo de posibilidades aún mayor, sin embargo en para este análisis se plantean soluciones generales de estos tres paradigmas sin entrar en estas variantes.

La primera alternativa evaluada consiste en el despliegue de un sensor por cada lugar de estacionamiento para detectar la presencia de un automóvil en dicho espacio (Figura 3.25). Esta solución es bastante popular y puede ser encontrada en

Uruguay, por ejemplo en los estacionamientos de los distintos Shoppings de Montevideo como se muestra en la Figura 3.24.

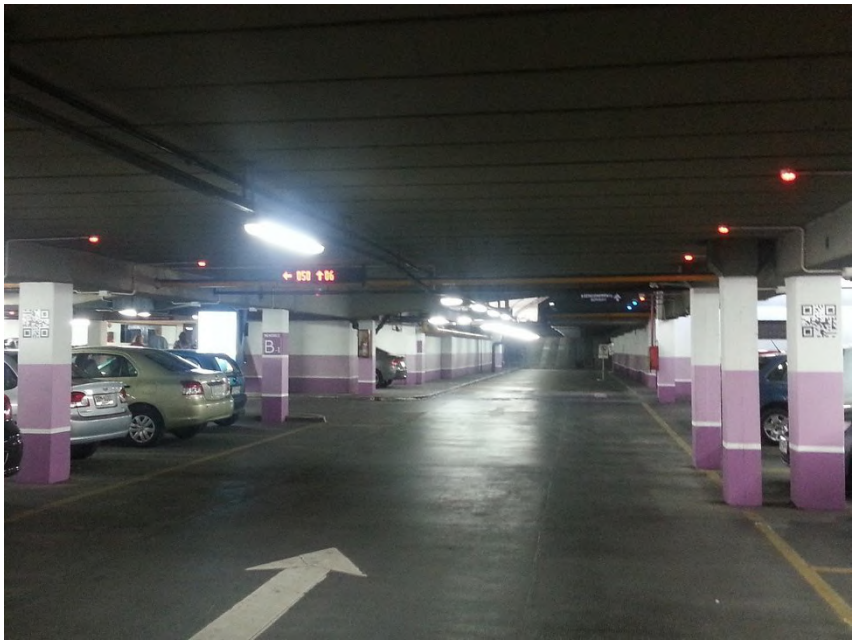

Figura 3.24: ***Fotografía del parking smart del Punta Carretas Shopping.*** Fuente: El Observador, 2012, [65]

Dependiendo del entorno en el cual se realiza el despliegue pueden existir distintas alternativas en cuanto a los sensores y tecnologías a utilizar. En ambientes cerrados por ejemplo, es posible utilizar sensores de ultrasonido que no tienen por qué ser resistentes al clima ni se deben preocupar de que sus mediciones se vean afectadas por la lluvia, mientras que para un parking al aire libre es necesario tener en cuenta estas variables. Por otra parte, en ambientes cerrados se debe hacer un despliegue de una red de cableado que llegue a cada sensor lo cual implica un costo de mano de obra y materiales considerable, en cambio en ambientes abiertos es más fácil el uso de tecnologías inalámbricas para establecer la comunicación con los sensores.

Es claro que existen otras variables a tener en cuenta al momento de diseñar una solución de este tipo pero las características que se mantienen en cada escenario posible es que si bien es una solución precisa y robusta, es inevitablemente costosa, compleja de mantener y muy difícil de emplear en la Facultad.

La segunda alternativa evaluada se lleva a cabo mediante la instalación de sensores de proximidad bidireccionales en las entradas y salidas del estacionamiento. Estos sensores bidireccionales están conformados por dos sensores instalados a un

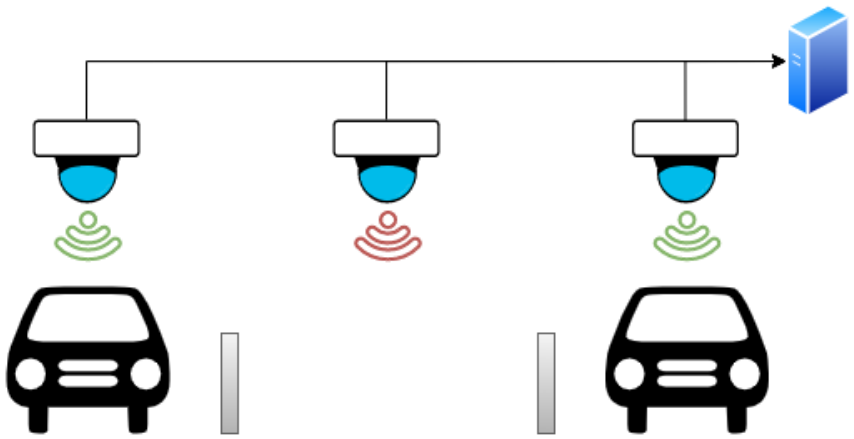

Figura 3.25: ***Esquema de funcionamiento de la primer opción de censado de parking.***

lado de las entradas y salidas con una cierta separación entre ellos de forma tal que siempre uno es activado por el automóvil antes que el otro, logrando de esa forma saber el sentido en el que este está circulando. Con estos sensores sumado al valor inicial de vehículos en el estacionamiento es posible llevar un contador en tiempo real que se mantiene actualizado ante las entradas y salidas de vehículos.

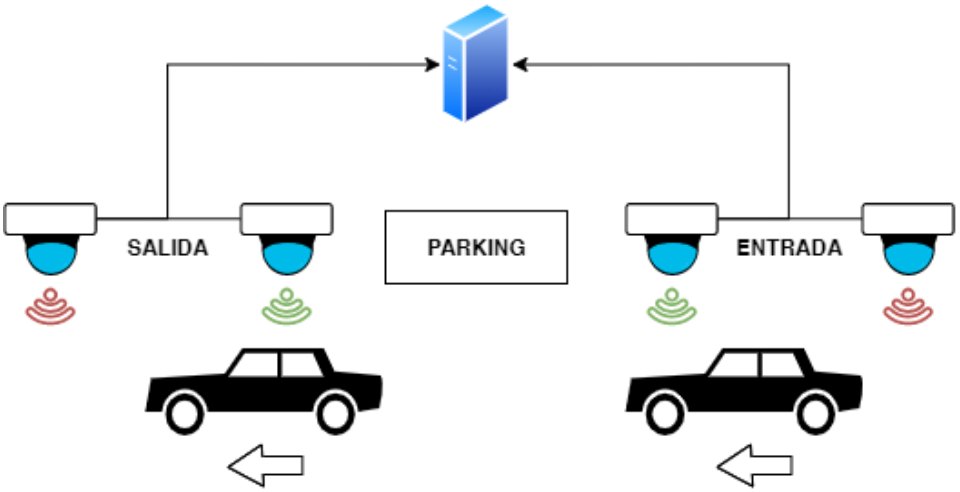

Figura 3.26: ***Esquema de funcionamiento de la segunda opción de censado de parking.***

Este enfoque es efectivamente menos costoso que el anterior tanto en materiales como en mano de obra y costo de mantenimiento, además de que podría llegar a ser posible de implementar en la Facultad teniendo las precauciones necesarias. Una desventaja a notar es que a diferencia del primer enfoque, con esta solución no se tiene información de cuáles son los lugares libres sino que solamente se sabe cual es la cantidad de plazas libres por zona.

El tercer y último enfoque es el finalmente elegido para abordar en este proyecto

y hace un cambio de paradigma respecto a los dos anteriores. En esta solución, en lugar de utilizar sensores físicos para detectar los vehículos se utiliza software, más concretamente se emplean algoritmos de detección de objetos (object detection). Cuando se trata de algoritmos de visión por computador y detección de objetos, el estado del arte son las redes neuronales convolucionales pertenecientes al mundo del deep learning (Zou et al., 2019, [102]). Utilizando estas tecnologías, se plantea una solución que consiste de cámaras de vigilancia o webcams regulares adaptadas para estar a la intemperie y conectadas a una PC que pueda leer la imagen recibida, ejecutar el algoritmo de reconocimiento de imágenes sobre ella y enviar a la plataforma el conteo de la cantidad de lugares libres.

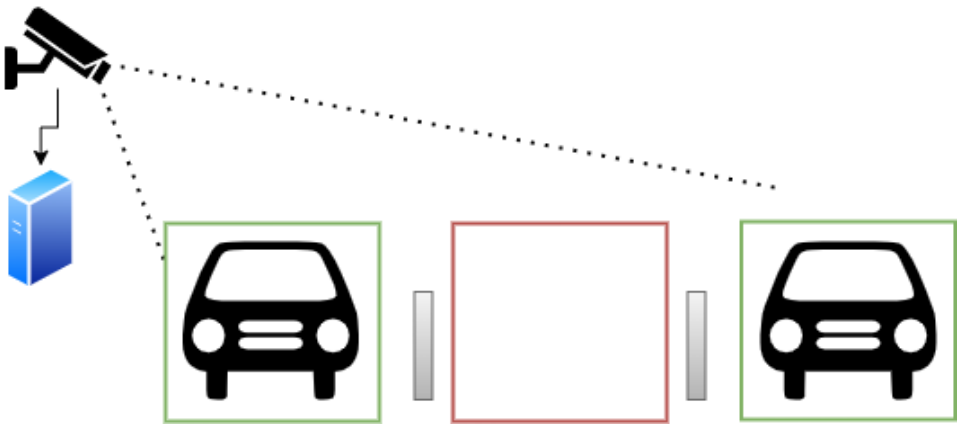

Figura 3.27: ***Esquema de funcionamiento de la tercer opción de censado de parking.***

Esta es la opción más económica de las tres y la más aplicable en la Facultad. Nuevamente comparte la desventaja con la segunda alternativa al no tener la información necesaria para saber que lugares son los que están libres, pero por otro lado tiene como ventaja respecto a las dos anteriores la versatilidad que provee el reconocimiento de imágenes. Con esta solución se tiene, por nombrar un ejemplo, la posibilidad de entrenar o utilizar un modelo que también reconozca personas en las imágenes y utilizar las mismas cámaras de conteo de estacionamiento como cámaras de seguridad durante la noche. Otra ventaja sustancial que veremos nuevamente en la sección de implementación es que el uso de las redes neuronales convolucionales para resolver esta tarea ya está estudiado, por lo que existen guías y bibliotecas específicas para dichas redes así como muchos artículos que tratan la temática obteniendo resultados prometedores.

3.4.3.2. Revisión de los estacionamientos en la Facultad

En la Facultad de Ingeniería existen al momento de realizado el proyecto tres estacionamientos bien diferenciados los cuales se delimitarán por los colores verde, azul y anaranjado según lo marcado en la Figura 3.28. Si bien todos los estacionamientos se hallan al aire libre, se tiene en particular dos de ellos que pueden ser considerados como los más sencillos de abordar para el conteo de plazas libres con reconocimiento de imágenes. Estos son el azul y el anaranjado porque no poseen árboles ni obstrucciones visuales de ningún tipo y pueden ser capturados con una cámara desde distintos ángulos y alturas desde los edificios de la Facultad. Por otra parte se halla el estacionamiento verde el cual tiene múltiples árboles que obstruyen una posible vista desde una altura que abarque toda el área del estacionamiento, en esta zona en particular se requiere de un enfoque distinto que puede resultar en una solución más compleja con múltiples cámaras enfocando hacia distintas direcciones.

Figura 3.28: ***Delimitación de las tres zonas de estacionamiento en la Facultad de Ingeniería.*** Fuente: Perfiles de Facebook de Facultad de Ingeniería y de Posgrados Facultad de Ingeniería - UdelaR

Dentro del marco del proyecto se tomaron múltiples fotografías con distintos puntos de vista desde los edificios de la Facultad utilizando tres tipos de cámaras económicas detalladas en la Subsubsección 3.4.3.6 donde se lista el hardware. Para realizar las pruebas se tomaron fotografías del estacionamiento anaranjado y del verde en distintos horarios para poder sumar al análisis los distintos niveles de luz. El estacionamiento azul no fue fotografiado pese a que por su disposición, forma y la nula obstrucción visual es el mejor para llevar a cabo las pruebas. Este estacionamiento fue descartado por motivos de acceso ya que para llegar al mejor punto de vista ubicado en la cornisa del techo que da hacia este parking, se debe o bien utilizar un elevador o caminar bordeando el perímetro del techo desde el otro lado del

edificio donde se ubica la escalera del balcón del instituto de materiales por donde se accede a él.

En la Figura 3.29 se muestran los cinco puntos de vista tomados por las fotografías usadas para las pruebas, mostrando con el punto rojo el lugar exacto desde donde se tomó la foto.

Figura 3.29: ***Puntos de vista usados para las fotografías en las pruebas de conteo de vehículos.***

3.4.3.3. Consideraciones arquitectónicas

Antes de comenzar a idear una solución y determinar el software de clasificación de imágenes que se utilizará para el conteo de estacionamientos, fue preciso evaluar las características particulares que este tipo de dispositivo de IoT tendrá. Al emplear un algoritmo de visión por computador para resolver y calcular el dato buscado (el número de lugares libres), se requiere también de un poder cómputo mucho mayor al del resto de sensores detallados en este documento. Además de lo ya mencionado, el tamaño de los datos tomados del entorno para poder realizar este cálculo es también mucho mayor a los del resto de sensores puesto que en particular se requiere de un archivo de imagen, mientras que el resto de sensores captan magnitudes físicas del ambiente que se representan computacionalmente con tipos de datos más elementales como strings o enteros.

A partir de estas dos particularidades es que se considera oportuno comparar dos posibles escenarios aplicables al campus: uno de ellos usando el paradigma de computación centralizada o *centralized computing* y otro bajo el paradigma de la computación de frontera o *edge computing*.

En el caso de la computación centralizada (que llamaremos *caso uno*) se plantea el dispositivo como un sensor con el menor poder de procesamiento posible, es

decir una cámara con capacidad de transmitir información hacia el servidor del campus. En este caso el sensor tendría la responsabilidad de tomar la fotografía del estacionamiento y enviarla hacia el servidor por la red WiFi. El servidor al recibir la imagen o el conjunto de imágenes tendrá la tarea de ejecutar el algoritmo de clasificación y guardar en la plataforma el dato obtenido.

En el caso de la computación de frontera (que llamaremos *caso dos*) se plantea un dispositivo sensor con un poder de cómputo capaz de ejecutar algún algoritmo reducido y optimizado de clasificación de imágenes. En este escenario el sensor tendría la responsabilidad de tomar la fotografía como en el caso anterior pero también de procesarla para obtener el dato de espacios de estacionamiento libres, seguido de esto su último paso es enviar ese dato obtenido por medio de alguna tecnología de Internet de las Cosas hacia el servidor del campus el cual tiene la única tarea de recibirlo y almacenarlo.

Teniendo en cuenta el funcionamiento general de ambas alternativas, procedemos a realizar una comparativa viendo las ventajas y desventajas de cada una en la siguiente serie de variables y requerimientos no funcionales:

- **Costo**: Para este requerimiento se evalúan los costos económicos de los sensores y del servidor para los dos casos. Con respecto a los sensores, por las características que posee el caso uno, es posible usar sensores mucho más económicos que en el caso dos al casi no requerir de poder de cómputo, particularmente en el caso dos cada sensor deberá disponer de al menos una computadora monoplaca o algún hardware equiparable.

  Por otra parte, cuando se trata del costo del servidor es el caso dos el que permite abordarse con un menor costo como mínimo en el largo plazo si la solución escala en número de sensores. Si bien es cierto que la ejecución de una red neuronal para clasificación de imágenes no requiere particularmente de mucho poder de cómputo una vez que esta tiene su modelo entrenado, sí puede llegar a presentarse como un problema en el caso de tener un gran número de sensores desplegados enviando fotografías de forma continua. En nuestro escenario particular no se necesitaría de una cantidad significativa de sensores para contar los estacionamientos ya que la Facultad posee solamente tres, pero sí podría llegar a necesitarlo ante una posible nueva propuesta que también utilice clasificación de imágenes pero aplicada a otra problemática en la que se despliegue una mayor cantidad de sensores.

- **Escalabilidad**: En este apartado ambos enfoques son posibles de escalar horizontalmente pero el caso dos a diferencia del uno nunca va a necesitar de un escalado de recursos de servidor.

- **Flexibilidad**: En este requerimiento se toman en cuenta por una parte la flexibilidad de protocolos de comunicación posibles para cada caso y por otra parte las implicancias de ofrecer flexibilidad en cuanto a la variedad de algoritmos de procesamiento de imágenes. Sobre el primer punto nombrado, se puede ver que como en el caso dos las imágenes son procesadas en el dispositivo sensor y lo que se termina transmitiendo es solamente un dato numérico, es posible utilizar cualquier tecnología de transmisión inalámbricas de bajo ancho de banda (como por ejemplo LoRa), lo que termina dando más versatilidad y flexibilidad a posibles cambios en la solución.

  En el caso uno al tener que estar transmitiendo imágenes, el uso de estas tecnologías de IoT se vuelven prácticamente inutilizables. Sobre el segundo punto de comparación tenemos que, de ser necesario en el caso dos se pueden instalar distintos algoritmos de clasificación de imágenes de una manera sencilla y directa para cada dispositivo. Esta situación podría llegar a darse por ejemplo ante la necesidad de contar estacionamientos en un parking que por su posición geográfica se ve afectado por las proyecciones de sombras que generan los edificios contiguos, en dicho caso puede ser deseado el uso de un modelo que haya sido entrenado para performar mejor en estas circunstancias mientras que el resto de dispositivos usa el algoritmo estándar de clasificación que haya sido seleccionado. Si bien esta situación también es posible de resolver en el caso uno mediante el despliegue de dos aplicaciones que funcionen de manera paralela con los dos modelos de clasificación, nuevamente se puede llegar a generar un overhead en el servidor si se desea escalar en el número de modelos.

- **Mantenimiento**: Finalmente en el apartado de mantenimiento el caso uno es el claro ganador ya que los dispositivos son más sencillos, económicos y fáciles de cambiar ante un fallo. Otra ventaja sustancial que este tiene sobre el caso dos es que las actualizaciones o modificaciones del algoritmo de clasificación son mucho más sencillas de aplicar, teniendo que trabajar en un único

punto en el servidor. Si bien estos dispositivos pueden ser configurados para recibir conexiones SSH entrantes que faciliten hacer estos cambios de manera remota, considerando que dependiendo de la solución pueden llegar a ser muchos, el mantenimiento podría dificultarse.

Se puede concluir que no hay un claro ganador, como siempre es necesario considerar las características de nuestro escenario para poder ponderar y decidir el más conveniente. Para el prototipo a presentar a continuación se optó por un acercamiento enfocado en el edge computing, cabe destacar que no se está afirmando que sea la mejor solución, sino que simplemente se optó por ella a efectos de crear un prototipo más llamativo para presentar en el proyecto.

3.4.3.4. Algoritmo de clasificación utilizado para el prototipo

Habiendo definido las características arquitectónicas del sensor que se desea implementar, el siguiente paso fue la elección de la red neuronal y el algoritmo a utilizar, el cual era deseable que en particular fuese capaz de ser ejecutado por una Raspberry Pi 4. Dentro del área de la visión por computador y la clasificación de imagenes utilizando inteligencia artificial existen varias redes neuronales creadas para ser utilizadas en edge computing, algunas de las más conocidas son SSD MobileNet (Liu et al., 2016, [58]), RetinaNet (Lin et al., 2018, [56]) o YOLO (Redmon et al., 2015, [76]). Cada una de estas redes neuronales dispone de múltiples versiones, además utilizan sus propias bibliotecas y frameworks y están a un nivel equiparable en cuanto a precisión promedio o AP. Sin embargo para este proyecto se optó por utilizar YOLO en su versión V3(Redmon y Farhadi, 2018, [75]) y V7(Wang y Hong-Yuan Mark Liao, 2022, [100]) para poner a prueba su precisión, ligereza y velocidad al ejecutarse en dispositivos frontera. En particular también se tomaron en cuenta los resultados obtenidos en (Ding y Yang, 2019, [28]) en donde mediante la utilización de YOLOV3 con un modelo entrenado a medida para el conteo de autos y estacionamientos vacíos se logró una precisión mayor al 90 %.

Es muy importante aclarar que en el marco de este proyecto se utilizaron los pesos predeterminados que se proveen al instalar YOLO y no se entrenaron modelos personalizados para la detección de espacios libres de estacionamiento. Es por esto que solamente se hacen detecciones de los vehículos estacionados y se realiza el conteo de los mismos para comprobar la viabilidad y precisión del sensor en su etapa más elemental. En el caso de un despliegue real de un dispositivo como este, el proceso sería más complejo e involucraría el uso de un modelo entrenado espe-

cíficamente para detectar y contar los lugares libres o alguna solución adecuada a la naturaleza de los estacionamientos de la Facultad. Teniendo en cuenta que muchos de estos estacionamientos de Facultad no están señalizados y tampoco tienen pintadas líneas divisorias, es preciso idear alguna solución alternativa. Un posible acercamiento podría ser algo similar a (Yusnita, Norbaya y Basharuddin, 2012, [101]) en donde la detección de lugares libres se hace gracias a círculos pintados en los estacionamientos que sean detectables por la cámara. Otra opción sería seguir realizando el conteo de vehículos y contrastar ese dato con un valor fijo calculado manualmente a partir de la capacidad máxima de estacionamientos en el encuadre de la cámara del dispositivo, este acercamiento está lejos de ser ideal puesto que al no haber señalización en los espacios, puede ocurrir que las personas estacionen con distintos espaciados entre sí, haciendo que el valor máximo de lugares no siempre sea el mismo.

YOLO (por sus siglas en inglés *You Only Look Once*) es una algoritmo de detección de objetos que hace uso de una única red neuronal neuronal convolucional. Como se indica en su nombre este algoritmo evalúa las imágenes una sola vez a diferencia de otros algoritmos que por sus características deben iterar sobre una imagen múltiples veces usando distintas ventanas de detección, por ende su gran velocidad en comparación a estos.

Su funcionamiento explicado de forma simple consiste en comenzar dividiendo la imagen en una grilla de *sxs* celdas, luego en cada división se aplica una serie de *b* rectángulos prefijados llamados bounding boxes a los que les ejecutará un algoritmo de clasificación. Estos bounding boxes vienen dados a partir del dataset que haya sido usado para entrenar el modelo y en particular están obligados a tener su punto centro dentro de la celda a la que pertenecen, sin embargo sus distintos tamaños y relaciones de aspecto pueden sobresalir de su celda permitiendo que sea posible que abarquen múltiples celdas contiguas.

A partir de cada bounding box se genera un vector de clasificación que indica sus coordenadas del punto centro y sus dimensiones así como la probabilidad de que tenga en su interior alguna de las clases de objetos buscadas. Luego de clasificar todas las bounding boxes se dejan solamente las que tengan el mayor índice de confianza según lo que el modelo dio en la clasificación y se eliminan las bounding boxes superpuestas usando una métrica llamada IoU (intersection over unión), dando así las bounding boxes finales con su etiqueta y score de confianza. La posibilidad de paralelizar todos estos cálculos descritos usando una GPU dedicada permite obtener desempeños mucho mayores, sobre todo en aplicaciones de análi-

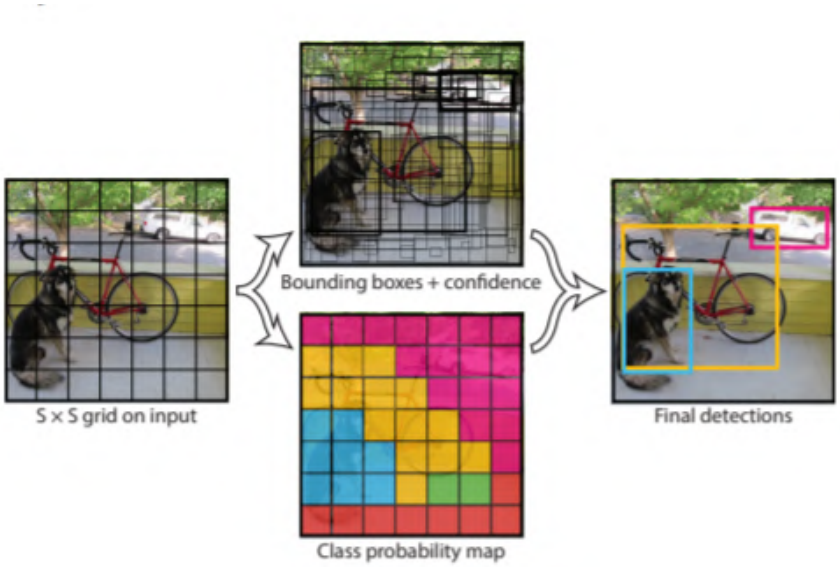

Figura 3.30: ***Esquema del funcionamiento del algoritmo YOLO.*** Fuente: Redmon et al., 2015, [76] (Fig. 2).

sis de video en tiempo real. Como todos estos tipos de software, YOLO utiliza para funcionar un framework open source para redes neuronales llamado Darknet[74].

Para ver la explicación de forma rigurosa se recomienda referirse a sus papers [76], [75] y [100], así como a recursos online [13].

3.4.3.5. Lógica del dispositivo sensor

Para lo que sería el prototipo llevado a la realidad, se desarrolló un script en Node.js que permite automatizar la toma de las fotografías, el uso de YOLO para clasificarlas y posteriormente el envío a la plataforma ThingsBoard. Mediante el uso de la biblioteca *node-schedule* se setea una ejecución periódica mediante una expresión con formato Cron, por ejemplo cada treinta minutos. Al realizarse dicha ejecución se llama el comando bash *fswebcam* mediante la función *exec* de la biblioteca *node:child_process* para tomar la foto. Una vez guardada la foto se vuelve a ejecutar un *exec* para invocar a YOLO junto con una función de callback que al finalizar la clasificación lee el archivo de texto generado por la misma usando los métodos brindados por la biblioteca *fs* de Node.js. Al leer el archivo de texto se cuentan las apariciones de las palabras *car* y *truck* para YOLOV3 o las entradas con código 2 para YOLOV7. Para finalizar, se envía ese resultado a la plataforma por medio de un método HTTP POST usando la biblioteca *axios*. Alternativamente es posible setear la variable threshold para considerar solamente los resultados que

tengan una confianza mayor a un valor mínimo.

3.4.3.6. Hardware utilizado

Raspberry Pi: Para el módulo de cómputo del sensor se utilizó una Raspberry Pi 4 Model B de 8GB con el sistema operativo Raspberry Pi OS (64-bits) (release 2023-02-21). Alternativamente también se realizaron pruebas secundarias de ejecución de los algoritmos en una Raspberry Pi 3B de 1GB con el sistema operativo Raspberry Pi OS (32-bits) (release 2023-02-21), un modelo más antiguo de esta línea de computadores.

Cámaras utilizadas: Para tomar las fotografías se utilizaron tres cámaras distintas con el fin de probar el algoritmo de clasificación de imágenes con fotografías de distintas calidades y campos de visión. Las cámaras son:

- **Logitech C270:** Webcam de gama de entrada con calidad 720p y campo visual de 55°.
- **Webcam HD:** Webcam de marca genérica con calidad 1080p y campo visual de 96°.
- **Cámara de smartphone:** Cámara trasera del smartphone Pocophone F1 con calidad 2160p.

3.4.4. Resultados obtenidos

En esta sección se presentan los resultados obtenidos de los distintos algoritmos usados juntos con los tiempos de ejecución de cada prueba y las métricas de *accuracy* y *recall* calculadas. Se tomaron en consideración para la prueba diez fotografías (seis de la zona anaranjada y cuatro de la zona verde) las cuales se muestran a continuación en la Figura 3.31, se recomienda usar zoom para verlas.

Como se puede observar en la Figura 3.31 hay imágenes tomadas con las tres cámaras, con distintos niveles de luz tanto de día como en la tarde/noche así como distintos ángulos de inclinación de la cámara respecto al plano horizontal y distancias hasta los vehículos.

Para presentar los resultados se muestra en la Tabla 3.2 y la Tabla 3.3 los resultados de YOLOV3 y YOLOV7 respectivamente para las diez imágenes usando distintos tamaños de procesamiento en el modelo, para YOLOV3 se usaron tres

Figura 3.31: ***Fotografías usadas para realizar las pruebas del algoritmo de reconocimiento de objetos YOLO.*** Nota: Se recomienda hacer zoom.

tamaños (224px, 416px y 608px) y para YOLOV7 dos (640px y 1280px). Las imágenes de la tabla están numeradas en el orden presentado en la Figura 3.31 y se les agregó el color haciendo referencia a la zona de estacionamiento a la que pertenecen y un icono indicando el horario de día y noche.

Antes de pasar a las tablas se brinda la definición de las métricas *Precision* y *Recall* usadas junto con la descripción de las variables que conforman su cálculo. Comenzando con las variables usadas para calcular las métricas tenemos:

- **T (Truth):** Esta variable contiene el valor real de vehículos en la imagen contados de forma manual por un humano. No se usa en las fórmulas de las métricas sino que se toma como el valor verdadero de referencia en las tablas.
- **TP (True Positive):** Esta variable contiene el valor de la cantidad de predicciones correctas hechas por el modelo, es decir que cuenta las detecciones de automóviles cuando efectivamente lo son.
- **FP (False Positive):** Esta variable contiene el valor de la cantidad de predicciones incorrectas hechas por el modelo, es decir que cuenta las detecciones de automóviles en lugares donde no los hay.
- **FN (False Negative):** Esta variable contiene el valor de la cantidad de predicciones omitidas por el modelo, es decir que cuenta los automóviles no detectados por el modelo en la imagen.

Se aclara como observación que para las variables descritas anteriormente solo se contabilizaron los autos dentro de los estacionamientos de la Facultad, tanto para

		YOLOV3														
		224px					416px					608px				
Img	T	TP	FP	FN	P	R	TP	FP	FN	P	R	TP	FP	FN	P	R
1 ☼	20	18	0	2	100 %	90 %	20	0	0	100 %	100 %	20	1	0	95 %	100 %
2 ☼	21	15	0	6	100 %	71 %	21	1	0	95 %	100 %	21	1	0	95 %	100 %
3 ☾	18	14	2	4	86 %	78 %	17	0	1	100 %	94 %	17	1	0	94 %	100 %
4 ☼	14	11	2	3	85 %	79 %	10	1	4	91 %	71 %	11	1	3	92 %	79 %
5 ☼	26	23	6	3	79 %	88 %	24	5	2	83 %	92 %	24	1	2	96 %	92 %
6 ☼	26	17	3	9	85 %	65 %	23	1	3	96 %	88 %	24	0	3	100 %	89 %
7 ☼	53	23	0	30	100 %	43 %	31	1	22	97 %	58 %	27	0	26	100 %	51 %
8 ☼	30	26	5	4	84 %	87 %	26	2	4	93 %	87 %	27	3	3	90 %	90 %
9 ☾	27	22	7	5	76 %	81 %	23	4	4	85 %	85 %	23	2	4	92 %	85 %
10 ☾	20	16	0	4	100 %	80 %	12	3	8	80 %	60 %	11	1	9	92 %	55 %

Tabla 3.2: Resultados obtenidos en cada fotografía con YOLOV3 junto con su Accuracy (P) y Recall (R).

el valor de referencia *Truth* como para las clasificaciones de dichos autos que determinan *TP*, *FP* y *FN*. Esto quiere decir que en las diez fotos se descartan de todo conteo los autos que se encuentran pasando por la calle o estacionados fuera de los límites del estacionamiento. Luego de calcular estos valores de forma manual en cada imagen se procede finalmente a realizar el cálculo de las dos métricas utilizadas con la fórmula que se presenta a continuación:

- **Precisión (P):** Es la proporción de True Positives (aciertos) respecto al total de predicciones hechas para autos. Es decir que si en una imagen con diez autos reales y de todas las predicciones con la etiqueta *auto* había nueve que eran efectivamente autos, la Precisión será de 90 %.
  Su fórmula es: $P = \frac{TP}{TP+FP}$
- **Recall (R):** Es la proporción de True Positives (aciertos) respecto al total de automóviles verdadero. Es decir que si en una imagen con diez autos el modelo encuentra y clasifica ocho y deja dos sin clasificar, el Recall será de 80 %.
  Su fórmula es: $R = \frac{TP}{TP+FN}$

En la Tabla 3.4 se observa un resumen de los resultados finales de las pruebas, concluyendo que el algoritmo YOLOV7 con imágenes de 1280px es la mejor opción al buscar exactitud, obteniendo una precisión promedio mayor al 97 % y recall promedio mayor al 90 %.

Otro aspecto muy importante que se puede concluir gracias a los resultados obtenidos en la clasificación de las fotografías 1, 2, 3, 4 y 7 es que el ángulo y altura

		YOLOV7									
		640px					1280px				
Img	T	TP	FP	FN	P	R	TP	FP	FN	P	R
1 ☼	20	20	0	0	100 %	100 %	20	1	0	95 %	100 %
2 ☼	21	20	1	1	95 %	95 %	21	0	0	100 %	100 %
3 ☾	18	18	0	0	100 %	100 %	18	0	0	100 %	100 %
4 ☼	14	14	1	0	93 %	100 %	14	1	0	93 %	100 %
5 ☼	26	21	0	5	100 %	81 %	23	1	3	95 %	86 %
6 ☼	26	16	0	10	100 %	62 %	22	1	4	96 %	85 %
7 ☼	53	16	0	37	100 %	30 %	27	0	26	100 %	51 %
8 ☼	30	25	3	5	89 %	83 %	28	0	2	100 %	93 %
9 ☾	27	23	0	4	100 %	85 %	26	0	1	100 %	96 %
10 ☾	20	17	0	3	100 %	85 %	18	1	2	95 %	90 %

Tabla 3.3: Resultados obtenidos en cada fotografía con YOLOV7 junto con su Accuracy (P) y Recall (R).

YOLOV3	Avg P	Avg R
224px	89,6 %	76,3 %
416px	92,0 %	83,7 %
608px	94,6 %	84,1 %

YOLOV7	Avg P	Avg R
680px	97,8 %	82,1 %
1280px	97,4 %	90,3 %

Tabla 3.4: Promedios de Accuracy (P) y Recall (R) de todos los modelos y dimensiones probados.

de la cámara es uno de los factores más influyentes en obtener una buena clasificación. En las fotografías 1, 2, 3 y 4 en donde la cámara se encuentra en una posición elevada respecto al parking, el modelo reconoció y clasificó con una precisión casi total. Mientras tanto en la figura 7 en donde el ángulo que forma la cámara con el plano horizontales es menor, se obtuvo la peor clasificación perdiendo en el mejor de los casos hasta la mitad de los vehículos. En la Figura 3.32 se ejemplifica como se entienden que tienen que estar posicionadas las cámaras para obtener un resultado óptimo.

La última conclusión obtenida a partir de las pruebas realizadas es que si bien la generación de un modelo entrenado específicamente para el reconocimiento de espacios de estacionamientos y autos puede llegar a obtener una mejor precisión, dado los buenos resultados obtenidos con el modelo predeterminado podría no ser necesario si las cámaras se disponen en los lugares correctos.

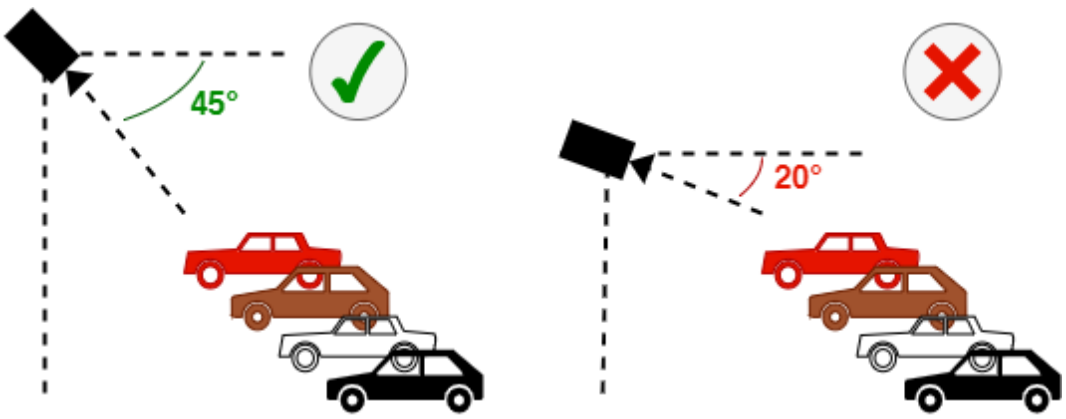

Figura 3.32: ***Esquema de recomendación del ángulo de posicionamiento de cámaras para reconocimiento de imágenes.***

Finalmente se dejan en la Figura 3.33 las imágenes clasificadas por YOLOV7 (1280px) para tener una referencia visual de las salidas que genera el algoritmo.

Pasando ahora a los tiempos de ejecución, en la Tabla 3.5 se presentan los resultados obtenidos de todas las variantes de ambos algoritmos de clasificación probados en la Raspberry Pi 4 Model B 8GB. Por otra parte en la Raspberry Pi 3B 1GB, solo YOLOV3 en tamaño 224px y 416px funcionaron correctamente. YOLOV3 en su versión de 608px experimenta un crash por falta de memoria y YOLOV7 no pudo ser ejecutado por problemas con Python y sus dependencias en el sistema operativo de 32 bits de la Raspberry.

Figura 3.33: ***Fotografías de las salidas obtenidas de la clasificación de YOLOV7 (1280px).*** Nota: Se recomienda hacer zoom.

	YOLOV3			YOLOV7	
Tamaño	224px	416px	608px	640px	1280px
Tiempo	12s	31s	1m 13s	53s	2m 30s

Tabla 3.5: ***Resultados de tiempo aproximados obtenidos de las ejecuciones del modelo en la Raspberry Pi 4 Model B.***

3.4.5. Mejoras sugeridas y trabajo futuro

Este prototipo desarrollado puede ser considerado como un *minimum viable product* que tiene un gran espacio para crecer. Recordando que las detecciones se realizan sobre los vehículos en las imágenes y que los procedimientos presentados para obtener el dato final de estacionamientos libres son rudimentarios y poco precisos, puede ser un buen camino intentar realizar un conteo real de espacios libres de estacionamiento, algo que puede llegar a involucrar trabajos de obra sobre algunos estacionamientos de la Facultad pero que podrían llegar a lograrse con relativo poco coste.

Sumado a esto, puede ser deseable realizar pruebas con un modelo entrenado a la medida con un dataset específico de espacios de estacionamientos o incluso con un dataset que tenga fotografías etiquetadas del estacionamiento de la Facultad. Nuevamente recordamos que la obtención y preparación de estos datasets conlleva un tiempo de trabajo considerable puesto que para proveer al modelo con datos de entrada de calidad, se deben definir manualmente las *bounding box* de cada objeto en la imagen.

Alternativamente, este mismo dispositivo puede ser usado para otros propósitos de conteo como por ejemplo el conteo de personas en salones o espacios públicos de la Facultad con el fin de estudiar aglomeraciones o tránsito de personas. Es importante aclarar que tanto el caso de uso llevado a cabo en este proyecto como la idea del conteo de personas pueden llegar a requerir esfuerzos adicionales en ciberseguridad y privacidad de los datos por el hecho de que, aunque las imágenes sean descartadas durante la ejecución estas suelen tener rostros y matrículas en ella.

3.5. Tercer caso de uso: Estación meteorológica

3.5.1. Introducción

Durante la fase de relevamiento de ideas de sensores para el campus inteligente se planteó la de una estación meteorológica para el censado del clima en las inmediaciones de la Facultad. Las estaciones meteorológicas permiten obtener múltiples variables de utilidad, es por esto que se encuentran muchas de ellas a lo largo del país.

3.5.2. Motivación y objetivos

El objetivo fue poder desarrollar e instalar una estación meteorológica en la Facultad con el fin de poder recolectar datos meteorológicos que puedan ayudar a potenciales nuevas investigaciones dentro de la misma así como proveer de esta información a los estudiantes y funcionarios que asisten a ella, permitiéndoles saber con mayor exactitud el clima en la Facultad y las inmediaciones.

3.5.3. Implementación

3.5.3.1. Hardware utilizado

Para implementar este caso de uso fue utilizado un kit de estación meteorológica Sparkfun[89] con anemómetro, veleta para la dirección de viento y pluviómetro (Figura 3.34), el cual requirió de distintas adaptaciones para poder ser utilizado con los chips ESP32. En primer lugar, los sensores de este kit poseen conectores RJ11 para transmitir los datos al controlador por lo que en principio no se disponía de ningún dispositivo capaz de leer los datos de esta estación. Para poder solucionar este problema de integración se utilizó un shield llamado *SparkFun Photon Weather*

Shield (Figura 3.35) el cual posee las fichas RJ11 necesarias para leer los datos de los sensores y traducirlos a impulsos accesibles por pines GPIO y ADC de nuestro controlador. Además de eso el shield posee sensores integrados de temperatura, humedad y presión atmosférica.

Figura 3.34: ***Fotografía del kit de estación meteorológica utilizado.*** Fuente: sparkfun.com [89]

Sin embargo como su nombre indica, este shield viene preparado para otra placa de desarrollo llamada Particle Photon[85] la cual encastra con sus pines directo sobre el shield. Para adaptar los pines de la misma ESP32[84] utilizada en los casos de uso anteriores al shield fue necesario diseñar una nueva placa de pertinax donde se pudieran soldar los pines de la ESP32 a pistas conectadas a los pines utilizados del shield, soldados en las perforaciones que el mismo trae a cada lado de las dos filas de conectores Dupont negros. En la Figura 3.36 se presenta el esquema con estas conexiones luego de la adaptación de pines para la placa ESP32.

Como se puede observar en esta imagen, se hace un adelanto al mostrar que también hay un sensor agregado en la parte izquierda de la misma. Este sensor es el Sparkfun *VEML6075* y permite medir el índice de intensidad de radiación ultravioleta (o índice UV) realizando cálculos sobre las mediciones de radiación UV de tipo A y B. Este sensor fue agregado posteriormente para poder obtener este dato que se consideraba de valor, en próximas secciones se describen las pruebas realizadas para su calibración junto con los problemas que surgieron en su instalación y los

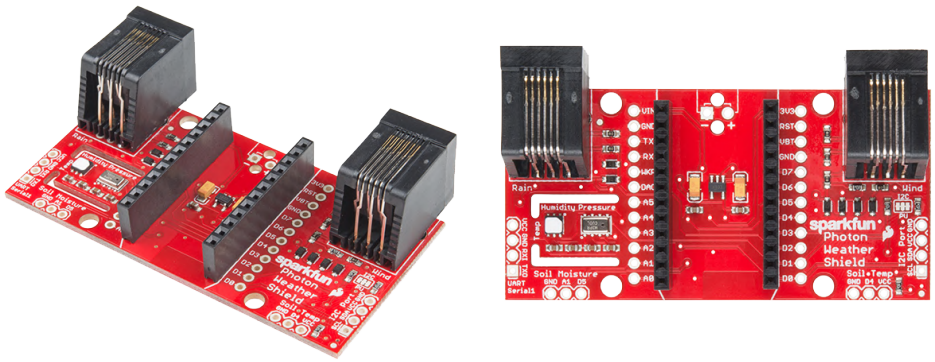

Figura 3.35: ***Fotografías del Photon Weather Shield usado para integrar el kit meteorológico.*** Fuente: sparkfun.com [86]

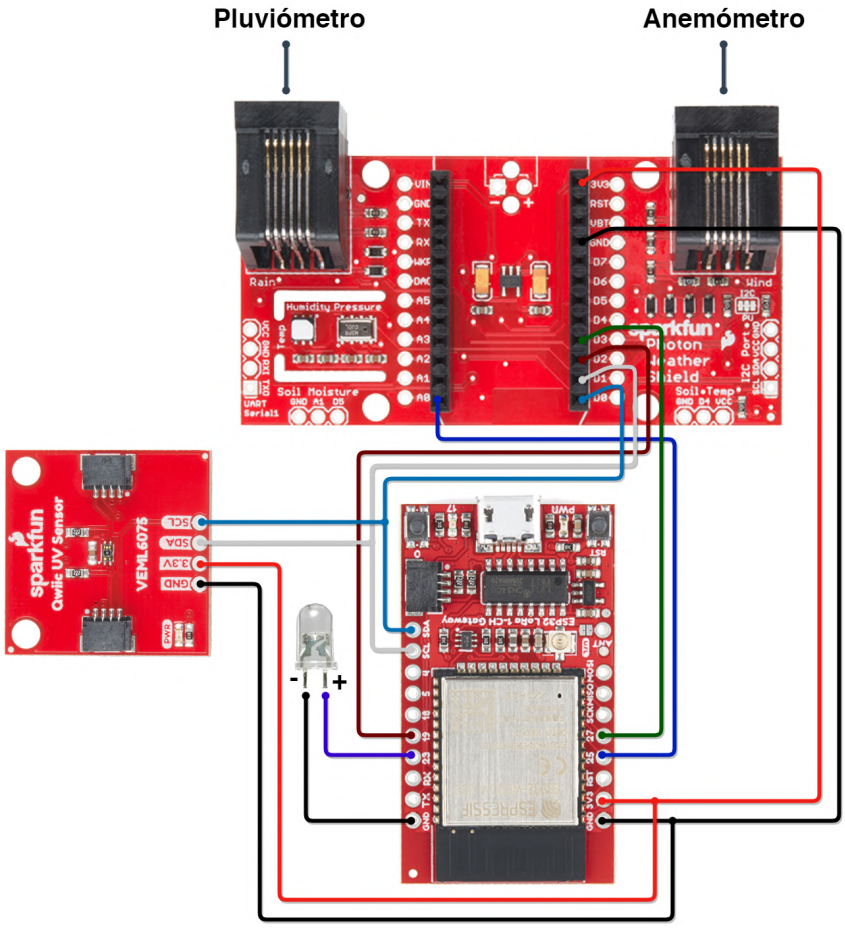

Figura 3.36: ***Diagrama de conexiones de la placa central de la estación meteorológica.***

resultados obtenidos.

3.5.3.2. Fabricación de un nuevo armazón para la estación meteorológica

Un aspecto importante en la creación de este sensor fue la fabricación de un nuevo armazón central para sostener a todos los sensores del kit (anemómetro, pluviómetro, placa central, etc) ya que al armarlo por primera vez, se pudo observar en conjunto con los tutores que si bien la calidad de fabricación del armazón era correcta, por una parte no parecía ser lo suficientemente robusta para soportar las inclemencias del clima durante un periodo prolongado de tiempo y por otra se observó que los cables que interconectan todos los sensores iban expuestos por fuera del armazón. Esto dio indicio de que si se deseaba conservar estos sensores funcionando durante un mayor período de tiempo y alargar lo más posible su vida útil era necesario replantear este soporte para poder resguardar mejor las partes críticas.

Este nuevo armazón (Figura 3.37) fue fabricado con caños de hierro cuadrado de una pulgada, soldado en los brazos inferiores donde se apoyan el pluviómetro y el contenedor de la placa central y atornillado en el travesaño superior donde se encuentran los medidores de viento. Este último soporte fue atornillado y no soldado porque el nuevo armazón fue diseñado para permitir remover todos los sensores del mismo en caso de ser necesario. El armazón tiene puntos de montaje para poder ser fijado a otra estructura utilizando bulones y sumado a esto se fabricó también una base sólida de hormigón para poder trasladarlo e instalarlo de forma inmediata apoyándolo en el piso en cualquier lugar que sea necesario su uso. Además del armazón se pintaron los sensores de lluvia y viento que van expuestos a la intemperie con la finalidad de brindarles protección extra contra el sol y alargar su vida útil.

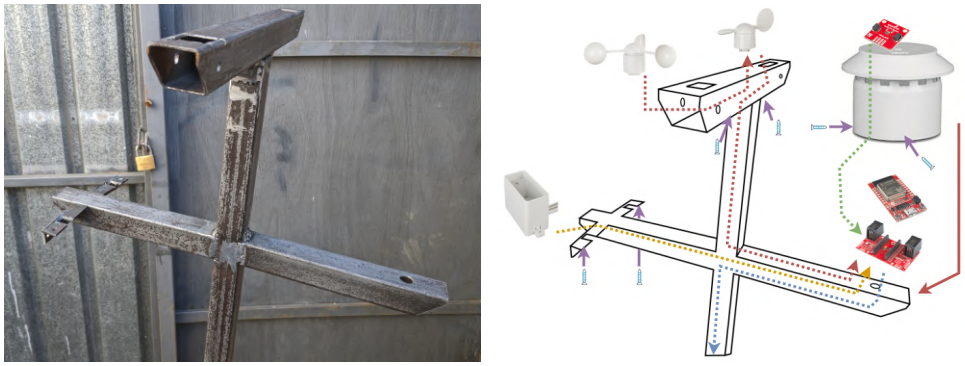

Figura 3.37: ***Izquierda: Fotografía del armazón de la estación meteorológica durante su construcción. Derecha: Esquema interior de los cables en el armazón.***

Figura 3.38: ***Fotografía frontal de la estación meteorológica en su base.***

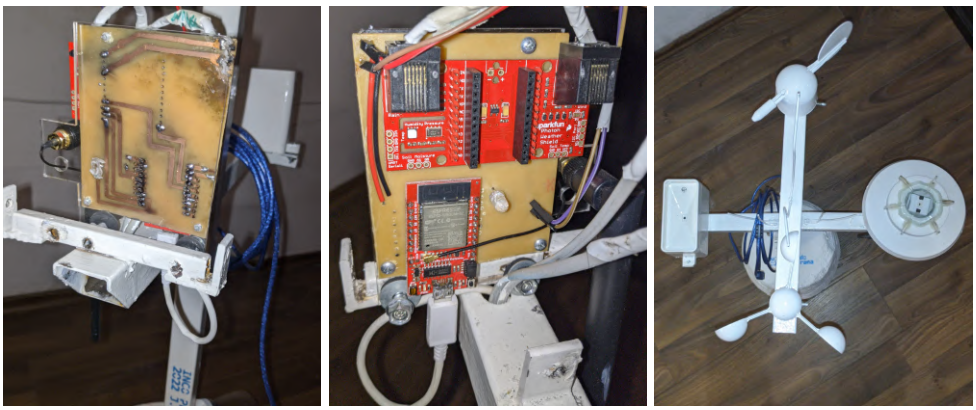

Figura 3.39: ***Fotografías de la placa de la estación meteorológica y vista desde arriba.***

Respecto al sensor UV, este fue instalado en la parte superior del contenedor de la placa central utilizando originalmente un techo fabricado en acrílico de 3mm

	Índice UV			
	Dynamic normal		Dynamic high	
	200ms	800ms	200ms	800ms
Sin tapa	3	-27	7	12
Tapa acrilico 3mm	0	-25	-1	1
Tapa vidrio 3mm	1	2	6	7

Tabla 3.6: Pruebas realizadas en la calibración del sensor VEML6075.

con el fin de dejarlo protegido de la lluvia a la vez que expuesto a la luz solar directa. Sin embargo durante las pruebas realizadas con el sensor se pudo observar que el acrílico filtraba de forma radical la radiación UV haciendo que el índice en ocasiones se volviera cercano a cero al estar midiendo a través de este material. Se investigaron formas de poder aplicar una función inversa que pudiese restablecer los valores originales sabiendo cual es la proporción de radiación filtrada por el material pero no se tuvo éxito ya que las mediciones solían tener comportamientos erráticos. Es por esto que finalmente se optó por fabricar una nueva tapa pero de vidrio de 3mm. Al volver a realizar las pruebas de medición se pudo observar que el vidrio filtra de manera consistente un solo punto del índice UV.

Calibración del sensor UV VEML6075: A continuación se presentan las pruebas de calibración realizadas para el sensor UV. La biblioteca provista por el fabricante posee múltiples formas de calcular el índice mediante la configuración de dos variables de medición. La primera variable es *integration time* el cual define cuantos milisegundos de medición se realizan, los valores posibles son 50ms, 100ms, 200ms, 400ms y 800ms. La segunda variable es *high dynamic* que define si se activa el modo normal o el high dynamic, que citando al fabricante, incrementa la resolución en un factor de 2. En la Tabla 3.6 se muestran los resultados obtenidos de medir con algunas combinaciones de valores posibles.

La medición fue realizada el día 15 de diciembre a las 16:15hs, el clima estaba nuboso pero entre intervalos se encontraba despejado y se obtenía luz solar directa, fue en dichos momentos en los que se realizaron las mediciones. Para contrastar los datos censados se obtuvieron índices UV de distintos sitios de internet para Montevideo, en ese momento el índice UV se estima que estaba entre 6 y 8 puesto que en general estos sitios web no dan un índice en tiempo real sino que, o bien dan un único valor para todo día o dan valores estimados para una lista de horarios en el día. Como se observa en la Tabla 3.6, la configuración que mejor resultados dió fue

el seteo del *integration time* en 200ms y el *dynamic* en high.

3.5.3.3. Software extra utilizado

Bibliotecas de los sensores: Sparkfun provee bibliotecas escritas en C para poder utilizar los sensores de forma directa, estas se encuentran en la página web del sensor sobre sección de documentación del sensor, las bibliotecas usadas son:

- Biblioteca del sensor **VEML6075**: https://github.com/sparkfun/SparkFun_VEML6075_Arduino_Library
- Biblioteca del **SparkFun Photon Weather Shield**: https://github.com/sparkfun/SparkFun_Photon_Weather_Shield_Particle_Library

3.5.3.4. Datos generados por el sensor

En esta sección se detallan todos los datos que se generan en cada envío hacia la plataforma. En general se pueden identificar cuatro datos correspondientes a cada una de las variables censadas: *avg*, *last*, *max* y *min*. *Avg* corresponde al valor promedio calculado de todas las mediciones realizadas de manera continua entre envío y envío. *Last* corresponde al último valor medido antes de realizar un envío y se toma como el valor 'instantáneo' relacionado al instante de tiempo en el que se reciben los datos en la plataforma. *Max* y *Min* corresponden al valor máximo y mínimo que se pudo llegar a medir entre envío y envío.

Es de notar que cuanto más distantes en el tiempo se hagan los envíos, más valor van a aportar estos cuatro campos medidos, puesto que cuando los envíos se realizan muy cercanos en el tiempo los cuatro valores tienden a converger a un único valor.

Los datos generados son:

- *id* [6 Bytes]: ID del dispositivo
- *aux* [2 Bytes]: Banderas auxiliares (no tienen ningún uso asignado)
- *avg_humidity* [8 Bytes]: Promedio de humedad (en %) medido entre envíos
- *last_humidity* [8 Bytes]: Valor de humedad (en %) medido al momento de enviar
- *max_humidity* [8 Bytes]: Valor máximo de humedad (en %) medido entre envíos

- *min_humidity* [8 Bytes]: Valor mínimo de humedad (en %) medido entre envíos
- *avg_temperature* [8 Bytes]: Promedio de temperatura (en °C) medido entre envíos
- *last_temperature* [8 Bytes]: Valor de temperatura (en °C) medido al momento de enviar
- *max_temperature* [8 Bytes]: Valor máximo de temperatura (en °C) medido entre envíos
- *min_temperature* [8 Bytes]: Valor mínimo de temperatura (en °C) medido entre envíos
- *avg_atmospheric_pressure* [8 Bytes]: Promedio de presión atmosférica (en Pa) medido entre envíos
- *last_atmospheric_pressure* [8 Bytes]: Valor de presión atmosférica (en Pa) medido al momento de enviar
- *max_atmospheric_pressure* [8 Bytes]: Valor máximo de presión atmosférica (en Pa) medido entre envíos
- *min_atmospheric_pressure* [8 Bytes]: Valor mínimo de presión atmosférica (en Pa) medido entre envíos
- *avg_uv_index* [8 Bytes]: Promedio del índice de radiación UV medido entre envíos
- *last_uv_index* [8 Bytes]: Valor del índice UV medido al momento de enviar
- *max_uv_index* [8 Bytes]: Valor máximo del índice UV medido entre envíos
- *min_uv_index* [8 Bytes]: Valor mínimo del índice UV medido entre envíos
- *avg_uv_a* [8 Bytes]: Promedio de medición raw de radiación UV de tipo A medido entre envíos
- *last_uv_a* [8 Bytes]: Valor raw de radiación UV de tipo A medido al momento de enviar
- *max_uv_a* [8 Bytes]: Valor máximo raw de radiación UV de tipo A medido entre envíos
- *min_uv_a* [8 Bytes]: Valor mínimo raw de radiación UV de tipo A medido entre envíos
- *avg_uv_b* [8 Bytes]: Promedio de medición raw de radiación UV de tipo B medido entre envíos
- *last_uv_b* [8 Bytes]: Valor raw de radiación UV de tipo A medido al momento de enviar

- *max_uv_b* [8 Bytes]: Valor máximo raw de radiación UV de tipo B medido entre envíos
- *min_uv_b* [8 Bytes]: Valor mínimo raw de radiación UV de tipo B medido entre envíos
- *avg_wind_speed* [8 Bytes]: Promedio de velocidad del viento (en Km/h) medido entre envíos
- *last_wind_speed* [8 Bytes]: Valor de velocidad del viento (en Km/h) al momento de enviar
- *max_wind_speed* [8 Bytes]: Valor máximo de velocidad del viento (en Km/h) medido entre envíos
- *min_wind_speed* [8 Bytes]: Valor mínimo de velocidad del viento (en Km/h) medido entre envíos
- *avg_wind_direction* [8 Bytes]: Dirección del viento (el punto cardinal) más recurrente entre envíos
- *last_wind_direction* [8 Bytes]: Dirección del viento (el punto cardinal) al momento de enviar
- *rain_measure* [8 Bytes]: Valor de precipitación (en mm) medido entre envíos

3.5.4. Mejoras sugeridas y trabajo futuro

Las pruebas de campo para este sensor quedaron por fuera del alcance del proyecto, por lo que se considera importante poder realizarlas por medio de un despliegue en alguna localización de la Facultad a la intemperie, probando con esto la conectividad LoRa, la resistencia al clima y la calidad de las mediciones. Para una guía inicial de configuración y flash del dispositivo ver la Sección 2.1 y (Veirana, 2021, Pag. 18-23, [99]). Para ver cómo utilizar la antena LoRa de la Facultad ver (Veirana, 2021, Pag. 33-36, [99]).

3.6. Cuarto caso de uso: Sensor de monitoreo de cultivos

3.6.1. Introducción

A mediados de 2022 surgió la iniciativa *Fing Circular*, un proyecto de extensión de la Facultad de Ingeniería en el que se construyó una huerta comunitaria en el

predio detrás del INCO. Dicha huerta es trabajada tanto por funcionarios como por vecinos y dispone de múltiples canteros en los cuales se plantan diferentes verduras de estación así como también contenedores en los que se reciclan residuos orgánicos para la generación de compost.

Este último caso de uso no estaba contemplado en el relevamiento inicial de sensores sino que surgió posteriormente como un extra al conocer la iniciativa, la cual casualmente comenzó casi a la par de este proyecto de grado. Es por esto que es el sensor más elemental y menos desarrollado de todos pero que aún así tiene un gran potencial para ser usado en la huerta.

Figura 3.40: ***Fotografías de la huerta comunitaria del proyecto Fing circular.*** Fuente: udelar.edu.uy [98]

3.6.2. Motivación y objetivos

En este caso de uso se buscó el diseño de un dispositivo que pudiera medir distintos datos de interés en la huerta de la Facultad para poder ayudar a mantener un control más preciso del estado de los cultivos y poder automatizar distintos procesos y cargas de trabajo manuales.

3.6.3. Implementación

En este caso se utiliza nuevamente la placa de desarrollo Sparkfun ESP32 con LoRa en conjunto con un sensor de humedad del suelo y un fotoresistor (LDR) para medir la intensidad de la luz solar que recibe la planta a lo largo del día. Con estos sensores es posible llevar a cabo un monitoreo de estas variables para poder conocer el ambiente en el que están los cultivos a lo largo de su crecimiento y así poder darles las mejores condiciones posibles para su correcto desarrollo.

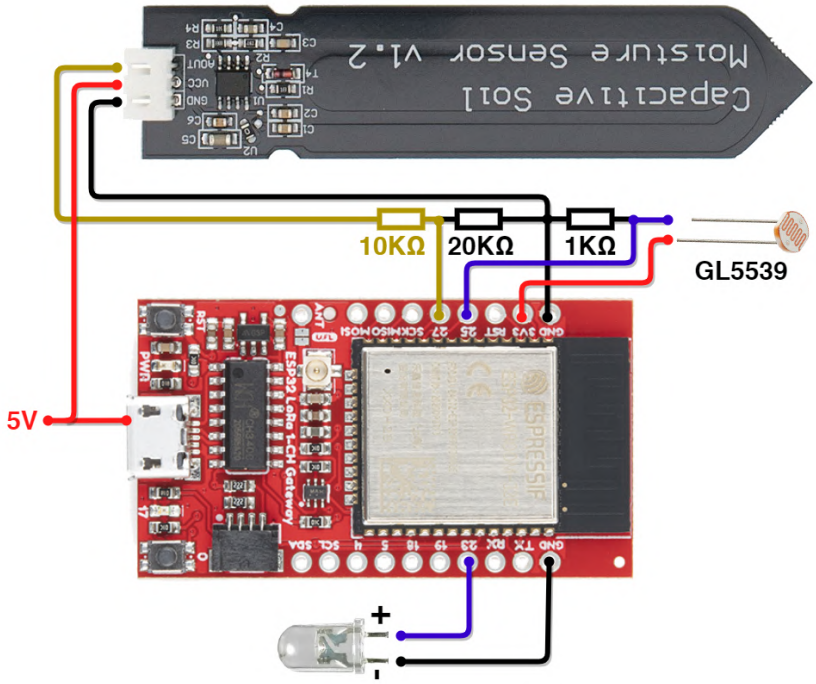

Figura 3.41: ***Diagrama de conexiones del sensor para huertas.***

En la Figura 3.41 se muestra el diagrama del sensor desarrollado, como se puede apreciar es un sensor básico pero que sin embargo posee algunas particularidades en su diseño que vale la pena mencionar.

3.6.4. Sobre los sensores de humedad del suelo

En esta sección se presenta una descripción de algunos tipos de sensores de humedad del suelo y algunas particularidades y desafíos presentadas por el sensor en particular utilizado para el prototipo. Se podría decir que existen dos grandes familias de estos sensores de humedad del suelo más comunes y económicas disponibles en el mercado para este tipo de microcontroladores. Cada una de estas familias de sensores miden el suelo utilizando distintas técnicas y tienen sus propias características, ventajas y desventajas.

La primera de estas familias es la de los sensores resistivos (Figura 3.42 a la izquierda), estos sensores miden la resistencia eléctrica que genera el suelo entre sus dos placas al hacer circular corriente continua. Cuanto más húmeda es la tierra en la que se encuentra, más va a conducir esta corriente y por ende ejerce menor resistencia. Estos sensores son extremadamente económicos y fáciles de utilizar pero tienen un problema grave de corrosión que hace que su utilización a largo

plazo sea completamente inviable[90].

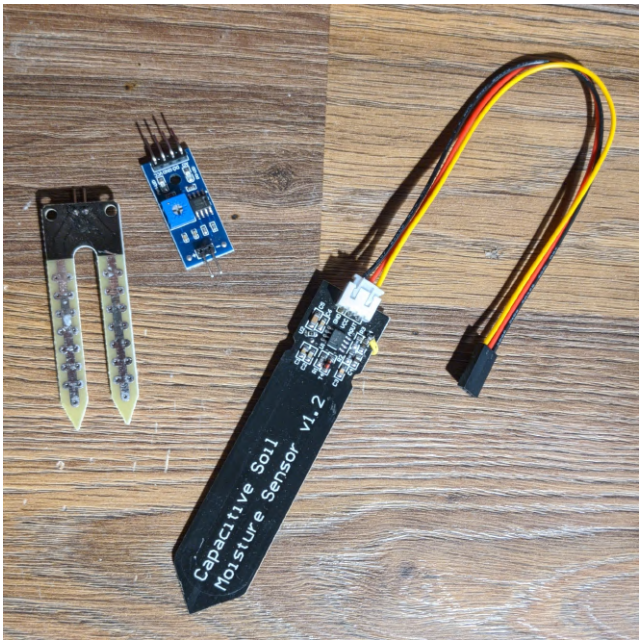

Figura 3.42: ***Fotografía de los dos tipos de sensores de humedad de suelo. Izquierda: sensor resistivo de humedad de suelo. Derecha: sensor capacitivo de humedad de suelo.***

La segunda familia es la de los sensores capacitivos (Figura 3.42 a la derecha), estos sensores poseen los dos terminales de un capacitor en su placa principal y utilizan el suelo como material dieléctrico entre estos terminales. Realizando mediciones en los cambios de capacitancia es que este sensor puede inferir cual es la humedad en la tierra, ya que el agua es el factor que más influye en esta variación. Estos no tienen partes metálicas expuestas por lo que solventan el gran problema de los sensores resistivos, sin embargo no son tan sencillos y directos de utilizar dado que existen a la venta múltiples variantes que aparentan ser el mismo sensor pero que tienen pequeñas diferencias y fallas de diseño en la circuitería de la parte superior del mismo, lo que hace que más del 80 % de los modelos disponibles no funcionen correctamente[69].

El sensor utilizado para el prototipo tenía un defecto en la circuitería que no permitía que el sensor funcionase en los rangos de voltaje 3.3V-5V declarados por el fabricante, sino que solamente funcionaba a 5V. Como la placa ESP32 solo trabaja a 3.3V en todos sus pines, no era posible su conexión de forma directa con el sensor. Es por esto que en el diagrama presentado en la Figura 3.41 se muestra como por un lado la alimentación del sensor es obtenida desde el cable de alimentación de 5V que va hacia el conector USB de la placa y por otro lado el pin 27 encargado de leer

las medidas analógicas del sensor posee un divisor de tensión[54] que lleva los 5V del sensor a los 3.3V con los que trabaja el pin, usando una resistencia de 10KΩ y otra de 20KΩ.

3.6.4.1. Datos generados por el sensor

En esta sección se detallan todos los datos que se generan en cada envío hacia la plataforma. En general se pueden identificar cuatro datos correspondientes a cada una de las variables censadas: *avg*, *last*, *max* y *min*. *Avg* corresponde al valor promedio calculado de todas las mediciones realizadas de manera continua entre envío y envío. *Last* corresponde al último valor medido antes de realizar un envío y se toma como el valor 'instantáneo' relacionado al instante de tiempo en el que se reciben los datos en la plataforma. *Max* y *Min* corresponden al valor máximo y mínimo que se pudo llegar a medir entre envío y envío.

Es de notar que cuanto más distantes en el tiempo se hagan los envíos, más valor van a aportar estos cuatro campos medidos, puesto que cuando los envíos se realizan muy cercanos en el tiempo los cuatro valores tienden a converger a un único valor.

Los datos generados son:

- *id* [6 Bytes]: ID del dispositivo
- *aux* [2 Bytes]: Banderas auxiliares (no tienen ningún uso asignado)
- *avg_soil_humidity* [8 Bytes]: Valor promedio de humedad relativa en el suelo medido entre envíos
- *last_soil_humidity* [8 Bytes]: Valor de humedad relativa en el suelo medido al momento de enviar
- *max_soil_humidity* [8 Bytes]: Valor máximo de humedad relativa en el suelo medido entre envíos
- *min_soil_humidity* [8 Bytes]: Valor mínimo de humedad relativa en el suelo medido entre envíos
- *avg_light* [8 Bytes]: Valor promedio de luz relativa medida entre envíos
- *last_light* [8 Bytes]: Valor de luz relativa medida al momento de enviar
- *max_light* [8 Bytes]: Valor máximo de luz relativa medida entre envíos
- *min_light* [8 Bytes]: Valor mínimo de luz relativa medida entre envíos

3.6.5. Mejoras sugeridas y trabajo futuro

Como se menciona en el comienzo de la sección, este sensor tiene potencial para seguir creciendo. Existen artículos en los que se hacen revisiones bibliográficas de proyectos IoT aplicados en el monitoreo de cultivos y la agricultura de precisión en los que se citan numerosas variables adicionales que pueden ser agregadas a una huerta. Uno de estos artículos (Aluthgama Acharige et al., 2019, [5]) muestra que una variable frecuentemente usada en la agricultura de precisión es el pH del suelo, lo cual se considera que efectivamente agregaría valor al sensor. En (Archbold Taylor et al., 2019, [7]), (Na et al., 2016, [64]) y (Sihombing y Listiari, 2020, [79]) se crean sensores capaces de leer el pH del suelo utilizando distintos hardwares.

Otras variables recabadas de (Farooq et al., 2020, [38]) que también pueden agregar valor al sensor son la temperatura del suelo y la calidad del agua usada para el riego. La temperatura y humedad ambiental dentro del predio de la huerta también podría ser de utilidad, para esto se puede utilizar el mismo sensor Si7021 usado en el caso de uso de sensores de calidad de aire en la Sección 3.3.

En lo que respecta a actuadores, existe la posibilidad agregar un controlador de riego en función de las mediciones de humedad del suelo con el fin de crear un sistema de riego automático.

3.6.6. Resultados obtenidos

Las pruebas de campo para este sensor quedaron por fuera del alcance del proyecto.

Las pruebas de medición durante el desarrollo fueron llevadas a cabo midiendo la humedad de tres muestras en tres vasos (Figura 3.43), donde el primero tiene tierra húmeda, el segundo solo tiene agua y el tercero tierra seca.

Los resultados obtenidos con el sensor se presentan en la Tabla 3.7, como se puede observar las mediciones analógicas quedan bien diferenciadas entre las muestras de tierras, el agua y en el aire.

En cuanto al sensor de luz solar se presenta a continuación en la Figura 3.44 las mediciones de todo un día con cielo despejado. Se marca con las líneas rojas el inicio y final de la exposición de luz solar directa sobre el LDR, comenzando aproximadamente a las 11:32am y terminando aproximadamente a las 17:17pm. Se puede observar que se forma una meseta que indica de forma clara las horas de luz solar directa que la planta obtiene.

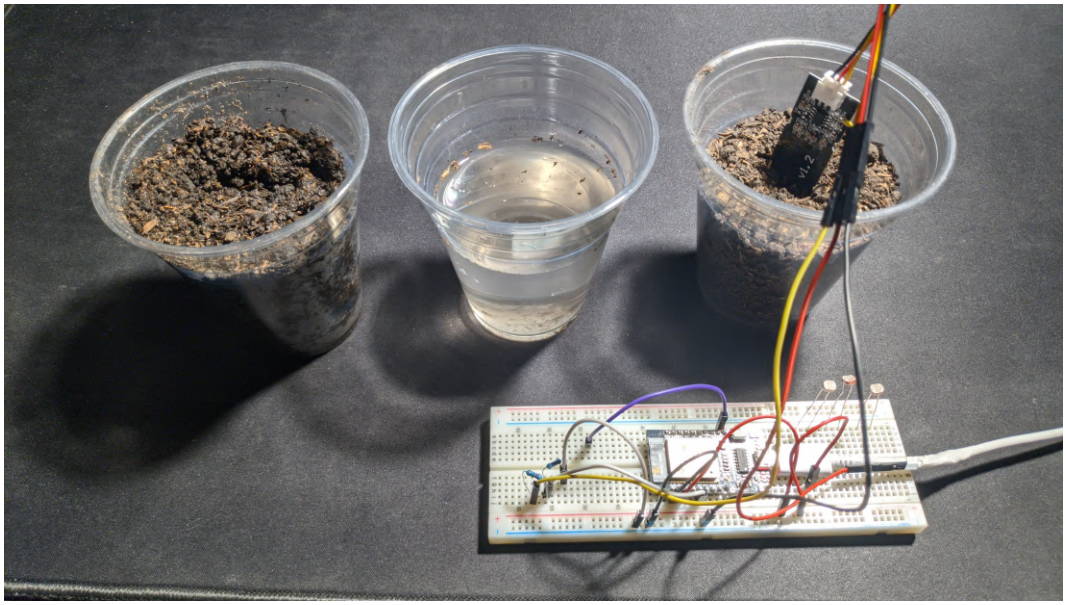

Figura 3.43: ***Fotografía de las pruebas realizadas en el desarrollo del sensor de humedad de suelo.***

Ubicación	Valor analógico
Aire libre	1600
Tierra seca	1400 - 1500
Tierra Húmeda	1100 - 1200
Agua	1000

Tabla 3.7: ***Resultados obtenidos en las pruebas de desarrollo del sensor de humedad de suelo.***

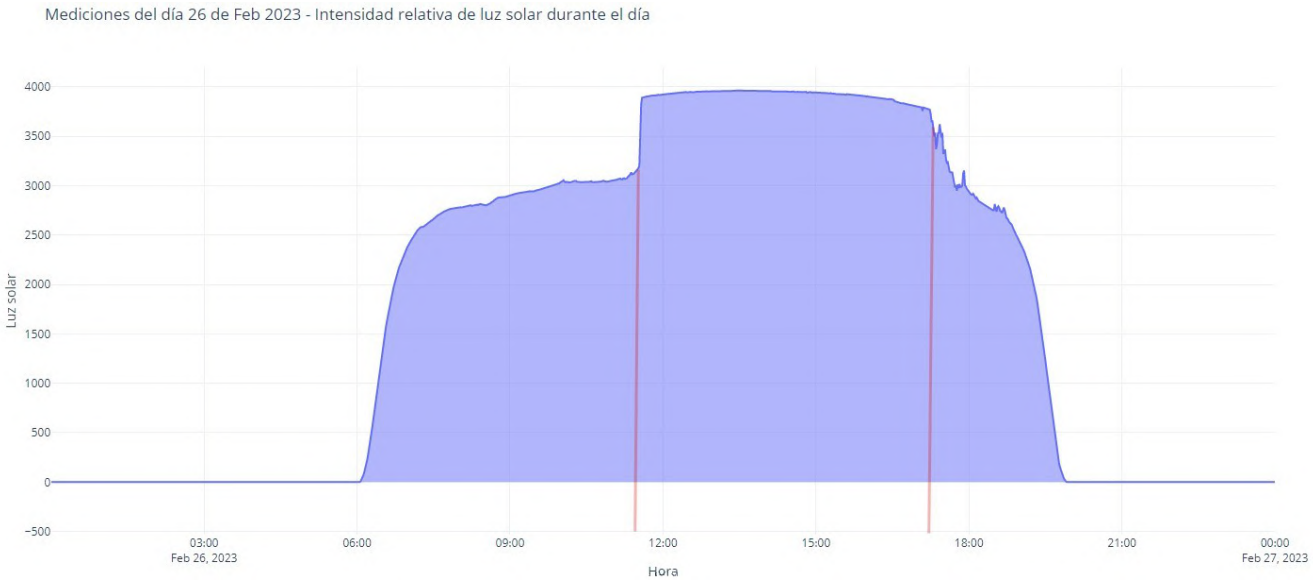

Figura 3.44: ***Gráfica de la medición del día 27/11/22 de intensidad relativa de luz solar.***

Capítulo 4

Consideraciones finales

4.1. Resumen del proyecto

Este proyecto de grado planteó como objetivo principal el desarrollo de un prototipo de campus inteligente para la Facultad de Ingeniería mediante la aplicación de tecnologías de Internet de las Cosas, sistemas embebidos y sistemas ciber-físicos.

Durante el proyecto se realizaron dos etapas principales de trabajo. La primera etapa consistió en el estudio de la plataforma ThingsBoard utilizada en el proyecto, seguido de la evaluación de las posibles infraestructuras sobre las cuales desplegar dicha plataforma y finalmente el desarrollo de dos posibles escenarios. El primero de estos escenarios consistió en un prototipo de prueba utilizando una única máquina virtual para alojar una instancia monolítica de ThingsBoard, dicho prototipo fue instalado y utilizado con éxito durante todo el proyecto. El segundo escenario fue basado en una instalación pensada para un ambiente productivo utilizando un cluster de Kubernetes, en donde se busca satisfacer los requerimientos de alta disponibilidad, robustez y seguridad. Este último escenario fue probado en ambientes simulados pero no fue desplegado en un escenario productivo real.

En la segunda etapa del proyecto se desarrollaron cuatro clases de sensores distintos pensados para ser desplegados en el campus inteligente e integrados directamente a la plataforma obtenida de la primera etapa. De estos cuatro tipos de sensores, dos fueron finalizados en cuanto a su construcción y están listos para ser desplegados: el sensor de ambiente y calidad de aire y el sensor de estación meteorológica. Sin embargo, de estos dos nombrados solo del sensor de ambiente y calidad de aire se llegaron a realizar pruebas de mediciones reales dentro de la Facultad. La estación meteorológica fue probada solamente de forma simulada durante

el desarrollo del código fuente del controlador principal. Los otros dos sensores (el sensor de conteo de espacios de estacionamiento y el sensor de huerta) no llegaron a ser construidos en una versión final, quedando en fase de prototipo. De estos dos últimos, ambos fueron probados en ambientes simulados para estudiar su viabilidad.

4.2. Conclusiones

A partir del trabajo realizado durante el proyecto se pudieron obtener resultados positivos tanto de la plataforma, como de los sensores y las tecnologías utilizadas. Se pudo observar que los sensores de ambiente y calidad de aire desplegados resultaron ser fiables y robustos, logrando obtener datos continuos y coherentes durante varias semanas casi sin interrupción. Esto muestra que tanto el chip ESP32, como también los sensores SGP40, Si7021 y sonido instalados en él tienen la capacidad de ser utilizados con un desempeño satisfactorio, pudiendo trabajar en ambientes productivos para la recolección de datos e incluso, mediante estos datos controlar sistemas no críticos utilizando actuadores que puedan generar un impacto sustancial en la Facultad. Por citar un ejemplo relacionado al caso tratado, trabajando en un sistema de circulación de aire que se active al detectar un índice malo de calidad de aire.

Gracias al caso del sensor de conteo de estacionamientos, el cual está integrado con el mundo de la Inteligencia Artificial, se pudo nuevamente dar constancia de que las posibilidades resultantes de unir esta disciplina de la computación con el Internet de las Cosas se amplían de manera sustancial y crean resultados que pueden brindar un notorio valor. Si bien este caso de uso en particular ya es bastante conocido dentro del ámbito académico, no hay duda de que nuevas posibles aplicaciones continuarán surgiendo en el futuro gracias al acelerado crecimiento que están teniendo tanto el mundo de la Inteligencia Artificial como el del Internet de las Cosas.

En líneas generales se constató que estas tecnologías, si bien son asequibles y de fácil acceso, pueden llegar a ser extremadamente potentes y versátiles para soluciones de IoT, teniendo un número incalculable de posibles aplicaciones dentro de un campus inteligente y en smart cities.

Respecto a la plataforma ThingsBoard se pudo concluir que es un muy buen software para aplicaciones IoT, siendo sencillo de instalar, de utilizar y con una gran cantidad de funcionalidades correctamente integradas. Sus numerosas variantes a la hora de ser instalado le dan una amplia flexibilidad y adaptación a distintos

escenarios, algo que también se valora.

4.3. Trabajo futuro

Para detallar de forma ordenada todas las posibles áreas en donde se puede dar continuidad a lo abordado en este proyecto, se listan por un lado las posibilidades del trabajo futuro relacionado a la plataforma y la infraestructura y luego por otro lado el trabajo futuro relacionado a la parte de sensores.

4.3.1. Trabajo futuro en la plataforma e infraestructura

Realizar un despliegue productivo con la arquitectura de cluster: El **primer y más importante** objetivo para realizar un avance sustancial en la implementación del campus inteligente consiste en la realización de una puesta en producción de la plataforma en la infraestructura de la Facultad. Para poder llevar a cabo esto es indispensable realizar una investigación previa para ahondar en los temas de costos operativos y seguridad informática detallados en la Subsección 2.4.1, seguido de esto se debe preparar la infraestructura en donde se instalará el cluster de Kubernetes y finalmente el despliegue ThingsBoard sobre él. También es necesario investigar cómo se expondrá la plataforma a Internet así como la implementación de un nombre de dominio adecuado y la planificación del soporte, la gestión y la evolución de los sistemas.

De todos estos avances es importante generar manuales y guías de instalación para dejar documentado el proceso, facilitando el mantenimiento y futuras mejoras.

Integrar nuevas tecnologías de Internet de las Cosas: Durante el proyecto se utilizó únicamente WiFi para el envío de mensajes a la plataforma, pero para algunos de los sensores diseñados que deben ir instalados lejos de la red de la Facultad será necesario el uso de LoRa para enviar los mensajes. En la Facultad hay instalada una antena LoRa perteneciente a la empresa *Yeap!* que puede ser usada para recibir estos mensajes y posteriormente reenviarlos a una URL deseada, que en este caso es la URL de nuestra plataforma IoT. En (Veirana, 2021, Pag. 33-36, [99]) existe una guía de uso de la plataforma *Orbiwise* desde la cual se puede acceder a un sistema de gestión con un usuario cedido a la Facultad y con este administrar los dispositivos que envían datos a esta antena LoRa, allí se deben realizar configuraciones para poder hacer este reenvío de los datos hacia la plataforma. Por parte de

los sensores, todos los que están basados en el chip ESP32 tienen el código fuente preparado para poder transmitir usando este protocolo, por lo que basta con realizar un nuevo flasheo a cada dispositivo con las credenciales LoRa y el resto de parámetros de trabajo deseados. Estas credenciales y parámetros de trabajo son detallados en la Sección 2.1 y la guía para realizar el flash a estos dispositivos se puede ver en (Veirana, 2021, Pag. 18-23, [99]).

Habiendo explicado todo lo anterior, se tienen resueltas las partes respectivas a la emisión de datos (los sensores) y la recepción de datos (la antena LoRa de Facultad junto con la plataforma Orbiwise), por lo que lo único que falta desarrollar para completar el flujo es la traducción de los strings crudos de datos enviados por los sensores (ver el formato de estos strings en la Sección 2.1) a un formato que la plataforma ThingsBoard pueda interpretar y almacenar.

Esto en principio puede ser resuelto de dos formas: la primera opción es mediante una API REST intermediaria que haga de middleware entre Orbiwise y ThingsBoard, esta aplicación recibiría desde Orbiwise el string de datos y se encargaría de parsearlo e interpretarlo para reenviarlo a ThingsBoard usando el endpoint de telemetría habitual, de tener el cluster de Kubernetes instalado, esta aplicación podría ser desplegada allí. La segunda opción es intentar configurar el Rule Engine de ThingsBoard para poder detectar este tipo de mensajes mediante la creación de un nuevo nodo en la rama del grafo. Este nodo contendría una función en JS para detectar y parsear el string crudo del mensaje antes de pasarlo a la regla encargada de almacenarlo, para ver más en detalle el Rule Engine de ThingsBoard ver la Subsección 1.4.5.

Alternativamente a lo todo mencionado también existe la posibilidad de instalar uno o varios gateways propios para la creación de una red LoRa administrada completamente por la Facultad, respecto a estos dispositivos existe una gran variedad de alternativas, en (Veirana, 2021, Pag. 23-32, [99]) y en (Crizul y Gómez, 2021, Pag. 136, [22]) hay guías de trabajo para los gateways *The Things Gateway* y *Dragino* respectivamente.

Mejorar las bases de datos de las arquitecturas propuestas: En la arquitectura de cluster presentada se utiliza una instalación monolítica de la base de datos PostgreSQL para la plataforma IoT. Se considera necesario poder mejorar este apartado brindando una solución más compleja que aporte tolerancia a fallos o al menos la

capacidad de recuperación ante los mismos. En el mejor de los casos esto se traduce en crear una base de datos distribuida y replicada en diferentes nodos, o en su defecto se traduce en agregar la capacidad de auto generar respaldos regularmente de la base de datos.

Respecto al apartado de las tecnologías utilizadas en la base de datos, este puede mejorarse mediante la instalación de la modalidad híbrida que brinda ThingsBoard utilizando PostgreSQL y Cassandra. Con esta instalación, además de generar conocimiento, se puede dar por seguro que la base de datos no necesitará ser refactorizada a futuro en el caso de un crecimiento considerable en la cantidad de sensores y datos generados.

Desarrollar aplicaciones web o una app que muestre los datos: Una vez se tenga un sólido despliegue productivo del campus y una cantidad mayor de sensores conectados a él, se estaría en posición de comenzar con el desarrollo de aplicaciones que puedan permitir al público general de la Facultad acceder a esos datos de una manera sencilla y amigable.

4.3.2. Trabajo futuro en los dispositivos y sensores

Respecto al trabajo futuro para los sensores desarrollados, se plantearon diversas mejoras específicas en las respectivas secciones de cada uno de ellos (a excepción del caso de uso tres). Por otra parte, se presentan a continuación otras mejoras o propuestas más generales.

Integrar la red WiFi de Antel para el envío de datos: Una mejora que no fue mencionada en el informe y que es transversal a todos los sensores es la adaptación de estos a la red WiFi *antel_Libre* existente en la Facultad. Esta red WiFi tiene mayor cobertura que *wifing* pero tiene la particularidad de poseer un portal cautivo conformado por una página web con la que el usuario debe interactuar pulsando un botón en ella para confirmar la conexión a la red.

A la hora de realizar la conexión de los dispositivos, no fue posible simular el comportamiento descrito dentro de la lógica del sensor. Se intentó enviar un HTTP request armado por el sensor con el mismo formato y datos presentes en el request enviado por la página web al momento de pulsar el botón pero la conexión no se termina de realizar con éxito.

En el repositorio del proyecto se adjunta el código fuente de esta página web

así como las grabaciones de Wireshark en donde se registraron los paquetes que se intercambian en esta conexión a antel_Libre.

Profundizar en los actuadores, comunicación servidor-dispositivo y mesh de sensores: Un apartado que no fue tratado en profundidad en el proyecto es la comunicación iniciada desde el servidor hacia un dispositivo, por ejemplo en el caso de poseer un actuador que requiere ser activado o desactivado, si bien ThingsBoard posee RPCs que tratan esta funcionalidad[11], en el proyecto solo se estudiaron los RPCs comenzados desde el dispositivo (Subsubsección 1.4.5.1).

También podría llegar a ser de interés ahondar en temáticas más abstractas o teóricas como la creación de un prototipo de mesh de sensores, en donde estos se puedan intercomunicar entre sí a la vez que lo hacen libremente con el servidor y en base a variables o relaciones complejas captadas del ambiente, poder tomar decisiones o realizar ciertas acciones.

Plantear nuevos sensores y dispositivos para el campus: Durante las fases iniciales del proyecto se hizo un brainstorming con ideas para posibles sensores de la cual surgió una extensa lista. Si bien la gran mayoría de ellos no fueron implementados en el proyecto por razones de prioridad y alcance, se dejan a continuación algunas ideas que se consideran posibles o como mínimo interesantes.

- **Sensor de termográfico para medir la temperatura corporal o la temperatura de maquinaria:** Durante el relevamiento de ideas se comprobó la existencia del sensor *SparkFun IR Array Breakout*[83] con el cual se podrían llegar a generar casos de uso para la medición de la temperatura corporal de los estudiantes que por ejemplo, pasan por la entrada principal. Alternativamente podría usarse para monitorear la temperatura de alguna maquinaria crítica existente en la Facultad, por ejemplo un rack de servidores.
- **Iluminación inteligente:** Este punto abarca un tema que puede llegar a ser complejo de implementar y que tiene múltiples formas de ser enfocado, sin embargo se considera que, de ser implementado correctamente puede generar mucho valor a la Facultad así como ahorro de dinero y energía eléctrica.
- **Detección en tiempo real de equipos libres en salas de máquinas Linux y Windows:** Este punto no fue investigado en profundidad, pero podría ser interesante para los estudiantes tener acceso a esa información.

- **Sensor de gas en la cantina:** Sparkfun posee numerosas versiones de sensores de detección de gas en el aire, con estos se podría llegar a crear un sensor de seguridad instalado en la cocina de la cantina que se encargue de detectar posibles filtraciones de gas.

El resto se deja a cargo del lector y de su creatividad, se alienta encarecidamente a todo aquel que tenga nuevas ideas a plantearlas para poder ser contempladas.

Referencias bibliográficas

[1] Gastón Abellá, Ana Machado y Daniel Susviela. «Tecnologías geoespaciales en plataformas de Smart Cities». En: (2021). DOI: 20.500.12008/29195. URL: https://hdl.handle.net/20.500.12008/29195.

[2] Federico Acevedo, Guillermo Coduri y Guzmán Perera. «CattleNet : geolocalizacoón con LoRa mediante multilateración». En: (2018). DOI: 20.500.12008/20293. URL: https://hdl.handle.net/20.500.12008/20293.

[3] Alexandre Cassen. *Sitio web oficial de Keepalived*. URL: https://www.keepalived.org/index.html.

[4] Santiago Alles Conde. «Campus Inteligente». En: (2022). DOI: 20.500.12008/35410. URL: https://hdl.handle.net/20.500.12008/35410.

[5] Raneesha Aluthgama Acharige et al. «Adoption of the Internet of Things (IoT) in Agriculture and Smart Farming towards Urban Greening: A Review». En: *International Journal of Advanced Computer Science and Applications* 10 (abr. de 2019), págs. 11-28. DOI: 10.14569/IJACSA.2019.0100402.

[6] Inc. Amazon Web Services. *Sitio web oficial de FreeRTOS*. URL: https://www.freertos.org/.

[7] George Archbold Taylor et al. «pH Measurement IoT System for Precision Agriculture Applications». En: *IEEE Latin America Transactions* 17.05 (2019), págs. 823-832. DOI: 10.1109/TLA.2019.8891951.

[8] Alexis Arriola García y Gabriela Wynants Lombardini. «Relevamiento de arquitecturas para desarrollo de aplicaciones de Internet de las Cosas». En: (2018). DOI: 20.500.12008/19035. URL: https://hdl.handle.net/20.500.12008/19035.

[9] atomic14. *ESP32 ESP-Now Real World Range Test - Standard and Long Range Mode Investigated*. URL: https://www.youtube.com/watch?v=oz0a7Ur7nko.

[10] The ThingsBoard Authors. *Installing ThingsBoard CE on Ubuntu Server*. URL: https://thingsboard.io/docs/user-guide/install/ubuntu/.

[11] The ThingsBoard Authors. *Send RPC Request to the Related Device*. URL: https://thingsboard.io/docs/user-guide/rule-engine-2-0/tutorials/rpc-request-tutorial/.

[12] Microsoft Azure. *Sitio web oficial de Microsoft Azure*. URL: https://azure.microsoft.com/es-es.

[13] SCET Berkeley. *YOLO Object Detection*. 2020. URL: https://www.youtube.com/watch?v=2hAiJe8ITsE.

[14] Antonio Bracco, Federico Grunwald y Agustín Navcevich. « Localización indoor basada en Wi-Fi». En: (2019). DOI: 20.500.12008/20593. URL: https://hdl.handle.net/20.500.12008/20593.

[15] Canonical. *Create a MicroK8s cluster*. URL: https://microk8s.io/docs/clustering.

[16] Canonical. *Documentación de los addons de Microk8s*. URL: https://microk8s.io/docs/addons.

[17] Canonical. *Sitio web oficial de Microk8s*. URL: https://microk8s.io/.

[18] Canonical. *Sitio web oficial de Microk8s sección Get started*. URL: https://microk8s.io/docs/getting-started.

[19] Cristian Cantos. *Contando coches con una Raspberry Pi*. 2019. URL: https://canbe.gitlab.io/es/posts/2019/03/contando-coches-con-una-raspberry-pi/.

[20] ciudadesdelfuturo.org.ar. *Montevideo Inteligente: el rol de la plataforma FIWARE*. URL: https://www.ciudadesdelfuturo.org.ar/fiware/novedades/montevideo-inteligente-el-rol-de-la-plataforma-fiware/.

[21] Google Cloud. *Sitio web oficial de Google Cloud*. URL: https://cloud.google.com/?hl=es.

[22] Francisco Crizul y Gerardo Gómez. «Localización en interiores utilizando infraestructura de Internet de las Cosas». En: (2021). DOI: 20.500.12008/27962. URL: https://hdl.handle.net/20.500.12008/27962.

[23] Fredrik Dahlqvist et al. «Growing opportunities in the Internet of Things». En: *McKinsey & Company* (2019), págs. 1-6.

[24] demo-dijiudu.readthedocs.io. *ULP coprocessor programming*. URL: https://demo-dijiudu.readthedocs.io/en/latest/api-guides/ulp.html.

[25] Federico Detta. «Aplicación de IoT con diversas tecnologías inalámbricas». En: (2020). DOI: 20.500.12008/23742. URL: https://hdl.handle.net/20.500.12008/23742.

[26] DigitalOcean. *How To Implement SSL Termination With HAProxy on Ubuntu 14.04*. URL: https://www.digitalocean.com/community/tutorials/how-to-implement-ssl-termination-with-haproxy-on-ubuntu-14-04.

[27] DigitalOcean. *Sitio web oficial de DigitalOcean*. URL: https://www.digitalocean.com/.

[28] Xiangwu Ding y Ruidi Yang. «Vehicle and Parking Space Detection Based on Improved YOLO Network Model». En: *Journal of Physics: Conference Series* 1325.1 (oct. de 2019), pág. 012084. DOI: 10.1088/1742-6596/1325/1/012084. URL: https://dx.doi.org/10.1088/1742-6596/1325/1/012084.

[29] Docker. *Sitio web oficial de Docker*. URL: https://www.docker.com/.

[30] docs.espressif.com. *Documentación técnica de ESP-NOW*. URL: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/esp_now.html.

[31] docs.espressif.com. *FreeRTOS (Overview)*. URL: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/freertos.html.

[32] María Soledad Escolar Díaz. *Smart ESI, un nuevo modelo de escuela*. 2021. URL: https://esi.uclm.es/index.php/programas-singulares/smartesi/.

[33] espressif.com. *ESP-NOW*. URL: https://www.espressif.com/en/news/ESP-NOW.

[34] espressif.com. *ESP32 Series Datasheet*. 2023. URL: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf.

[35] espressif.com. *Sitio web oficial de Espressif*. URL: https://www.espressif.com/.

[36] Facultad de Ingeniería. *Ciencia y tecnología en tiempos de pandemia*. 2020. URL: https://www.fing.edu.uy/index.php/es/paginas/ciencia-y-tecnologia-en-tiempos-de-pandemia/.

[37] Laith Farhan et al. «A Concise Review on Internet of Things (IoT) - Problems, Challenges and Opportunities». En: (2018). URL: https://e-space.mmu.ac.uk/id/eprint/621330.

[38] Shoaib Farooq et al. «Role of IoT Technology in Agriculture: A Systematic Literature Review». En: *Electronics* 9 (feb. de 2020), pág. 319. DOI: 10.3390/electronics9020319.

[39] Jill D. Fenske y Suzanne E. Paulson. «Human Breath Emissions of VOCs». En: *Journal of the Air & Waste Management Association* 49.5 (1999), págs. 594-598. DOI: 10.1080/10473289.1999.10463831. URL: https://doi.org/10.1080/10473289.1999.10463831.

[40] The Apache Software Foundation. *Sitio web oficial de Apache Cassandra*. URL: https://cassandra.apache.org/_/index.html.

[41] Othmane Friha et al. «Internet of Things for the Future of Smart Agriculture: A Comprehensive Survey of Emerging Technologies». En: *IEEE/CAA Journal of Automatica Sinica* 8.4 (2021), págs. 718-752. DOI: 10.1109/JAS.2021.1003925.

[42] Comunidad de GitHub. *What are the minimal hardware requirements to run microk8s*. URL: https://github.com/canonical/microk8s/issues/319.

[43] Google Cloud. *¿Qué es Kubernetes?* URL: https://cloud.google.com/learn/what-is-kubernetes?hl=es-419.

[44] Instituto de Computación y Facultad de Ingeniería Grupo MINA. *Sitio web oficial del Workshop en sistemas Ciber-Físicos de la Facultad de Ingeniería*. URL: https://www.fing.edu.uy/inco/grupos/mina/wscf2022/.

[45] HAProxy Technologies, LLC. *Documentación de las principales secciones de HAProxy*. URL: https://www.haproxy.com/blog/the-four-essential-sections-of-an-haproxy-configuration/.

[46] HAProxy Technologies, LLC. *Sitio web oficial de HAProxy*. URL: https://www.haproxy.com/.

[47] IBM. *Sitio web oficial de IBM Cloud*. URL: https://www.ibm.com/cloud.

[48] Facultad de Ingeniería. *Sitio web oficial de Ingeniería deMuestra*. URL: https://idm.uy/.

[49] Keysight Technologies. «The Internet of Things: Enabling Technologies and Solutions for Design and Test». En: (2017). URL: https://www.ee.co.za/wp-content/uploads/2017/03/IoT-enabling-solns-for-design-and-test-5992-1175EN.pdf.

[50] Kubernetes Authors. *Documentacion oficial de Kubernetes*. 2022. URL: https://kubernetes.io/docs/home/.

[51] Kubernetes Authors. *Documentacion oficial de los componentes de Kubernetes*. 2022. URL: https://kubernetes.io/docs/concepts/overview/components/.

[52] *kubernetes-sigs/nfs-subdir-external-provisioner*. URL: https://github.com/kubernetes-sigs/nfs-subdir-external-provisioner.

[53] James F. Kurose y Keith W. Ross. *Redes de computadoras. Un enfoque descendente, 7 Ed.* Pearson, 2017. ISBN: 978-84-9035-528-2.

[54] learn.sparkfun.com. *Voltage Dividers*. URL: https://learn.sparkfun.com/tutorials/voltage-dividers/all.

[55] E. A. Lee y S. A. Seshia. *Introduction to Embedded Systems- A Cyber-Physical Systems Approach, Second Edition*. MIT Press, 2017. ISBN: 978-0-262-53381-2.

[56] Tsung-Yi Lin et al. «Focal Loss for Dense Object Detection». En: (2018). DOI: https://doi.org/10.48550/arXiv.1708.02002. arXiv: 1708.02002 [quant-ph]. URL: https://arxiv.org/abs/1708.02002.

[57] S. Liu et al. «Contribution of human-related sources to indoor volatile organic compounds in a university classroom». En: *Indoor Air* 26.6 (2016), págs. 925-938. DOI: https://doi.org/10.1111/ina.12272. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/ina.12272.

[58] Wei Liu et al. «SSD: Single Shot MultiBox Detector». En: (2016). DOI: https://doi.org/10.48550/arXiv.1512.02325. arXiv: 1512.02325 [quant-ph]. URL: https://arxiv.org/abs/1512.02325.

[59] Praveen Kumar Malik et al. «Industrial Internet of Things and its Applications in Industry 4.0: State of The Art». En: *Computer Communications* 166 (2021), págs. 125-139. ISSN: 0140-3664. DOI: https://doi.org/10.1016/j.comcom.2020.11.016. URL: https://www.sciencedirect.com/science/article/pii/S0140366420319964.

[60] Joaquín Márquez y Gabriel Rodríguez. «Análisis de seguridad del protocolo DLMS/COSEM en el contexto de SmartGrids». En: (2020). DOI: 20.500.12008/25264. URL: https://hdl.handle.net/20.500.12008/25264.

[61] media.mit.edu. *Sitio web oficial del MIT media lab research*. URL: https://www.media.mit.edu/research/?filter=everything&tag=internet-things.

[62] Microchip Technology Inc. *TCP/IP Five-Layer Software Model Overview*. 2021. URL: https://microchipdeveloper.com/tcpip:tcp-ip-five-layer-model.

[63] montevideo.gub.uy/. *Sensores y sistemas interconectados*. URL: https://montevideo.gub.uy/areas-tematicas/servicios-digitales/sensores-y-sistemas-interconectados.

[64] Abdullah Na et al. «An IoT based system for remote monitoring of soil characteristics». En: *2016 International Conference on Information Technology (InCITe) - The Next Generation IT Summit on the Theme - Internet of Things: Connect your Worlds*. 2016, págs. 316-320. DOI: 10.1109/INCITE.2016.7857638.

[65] El Observador. *Un estacionamiento más inteligente en el shopping*. 2012. URL: https://www.elobservador.com.uy/nota/un-estacionamiento-mas-inteligente-en-el-shopping-2012112314130.

[66] Felipe Osimani y Bruno Stecanella. «Aplicación de tecnologías de Internet de las Cosas para recolección de datos». En: (2018). DOI: 20.500.12008/19036. URL: https://hdl.handle.net/20.500.12008/19036.

[67] Martin Pacheco. «Despliegue de red IoT Prueba de concepto de despliegue de una red IoT utilizando el framework FIWARE y el dispositivo KitIoT». En: (2020). URL: https://eva.fing.edu.uy/pluginfile.php/324214/mod_folder/content/0/despliegue%5C%20de%5C%20red%5C%20IoT-PachecoMartin.pdf?forcedownload=1.

[68] Sebastián Passaro y Martín Pacheco. «Contramedidas para la manipulación maliciosa de dispositivos en LoRaWAN». En: (2020). DOI: 20.500.12008/25066. URL: https://hdl.handle.net/20.500.12008/25066.

[69] Flaura - Smart Plant Pot. *Capacitive Soil Moisture Sensors don't work correctly + Fix for v2.0 v1.2 Arduino ESP32 Raspberry Pi*. URL: https://www.youtube.com/watch?v=IGP38bz-K48.

[70] *Preliminary Datasheet SGP40, Indoor Air Quality Sensor for VOC Measurements*. URL: https://cdn.sparkfun.com/assets/e/6/2/6/d/Sensirion_Gas_Sensors_SGP40_Datasheet.pdf.

[71] Sara Santos - randomnerdtutorials.com. *ESP-NOW Two-Way Communication Between ESP32 Boards*. URL: https://randomnerdtutorials.com/esp-now-two-way-communication-esp32/.

[72] Sara Santos - randomnerdtutorials.com. *ESP32 BLE Server and Client (Bluetooth Low Energy)*. URL: https://randomnerdtutorials.com/esp32-ble-server-client/#ESP32-BLE-Server.

[73] raspberrypi.com. *Sitio web oficial de Raspberry Pi*. URL: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/.

[74] Joseph Redmon. *Darknet: Open Source Neural Networks in C*. http://pjreddie.com/darknet/. 2013–2016.

[75] Joseph Redmon y Ali Farhadi. «YOLOv3: An Incremental Improvement». En: (2018). DOI: https://doi.org/10.48550/arXiv.1804.02767. arXiv: 1804.02767 [quant-ph]. URL: https://arxiv.org/abs/1804.02767.

[76] Joseph Redmon et al. «You Only Look Once: Unified, Real-Time Object Detection». En: (2015). DOI: https://doi.org/10.48550/arXiv.1506.02640. arXiv: 1506.02640 [quant-ph]. URL: https://arxiv.org/abs/1506.02640.

[77] Sensirion. *SGP40 – VOC Index forExperts*. 2020. URL: https://cdn.sparkfun.com/assets/e/9/3/f/e/GAS_AN_SGP40_VOC_Index_for_Experts_D1.pdf.

[78] Amazon Web Services. *Sitio web oficial de Amazon Web Services*. URL: https://aws.amazon.com/es/.

[79] Yuan Alfinsyah Sihombing y Sustia Listiari. «Detection of air temperature, humidity and soil pH by using DHT22 and pH sensor based Arduino nano microcontroller». En: *AIP Conference Proceedings* 2221.1 (2020), pág. 100008. DOI: 10.1063/5.0003115. eprint: https://aip.scitation.org/doi/pdf/10.1063/5.0003115. URL: https://aip.scitation.org/doi/abs/10.1063/5.0003115.

[80] SoundSystems.es. *La envolvente acústica*. 2019. URL: https://soundsystems.es/la-envolvente-acustica/.

[81] sparkfun.com. *SparkFun Air Quality Sensor - SGP30*. URL: https://www.sparkfun.com/products/16531.

[82] sparkfun.com. *SparkFun Air Quality Sensor - SGP40*. URL: https://www.sparkfun.com/products/18345.

[83] sparkfun.com. *SparkFun IR Array Breakout - 55 Degree FOV, MLX90640 (Qwiic)*. URL: https://www.sparkfun.com/products/14844.

[84] sparkfun.com. *SparkFun LoRa Gateway - 1-Channel (ESP32)*. URL: https://www.sparkfun.com/products/18074.

[85] sparkfun.com. *SparkFun Particle Photon*. URL: https://www.sparkfun.com/products/13774.

[86] sparkfun.com. *SparkFun Photon Weather Shield*. URL: https://www.sparkfun.com/products/retired/13674.

[87] sparkfun.com. *SparkFun Sound Detector*. URL: https://www.sparkfun.com/products/13763.

[88] sparkfun.com. *SparkFun Sound Detector*. URL: https://www.sparkfun.com/products/14262.

[89] sparkfun.com. *Weather Meter Kit*. URL: https://www.sparkfun.com/products/15901.

[90] Andreas Spiess. *207 Why most Arduino Soil Moisture Sensors suck (incl. solution)*. URL: https://www.youtube.com/watch?v=udmJyncDvw0.

[91] C. Stönner, A. Edtbauer y J. Williams. «Real-world volatile organic compound emission rates from seated adults and children for use in indoor air studies». En: *Indoor Air* 28.1 (2018), págs. 164-172. DOI: 20.500.12008/23742. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/ina.12405.

[92] The ThingsBoard Authors. *Sitio web oficial de Thingsboard*. 2023. URL: https://thingsboard.io/.

[93] The ThingsBoard Authors. *Sitio web oficial de Thingsboard*. 2023. URL: https://thingsboard.io/docs/getting-started-guides/what-is-thingsboard/.

[94] The ThingsBoard Authors. *Sitio web oficial de Thingsboard*. 2023. URL: https://thingsboard.io/docs/reference/monolithic/.

[95] The ThingsBoard Authors. *Sitio web oficial de Thingsboard*. 2023. URL: https://thingsboard.io/docs/reference/msa/.

[96] ThingsBoard. *Repositorio GitHub: thingsboard/thingsboard-ce-k8s*. URL: https://github.com/thingsboard/thingsboard-ce-k8s.git.

[97] tutorials-raspberrypi.com. *How-To: Bluetooth Connection between ESP32's and Raspberry Pi's*. URL: https://tutorials-raspberrypi.com/esp32-bluetooth-connection-to-esp8266-and-raspberry-pi/.

[98] udelar.edu.uy. *Huerta comunitaria en Fing: clasificar transformar y producir*. URL: https://udelar.edu.uy/portal/2022/08/huerta-comunitaria-en-fing-clasificar-transformar-y-producir/.

[99] Joaquin Veirana. «Sensores de Calidad de Aire». En: (2021). DOI: 10.5281/zenodo.7633049. URL: https://doi.org/10.5281/zenodo.7633049.

[100] Chien-Yao Wang y Alexey Bochkovskiy abd Hong-Yuan Mark Liao. «YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors». En: (2022). DOI: https://doi.org/10.48550/arXiv.2207.02696. arXiv: 2207.02696 [quant-ph]. URL: https://arxiv.org/abs/2207.02696.

[101] R. Yusnita, Fariza Norbaya y Norazwinawati Basharuddin. «Intelligent Parking Space Detection System Based on Image Processing». En: *International Journal of Innovation, Management and Technology* 3.3 (2012), págs. 232-235. URL: http://ijimt.org/papers/228-G0038.pdf.

[102] Zhengxia Zou et al. «Object Detection in 20 Years: A Survey». En: (2019). DOI: 10.48550/ARXIV.1905.05055. URL: https://arxiv.org/abs/1905.05055.

[103] Elizabeth D. Zwicky, Simon Cooper y D. Brent Chapman. *Building Internet Firewalls, 2nd Edition*. O'Reilly Media, Inc., 2000. ISBN: 9781565928718.

APÉNDICES

Apéndice 1

Procedimientos del capítulo 2

1.1. Creación de la infraestructura del servidor del primer caso (Arquitectura del prototipo)

La aplicación de ThingsBoard en su versión monolítica llevada a cabo en el prototipo utilizó una única máquina virtual (Minube) en donde se encuentran la aplicación, la base de datos y el software HAProxy para exponer la plataforma a Internet.

Para llevar a cabo la instalación de ThingsBoard se utilizó la guía oficial de instalación en Ubuntu provista por ThingsBoard[10]. Por otro lado, para la exposición de la aplicación utilizando HAProxy fue necesario instalarlo con el comando a continuación.

```
$ sudo apt–get install haproxy psmisc –y
```

Listing 1.1: Instalación de HAProxy

Seguido del procedimiento detallado en la Sección 1.3 para concluir la configuración de HAProxy.

1.2. Creación de la infraestructura del servidor del segundo caso (Arquitectura de cluster)

Para desplegar la aplicación de ThingsBoard se utilizó un cluster de Kubernetes instalado en múltiples máquinas virtuales en conjunto con un balanceador de carga con el fin de proveer alta disponibilidad y tolerancia a fallos. Para la creación del

cluster se utilizó la herramienta *MicroK8s* en su versión 1.24. Dicha herramienta es posible de obtener al momento de la instalación del sistema operativo Ubuntu o posteriormente por medio de una instalación normal.

La herramienta Microk8s provee una forma directa y sencilla para crear un cluster distribuido. Mientras que con Kubeadm, la herramienta oficial de Kubernetes, el proceso de instalación es más largo y complejo con Microk8s se requiere de la ejecución de un solo comando por cada máquina virtual que se desee unir para lograrlo. Sumado a esto, Microk8s tiene plugins para agregar al cluster luego de instalado, entre estos plugins se encuentran el dashboard UI de Kubernetes, la herramienta de paquetes Helm, el proxy reverso y balanceador de carga interior al cluster llamado Ingress, entre otras.

1.2.1. Requerimientos de hardware

Para poner en funcionamiento cada nodo del cluster se recomienda tener en cada máquina virtual 4GB de memoria RAM y 20GB de almacenamiento según los requerimientos dados por Microk8s[18]. Sin embargo, según el reporte de algunos usuarios es posible hacerlo funcionar con menores capacidades[42].

1.2.2. Instalación del cluster

En la guía que se presenta en esta sección se realiza una instalación reducida respecto al despliegue presentado en la Subsección 2.4.3, en donde en lugar de utilizar seis máquinas virtuales para alojar tres nodos maestro y tres nodos trabajadores, se utilizan solamente tres máquinas virtuales que hacen de nodo maestro y nodo trabajador a la vez. Por otra parte se utilizan dos máquinas virtuales para el proxy y balanceador de carga, luego una máquina virtual más para la base de datos y una última para el servidor NFS.

1.2.2.1. Especificación del ambiente de partida

Habiendo mencionado lo anterior, se muestra en la Tabla 1.1 una lista con cada VM utilizada, su IP asignada y su propósito en el ambiente creado. En siguientes secciones se comenzará la instalación de los distintos softwares usados en cada máquina virtual para cumplir con su rol asignado.

Hostname	Dirección IP	Función/Rol
	192.168.1.a	IP virtual para los Load balancer
load-balancer-1	192.168.1.b	HAProxy/Keepalived Master
load-balancer-2	192.168.1.c	HAProxy/Keepalived Backup
kube-node-1	192.168.1.d	Master/Worker node
kube-node-2	192.168.1.e	Master/Worker node
kube-node-3	192.168.1.f	Master/Worker node
tb-postgres	192.168.1.g	Base de datos PostgreSQL
nfs-server	192.168.1.h	Servidor de archivos NFS

Tabla 1.1: Lista de las máquinas virtuales utilizadas para la instalación de cluster.

1.2.2.2. Configuración de IPs estáticas en las máquinas virtuales

En este apartado se detalla un paso a realizar previo a las instalaciones. Luego de creadas las máquinas virtuales es conveniente desactivarles el protocolo DHCP y asignarles una IP estática, para hacer esto se debe modificar el archivo *00-installer-config.yaml* en cada una de las máquinas virtuales usando los comandos mostrados a continuación.

```
$ sudo nano /etc/netplan/00–installer–config.yaml
```

Listing 1.2: Comando para abrir el archivo de texto a modificar

Al abrir el archivo de texto se debe ingresar el siguiente contenido.

```
network:
  version: 2
  ethernets:
    enp0s3:
      dhcp4: no
      addresses:
       - <ip-fija-para-vm>/24
      routes:
       - to: default
         via: <ip-enrutador-gateway>
```

Luego de modificar y salvar el archivo de configuración, se aplican los cambios con los comandos:

```
$ sudo netplan apply
$ sudo reboot
```

Listing 1.3: Comandos para aplicar los cambios realizados en el archivo de netplan

Al reiniciar la VM, se puede comprobar que la nueva IP fue asignada con el siguiente comando.

```
$ ip a
```

Listing 1.4: Comando de comprobación de IP

1.2.2.3. Instalacion de HAProxy y Keepalived en los balanceadores de carga

En esta sección se detalla la instalación y despliegue de las máquinas virtuales encargadas de balancear la carga y redirigirla hacia el cluster de Kubernetes. Como se detalla en la arquitectura de la solución, el balanceador de carga está compuesto por dos máquinas virtuales que comparten una misma dirección IP llamada IP virtual o IP flotante.

Se comienza instalando el software necesario:

```
$ sudo apt–get install keepalived haproxy psmisc –y
```

Listing 1.5: Instalación de HAProxy y Keepalived

Configuracion de HAProxy: Seguido de instalar el software, primero es necesario configurar el proxy, para esto es necesario modificar el archivo */etc/haproxy/haproxy.cfg*. Este archivo de configuración dispone de cuatro secciones principales, *Global*, *Defaults*, *Frontend* y *Backend* que se detallan a continuación.

- ***Global:*** Es la sección que contiene definiciones de seguridad y performance que afectan a todos los procesos de HAProxy.
- ***Defaults:*** Es usada para reducir la duplicación, la configuración declarada en esta sección aplica a las siguientes secciones *Frontend* y *Backend*.
- ***Frontend:*** Es usada para definir las direcciones IP y los puertos a los que los clientes se pueden conectar cuando HAProxy es usado como un proxy reverso.

- ***Backend:*** Es usada para definir los servidores a los que se les balancea la carga y redireccionan los request.

```
$ sudo nano /etc/haproxy/haproxy.cfg
```

Listing 1.6: Comando para abrir el archivo de configuración

La única sección que será necesario modificar es la sección *Backend*, en donde habrá que agregar los tres nodos trabajadores del clúster con su respectivas direcciones IP.

```
(...)
server kube-node-1 192.168.1.d:80 check
server kube-node-2 192.168.1.e:80 check
server kube-node-3 192.168.1.f:80 check
```

Finalmente aplicamos los cambios con los siguientes comandos:

```
$ systemctl restart haproxy
$ systemctl enable haproxy
```

Listing 1.7: Reinicio HAProxy para aplicar los cambios

Configuración de Keepalived: Keepalived será el encargado de asignar la IP flotante entre los nodos del balanceador de carga. Para esto es necesario configurar el archivo */etc/keepalived/keepalived.conf*.

En el caso de estar ubicados en la máquina virtual *load-balancer-1* de nuestro despliegue, los campos a modificar son los siguientes:

1. El campo *interface* para matchear el nombre de la interfaz de red que tenga el terminal.

```
interface ens18
```

2. El campo *unicast_src_ip* para matchear la IP del terminal.

```
unicast_src_ip 192.168.1.b
```

3. El objeto *unicast_peer* en donde se deberán agregar las IPs de cada máquina virtual que conforme el balanceador de carga a excepción de la VM actual (es decir, en nuestro caso habrá una sola IP perteneciente a la otra máquina virtual *load-balancer-2* dentro del balanceador de carga).

```
unicast_peer {
    192.168.1.c
}
```

4. El objeto *virtual_ipaddress* donde se ingresa la IP flotante definida en la arquitectura.

```
virtual_ipaddress {
    192.168.1.a
}
```

Finalmente aplicamos los cambios con los siguientes comandos:

```
$ systemctl restart keepalived
$ systemctl enable keepalived
```

Listing 1.8: Reinicio Keepalived para aplicar los cambios

Comprobación de la alta disponibilidad: Luego de realizado el proceso de los puntos anteriores, en cada VM perteneciente al balanceador de carga se puede comprobar que efectivamente la IP flotante migra de una máquina virtual a otra una vez que la máquina que posee esta IP queda indisponibilizada. Para probar esto se baja la máquina virtual que tiene asignada la IP flotante (esto puede comprobarse con el comando *ip a*).

```
$ systemctl stop haproxy
```

Listing 1.9: Detener el proceso HAProxy

Allí la IP flotante debe desaparecer de la lista de direcciones del host y al acceder a la otra VM perteneciente al load balancer se deberá ver la IP flotante asignada automáticamente.

1.2.2.4. Instalación del servidor NFS

Para comenzar la instalación del servidor NFS se ejecutan los comandos presentados a continuación con el fin de instalar el software necesario, luego es preciso crear y modificar los permisos de la carpeta dentro del filesystem que servirá como almacenamiento de los datos. En el Listing 1.10 se muestran los cuatro comandos necesarios.

```
$ sudo apt–get install nfs–kernel–server
$ sudo mkdir –p /srv/nfs/kubedata
$ sudo chown nobody:nogroup /srv/nfs/kubedata
$ sudo chmod 0777 /srv/nfs/kubedata
```

Listing 1.10: Instalación del software y creación del directorio de almacenamiento

Seguido de esto es necesario editar el archivo */etc/exports* para agregar las IPs de las VMs que accederán al servidor y por último reiniciamos el servicio. Notar que el comando *echo* del Listing 1.11 se encuentra en múltiples líneas por el simple hecho de poder visualizarlo bien al ser largo.

```
$ sudo mv /etc/exports /etc/exports.bak
$ echo
  '/srv/nfs/kubedata␣192.168.1.0/24(rw,sync,no_subtree_check)'
  | sudo tee /etc/exports
$ sudo systemctl restart nfs–kernel–server
```

Listing 1.11: Comandos para agregar las VMs que tendrán acceso al servidor

Luego de esto el servidor NFS ya esta funcional, para poder corroborar el correcto funcionamiento del mismo desde las máquinas virtuales que conforman el cluster se puede intentar montarlo con el primer comando del Listing 1.12. Si no hubieron errores, se desmonta con el segundo comando.

```
$ sudo mount –t nfs 192.168.1.h:/srv/nfs/kubedata /mnt
$ sudo umount /mnt
```

Listing 1.12: Montura y desmontaje del servidor NFS

1.2.2.5. Instalación de Kubernetes

En esta sección comenzamos con la creación e interconexión de los nodos que conforman el cluster.

Requisitos previos: El paso previo a realizar la creación del cluster consiste en modificar el archivo */etc/hosts* de cada una de las máquinas virtuales que compondrán el cluster. Se deben agregar las IPs del resto de máquinas junto con su hostname y eliminar la entrada preexistente del tipo *127.0.0.1 <hostname-de-vm>*. El resultado de cada archivo hosts se verá similar al siguiente.

```
127.0.0.1 localhost
192.168.1.d kube-node-1
192.168.1.e kube-node-2
192.168.1.f kube-node-3
(...)
```

Por otro lado es necesario instalar el software que soporta la conexión con el servidor NFS cuya integración en el cluster se muestra en los siguientes pasos. El repositorio necesario es *nfs-kernel-server*.

```
sudo apt install nfs–kernel–server
```

Listing 1.13: Instalación del software para conectarse al servidor NFS

Creación y conexión de los nodos del cluster: Para comenzar con la creación de los nodos, en el primero de ellos se ejecuta el comando del Listing 1.14.

```
$ microk8s add–node
```

Listing 1.14: Comando para crear el cluster

Este comando realiza la creación de un nodo maestro para el cluster y al finalizar disponibiliza un comando del tipo *microk8s join* el cual deberá ser copiado y pegado en el resto de VMs para ir uniéndose al cluster.

```
$ microk8s join <ip–master>:<puerto>/<token>
```

Listing 1.15: Template del comando join de Microk8s usado en cada nodo

Se le puede agregar la flag –worker al final si se quiere que el nodo a agregar solo cumpla el rol de worker node, para más información sobre este procedimiento referirse a la guía oficial [15].

Nota: Puede ser necesario ejecutar los siguientes comandos del Listing 1.16 antes del comando de join para modificar los permisos.

```
$ sudo usermod –a –G microk8s <usuario>
$ sudo chown –f –R <usuario> ~/.kube
$ newgrp microk8s
```

Listing 1.16: Comandos de permisos para Microk8s

Luego de realizar el *join* se puede comprobar que el nodo se haya unido correctamente volviendo al nodo en donde se inició el cluster y ejecutando el comando del Listing 1.17 el cual debe mostrar una lista con todos los nodos que hayamos unido.

```
$ microk8s kubectl get nodes
```

Listing 1.17: Comando de comprobación de los nodos unidos al cluster

Teniendo el cluster creado y conectado entre las distintas máquinas virtuales, procedemos a instalar el complemento *Ingress* como un addon de Microk8s para aprovisionar en nuestro cluster los recursos necesarios para poder exponer servicios.

```
$ microk8s enable dns
$ microk8s enable ingress
```

Listing 1.18: Comandos de instalación del complemento Ingress

Despliegue de recursos para integrar el servidor NFS: Para integrar el servidor NFS al cluster es necesario desplegar el aprovisionador NFS del repositorio *kubernetes-sigs/nfs-subdir-external-provisioner*[52]. Los archivos de despliegue son: *class.yaml*, *deployment.yaml* y *rbac.yaml*.

Este aprovisionador NFS permitirá a los despliegues dentro del cluster realizar claims de volúmenes persistentes de Kubernetes (PVC) y además realizar la creación del respectivo volumen persistente a ser claimeado de manera dinámica (dynamic provisioning). Esto facilita al administrador el tener que estar declarando manualmente los volúmenes persistentes mediante archivos de configuración yaml.

```
$ microk8s kubectl apply –f rbac.yaml
```

Listing 1.19: Aplicación del archivo rbac.yaml

Luego para aplicar el archivo *class.yaml* hay que agregar la annotation *is-default-class* para forzar que la *StorageClass* a desplegar sea la definida por defecto y así no tener que referenciarla en cada persistent volume claim, notar que en ThingsBoard no tenemos control sobre estos claims por lo que es necesario realizar este cambio.

```
(...)
metadata:
  name: nfs-client
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
(...)
```

Luego aplicamos el archivo con el siguiente comando:

```
$ microk8s kubectl apply –f class.yaml
```

Listing 1.20: Aplicación del archivo class.yaml

Por último, para aplicar el archivo *deployment.yaml* hay que modificar la IP de la máquina virtual en donde tenemos el servidor NFS (192.168.1.h) y el path del directorio que se utilizará para almacenar los datos en el mismo (src/nfs/kubedata).

```
          (...)
          env:
        - name: PROVISIONER_NAME
          value: k8s-sigs.io/nfs-subdir-external-provisioner
        - name: NFS_SERVER
          value: 192.168.1.h
        - name: NFS_PATH
          value: /srv/nfs/kubedata
  volumes:
    - name: nfs-client-root
      nfs:
        server: 192.168.1.h
        path: /srv/nfs/kubedata
```

```
$ microk8s kubectl apply –f depoyment.yaml
```

Listing 1.21: Aplicación del archivo deployment.yaml

Una vez aplicados los tres archivos se puede pasar a comprobar que el aprovisionamiento dinámico de volúmenes persistentes está funcionando correctamente usando el persistent volume claim de prueba llamado *test-claim.yaml*. Lo aplicamos y luego comprobamos la lista de PV y PVC para ver que se haya aprovisionado correctamente.

```
$ microk8s kubectl apply –f test–claim.yaml
$ microk8s kubectl get pv, pvc
```

Listing 1.22: Comandos de prueba del PVC

En caso de que se muestre con estado *pending* de forma indefinida, se puede ver mejor el error con el siguiente comando:

```
$ microk8s kubectl describe pvc <nombre–pvc>
```

Listing 1.23: Comando para visualizar el estado del PVC

1.2.3. Creación de la base de datos PostgreSQL remota

Para tener la base de datos resguardada fuera de la DMZ, se debe crear en una nueva máquina virtual que sea distinta a las utilizadas para el cluster. En esta nueva máquina virtual procedemos a crear la base de datos utilizando los scripts de la documentación oficial de ThingsBoard usados en la guía de despliegue en Ubuntu[10].

Los comandos a utilizar de esta documentación son solo los necesarios para crear la base de datos los cuales se listan en el Listing 1.24. Notar que también en este caso que algunos comandos fueron expresados en múltiples filas para mejorar la visualización, al momento de ejecutarlos deben abarcar una sola línea.

```
# install **wget** if not already installed:
$ sudo apt install –y wget

# import the repository signing key:
$ wget ––quiet –O
  – https://www.postgresql.org/media/keys/ACCC4CF8.asc
  | sudo apt–key add –

# add repository contents to your system:
$ RELEASE=$(lsb_release –cs)
$ echo
"deb␣http://apt.postgresql.org/pub/repos/apt/␣${RELEASE}"–pgdg
main | sudo tee /etc/apt/sources.list.d/pgdg.list

# install and launch the postgresql service:
```

```
$ sudo apt update
$ sudo apt –y install postgresql–12
$ sudo service postgresql start
```

Listing 1.24: Comandos de instalación de la base de datos PostgreSQL

Como la documentación nos sugiere, cambiamos la clave del usuario postgre usando los comandos del Listing 1.25.

```
$ sudo su – postgres
$ psql
$ \password
$ \q
```

Listing 1.25: Cambio de clave del usuario de PostgreSQL

Finalmente para aceptar conexiones remotas, se deben modificar los archivos *postgresql.conf* y *pg_hba.conf*, ambos se encuentran en el directorio */etc/postgresql/12/main/*.

En *postgresql.conf*, para permitir conexiones entrantes al servidor hay que descomentar y modificar la línea *listen_addresses* como se muestra a continuación:

```
listen_addresses='*'
```

En *pg_hba.conf*, para controlar las IPs que tienen acceso a la base hay que agregar las direcciones IP de los nodos del cluster que accederán a la base de la siguiente forma:

```
host all all <ip_node_1>/32 md5
(...)
host all all <ip_node_n>/32 md5
```

Por último reiniciamos el servicio para aplicar los cambios.

```
$ sudo service postgresql restart
```

Listing 1.26: Reinicio de PostgreSQL para aplicar los cambios

Para comprobar que se puede acceder a la base de datos remotamente desde las máquinas virtuales que conforman nuestro cluster se puede utilizar el siguiente comando:

```
$ psql –U <username_bd> –h <ip_bd_remota>
```

1.2.4. Despliegue de ThingsBoard en el cluster multi-nodo

1.2.4.1. Preparación de los archivos de despliegue de ThingsBoard

Para llevar a cabo el despliegue se debe clonar el repositorio en donde están los scripts para desplegar ThingsBoard en Minikube[96], estos scripts de despliegue deberan ser ligeramente modificados para que podamos respetar la arquitectura propuesta y la herramienta utilizada para manejar Kubernetes que en nuestro caso es Microk8s en lugar de Minikube.

Entre los archivos del repositorio de instalación de ThingsBoard a modificar se encuentran: el archivo de configuración de la base de datos *tb-node-db-configmap.yml* y todos los archivos de formato bash script *.sh* dentro del repositorio (*k8s_install_-tb*, *k8s_deploy_thirdparty.sh*, *k8s_deploy_resources.sh*, etc). Estos cambios deben hacerse antes de ejecutar el script *k8s_install_tb.sh* el cual es el que se encarga de desplegar completamente ThingsBoard en el cluster.

Siendo más precisos, es necesario realizar tres cambios principales:

1. Referenciar la base de datos remota que creamos anteriormente
2. Omitir la instalación de la base de datos interna que se instancia en el cluster por defecto para ThingsBoard
3. Agregar el prefijo *microk8s* en cada sentencia *kubectl* que halla en los archivos de bash script *.sh* dentro del repositorio para aclarar que queremos utilizar nuestra herramienta de gestión de Kubernetes

Para completar el **punto 1.**, en el archivo *tb-node-db-configmap.yml* se debe modificar el valor del campo *SPRING_DATASOURCE_URL* para referenciar nuestra base de datos remota, a continuación se muestra entre los signos de mayor y menor (<>) los valores que se deben cambiar.

```
data:
  DATABASE_TS_TYPE: sql
  SPRING_JPA_DATABASE_PLATFORM:
    org.hibernate.dialect.PostgreSQLDialect
  SPRING_DRIVER_CLASS_NAME: org.postgresql.Driver
  SPRING_DATASOURCE_URL:
    jdbc:postgresql://<ip_bd_remota>:5432/thingsboard
  SPRING_DATASOURCE_USERNAME: <username_bd>
```

```
SPRING_DATASOURCE_PASSWORD: <clave_bd>
```

Para completar el **punto 2.**, en el archivo *k8s_install_tb.sh* se debe eliminar la línea 86 que ejecuta la función *installPostgress* como se muestra a continuación.

```
case $DATABASE in
        postgres)
            installPostgres <= ELIMINAR ESTA LINEA
            installTb ${loadDemo}
        ;;
        hybrid)
            installPostgres
            installCassandra
            installTb ${loadDemo}
        ;;
        *)
```

Para completar el **punto 3.**, en cada archivo de extensión *.sh*, se debe agregar el prefijo *microk8s* en cada ocurrencia de la instrucción *kubectl*. Es decir, si se halla una instrucción *kubectl apply -f file.yml*, esta terminará siendo *microk8s kubectl apply -f file.yml*.

1.2.4.2. Despliegue de ThingBoard

Una vez se hayan modificado los archivos ya estamos en condiciones de desplegar la aplicación en nuestro cluster. Para ello, se ejecutan los shell script que se encuentran en el directorio de ThingsBoard como se muestra en el Listing 1.27.

```
$ ./k8s-install-tb.sh
$ ./k8s-deploy-thirdparty.sh
$ ./k8s-deploy-resources.sh
```

Listing 1.27: Comandos de deploy de ThingsBoard

Si todo se instaló sin errores, debe ser posible comprobar la instalación con comandos kubectl para obtener los namespaces, deployments, pods e ingress como se muestran en el Listing 1.28.

```
$ microk8s kubectl get pods
$ microk8s kubectl get deployments
$ microk8s kubectl get namespaces
```

Listing 1.28: Comandos de chequeo de la instalación de ThingsBoard

1.2.4.3. Posibles errores al ejecutar los scripts de instalación de ThingsBoard

Error 1: Error from server: error dialing backend: dial tcp: lookup worker-node-3: Temporary failure in name resolution

Esto se puede solucionar modificando el archivo */etc/hosts* del nodo/os maestro/os, agregando en este archivo las entradas que mapean las direcciones IPs de cada nodo worker con su respectivo hostname.

```
<ip_worker_1> nombre-nodo-worker-1
(...)
<ip_worker_n> nombre-nodo-worker-n
```

Por ejemplo:

```
192.168.1.20 worker-node-3
```

Error 2: Readiness probe failed: Get "http://10.1.135.161:8080/login": dial tcp 10.1.135.161:8080: connect: connection refused

Si al ejecutar un comando *describe* de algún pod que haya quedado en estado pending, o en estado running pero con en 0/1 instancias y además el error que se muestra al final de los logs es el mencionado, es muy posible que se deban aumentar los valores en los campos *initialDelaySeconds* de los archivos .yaml de las componentes que estén dando el problema. Una vez hecho el aumento, volver a intentar desplegar dichos archivos.

Error 3: failed calling webhook "validate.nginx.ingress.kubernetes.io": failed to call webhook: Post "https://ingress-nginx-controller-admission.ingress-nginx.svc:443/networking/v1/ingresses?timeout=10s"

Este error puede surgir al querer aplicar el archivo *routes.yaml* cuando se están instalando los resources de ThingsBoard, es decir en el tercer comando del Listing 1.27, para solucionarlo ejecutar el siguiente comando.

```
$ microk8s kubectl delete –A ValidatingWebhookConfiguration
ingress–nginx–admission
```

1.3. Exposición de ThingsBoard por HTTPS

En la guía de instalación se realiza la exposición de la aplicación con HTTP por medio de HAProxy, recordamos que con HTTP los mensajes viajan en texto plano sin cifrado, siendo susceptible a ataques. En esta sección se muestran los pasos necesarios para pasar a HTTPS lo cual permite una comunicación cifrada entre el cliente y el servidor. Para llevar esto a cabo es necesario la generación de los certificados (auto firmados) así como la configuración de HAProxy para utilizarlos.

Esta sección está basada en la guía de DigitalOcean *How To Implement SSL Termination With HAProxy on Ubuntu 14.04*[26], pero usando las versiones actualizadas del software nombrado en ella.

1.3.1. Generación de los certificados

Para crear el certificado público y la key privada y luego combinarlos y pegarlos en el directorio final, se utilizan los comandos del Listing 1.29 ubicados en el directorio del usuario Linux para no tener problemas de permisos. Notar que el primer comando esta cortado en tres líneas.

```
$ sudo openssl req –x509 –nodes –days 9999 –newkey
rsa:2048 –keyout /etc/ssl/private/certif.key
–out /etc/ssl/private/certif.crt
$ cat certif.crt certif.key > combined.pem
$ sudo cp combined.pem /etc/ssl/private/
```

Listing 1.29: Comandos de creación

1.3.2. Referencia del certificado combinado desde HAProxy

Una vez se tiene el certificado combinado *combined.pem* en el directorio */etc/ssl/private*, se deben ingresar los distintos campos en el archivo haproxy.cfg como se denota en la guía. A continuación se muestra el archivo final haproxy.cfg usado para

el despliegue del prototipo monolítico señalando cada línea agregada, esto posiblemente aplique de la misma forma para cada nodo del loadbalancer en el despliegue de cluster pero no fue comprobado.

```
#HA Proxy Config
global
 maxconn 2048                       // <= Agregar esta linea
 tune.ssl.default-dh-param 2048     // <= Agregar esta linea

defaults
 log global
 mode http
 option forwardfor                  // <= Agregar esta linea
 option http-server-close           // <= Agregar esta linea
 timeout connect 5000ms
 timeout client 50000ms
 timeout server 50000ms
 timeout tunnel  1h
 default-server init-addr none
 stats enable                       // <= Agregar esta linea
 stats uri /stats                   // <= Agregar esta linea
 stats realm Haproxy\ Statistics    // <= Agregar esta linea
 stats auth user:password           // <= Agregar esta linea

frontend www-http
 bind 0.0.0.0:80
 reqadd X-Forwarded-Proto:\ http
 default_backend www-backend

frontend www-https      // <= Agregar este bloque
 bind 0.0.0.0:443 ssl crt /etc/ssl/private/combined.pem
 reqadd X-Forwarded-Proto:\ https
 default_backend www-backend

backend www-backend
  balance leastconn
```

```
server tb1 127.0.0.1:8080 check inter 5s
http-request set-header X-Forwarded-Port %[dst_port]
http-request set-header X-Forwarded-Proto https if { ssl_fc }
http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
```

Por último reiniciamos HAProxy para aplicar los cambios.

```
$ systemctl restart haproxy
$ systemctl enable haproxy
```

Listing 1.30: Reinicio HAProxy para aplicar los cambios

1.4. Guía basica de uso de ThingsBoard

En esta sección se muestran algunos procedimientos de funcionalidades de ThingsBoard utilizadas durante el proyecto con el fin de facilitar el abordaje a la plataforma a personas que no la hayan usado antes.

1.4.1. Clases de usuarios y creación de nuevos usuarios

Como se menciona en la Sección 2.3, ThingsBoard cuenta con tres tipos de usuarios que se mapean a tres niveles de permisos sobre la plataforma.

1. **Usuario administrador:** Esta clase de usuario tiene la capacidad de gestionar usuarios tenant.
2. **Usuario tenant:** Esta clase de usuario tiene la capacidad de gestionar un conjunto de usuarios cliente, creándolos y asignándoles dispositivos y demás entidades de ThingsBoard.
3. **Usuario cliente:** Esta clase de usuario puede visualizar y gestionar los recursos que le son asignados por el tenant correspondiente.

En el marco del proyecto se utilizó un único usuario tenant para la creación de los dispositivos que recolectan los datos.

1.4.1.1. Creación de un usuario tenant

Al loguearse como un usuario administrador, se debe ir a la sección *Propietarios* (Figura 1.1 Izq.). Para crear un nuevo grupo de usuarios tenant se debe utilizar el icono **+** (Figura 1.1 Med.) y una vez creado el grupo se accede a él (Figura 1.1 Der.)

para finalmente crear el usuario en sí usando nuevamente el ícono + (Figura 1.2 Izq.). Para crear el usuario se inserta como mínimo un mail y la opción *Mostrar enlace de activación* (Figura 1.2 Med.), al crearlo se generará un enlace de activación que debe ser entregado al dueño del nuevo usuario creado para que ingrese su propia contraseña (Figura 1.2 Der.).

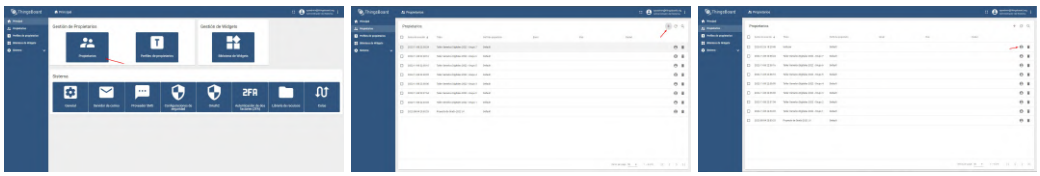

Figura 1.1: ***Capturas de pantalla con la guía de creación de usuarios tenant parte 1.***

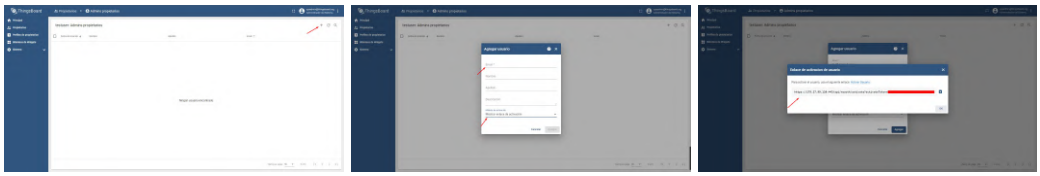

Figura 1.2: ***Capturas de pantalla con la guía de creación de usuarios tenant parte 2.***

1.4.2. Creación y gestión de nuevos dispositivos

Al loguearse como un usuario tenant se debe ir a la sección dispositivos (Figura 1.3 Sup. Izq.) en donde se desplegará la lista de dispositivos asociados al usuario tenant actual, allí se debe utilizar el ícono + y seguido de la opción *Agregar nuevo dispositivo* para crear uno nuevo (Figura 1.3 Sup. Med.). En el modal que surge basta con ingresar el nombre del dispositivo y dar al botón agregar (Figura 1.3 Sup. Der.). Una vez creado el dispositivo aparecerá en la lista de la pantalla, al ser cliqueado se abrirá un panel a la derecha de la pantalla con toda la información del dispositivo, encontramos allí el access token necesario para enviar datos a la plataforma por HTTP y HTTPS, el ID del dispositivo, la opción de asignar el dispositivo a un usuario cliente y en la pestaña *Última telemetría* podemos observar los últimos datos recibidos por este dispositivo (Figura 1.3 Inf. Med.).

1.4.3. Creación de paneles de datos en tiempo real

Para crear un panel con gráficos que muestre en tiempo real los datos que van llegando de un dispositivo, se debe ingresar con un ususario tenant e ir a la sección

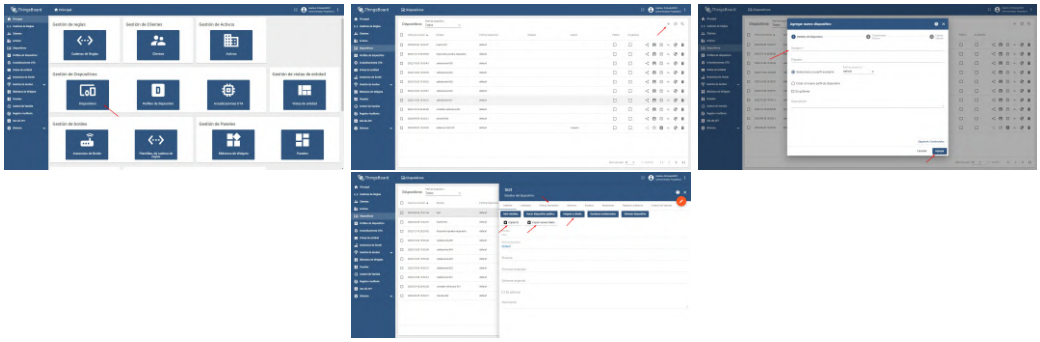

Figura 1.3: ***Capturas de pantalla con la guía de creación de dispositivos.***

Paneles (Figura 1.4 1ª fila Izq.). En esta sección nuevamente se usa el icono + y luego la opción *Crear nuevo panel* para crear uno nuevo (Figura 1.4 1ª fila Med.). En el modal que surge basta con ingresar un título y dar al botón agregar (Figura 1.4 1ª fila Der.).

Para ingresar al panel se debe cliquear en el ícono con rectángulos grises del panel correspondiente en la lista de paneles (Figura 1.4 2ª fila Izq.), al hacerlo se abrirá el panel vacío. Utilizando el botón anaranjado de edición se accede al menú de customización (Figura 1.4 2ª fila Med.). Dentro del menú de customización cliqueando en *Agregar nuevo widget* se desplegará el menú a la derecha de la pantalla con todas las opciones de widgets disponibles. Seleccionamos *Charts* para insertar una gráfica de telemetría en función de tiempo (Figura 1.4 2ª fila Der.).

Una vez seleccionado el tipo de gráfica, se procede a agregar un set de datos con el botón azul de agregar, al hacerlo se pedirá que la entidad a agregar tenga un alias (Figura 1.4 3ª fila Izq.), si no tenemos uno creado, le damos a crear. En el modal de creación del alias ingresamos los datos mostrados en (Figura 1.4 3ª fila Med.) para poder setear el alias a un dispositivo en particular que en nuestro caso es *huerta-001*. Finalmente con el alias seleccionado, elegimos el dato de este dispositivo que queremos graficar, en nuestro caso es *max_light* (Figura 1.4 3ª fila Der.). Al aceptar tendremos nuestra gráfica creada, utilizando el icono del lápiz sobre ella se podrán editar características de la misma y con el botón anaranjado del tick se confirman y guardan los cambios en el panel (Figura 1.4 4ª fila Med.).

1.4.4. Obtener un bearer token

Para obtener un bearer token se debe ingresar al ícono de *más opciones* en la esquina superior derecha del dashboard (Figura 1.5 Izq.) y luego la opción *Seguridad* (Figura 1.5 Med.), finalmente se obtiene el token cliqueando en el botón *Copiar*

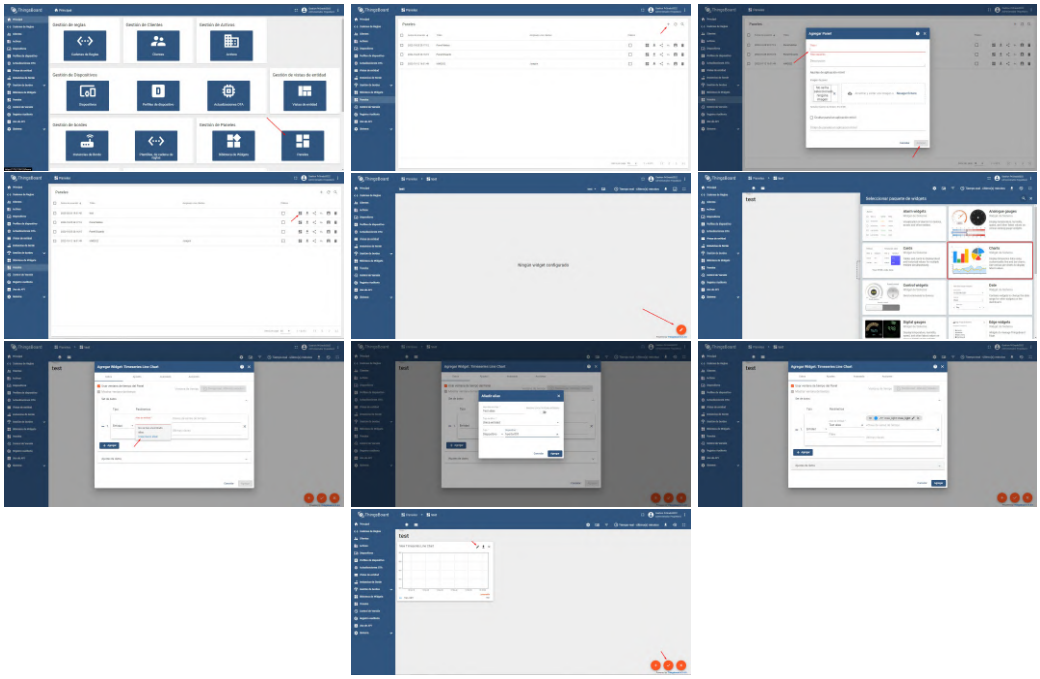

Figura 1.4: ***Capturas de pantalla con la guía de creación de paneles.***

JWT (Figura 1.5 Der.).

Figura 1.5: ***Capturas de pantalla con la guía de obtención de bearer token.***

1.4.5. Procesamiento de mensajes con el Rule Engine

Para modificar los pasos del Rule Engine al recibir un mensaje nuevo se debe ingresar a sección *Cadena de Reglas* (Figura 1.6 Izq.), una vez allí se selecciona el Rule Engine a modificar y en el panel que se despliega a la derecha de la pantalla se selecciona *Abrir cadena de reglas* (Figura 1.6 Med.) para acceder al menú de customización. Dentro del menú se puede observar a la izquierda una lista con los nodos que se pueden usar en el motor y a la derecha el grafo modificable del mismo (Figura 1.6 Der.), en donde en particular se remarca en rojo sobre la imagen un flujo alternativo creado manualmente para atender los Remote Procedure Calls (RPC) que se detallan en la siguiente subsección. Este flujo de RPCs es un caso particular pero existe una extensa lista de reglas posibles.

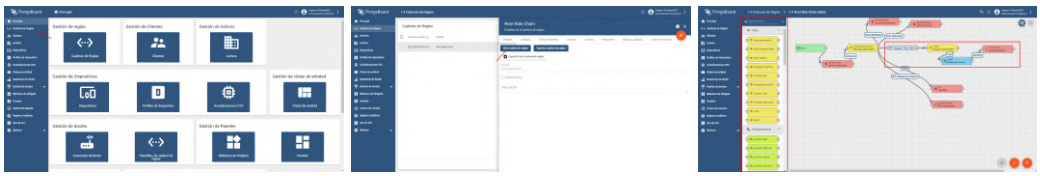

Figura 1.6: ***Capturas de pantalla con la guía de modificación del Rule Engine.***

1.4.5.1. Remote Procedure Calls (RPC) de tipo client-side

En esta sub sección se detalla el prototipo realizado para probar los Remote Procedure Calls (RPC) de carácter *client-side*, o dicho de otra forma activados por el dispositivo al realizar un envío a un endpoint especial de ThingBoard. Este RPC permite a un dispositivo obtener una respuesta creada desde el servidor por medio del response del mensaje enviado hacia este con el fin de poder tomar una determinada acción en el dispositivo a partir de la respuesta del servidor.

En el caso que presentamos a continuación se creó un prototipo básico en donde el sensor envía un dato de temperatura y ThingsBoard le responde si esta es aceptable o si pasó de los 50°C. En caso de estar en el segundo escenario el dispositivo podría tomar medidas al respecto, por ejemplo encender un sistema de refrigeración.

Comenzando por la configuración necesaria del lado del Rule Engine, se observa en la (Figura 1.7 Izq.) que el procesamiento tiene tres nodos (amarillo, celeste y por último rojo). En el nodo amarillo (Figura 1.7 Med.) se tiene un script escrito en JavaScript que comprueba el campo *method* del mensaje recibido con el fin de determinar que RPC es el que se desea activar. En este caso el RPC es *prueba-rpc-cliente* y como se observa en ambas fotos de la Figura 1.8, los mensajes enviados por Postman tienen ese mismo nombre en el campo *method*. El recuadro celeste (Figura 1.7 Der.) tiene una función JavaScript que se encarga de tomar el valor de temperatura recibido y crear el mensaje de respuesta en base a él, nuevamente se puede ver en ambas fotos de la Figura 1.8 que según el valor de temperatura enviado en el mensaje, la respuesta del servidor varía según esté por debajo o por encima de 50. Por último vale la pena señalar que como se ve en la Figura 1.8, la URL utilizada para activar los RPC no es la misma que se usa para enviar telemetría.

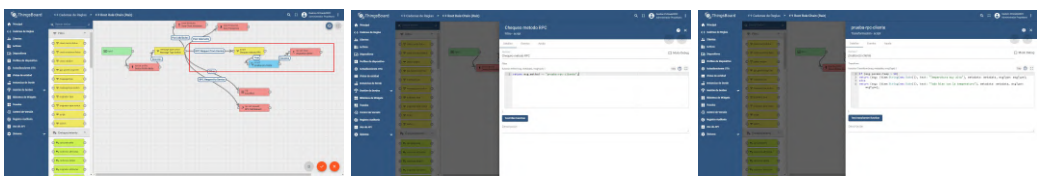

Figura 1.7: ***Capturas de pantalla con la guía del RPC client-side.***

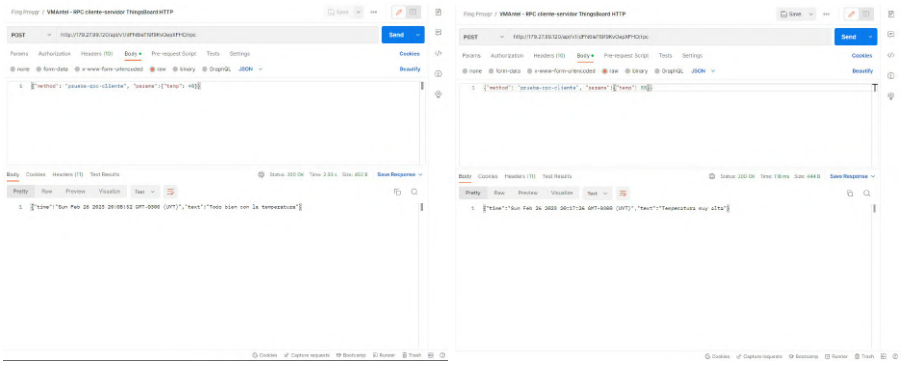

Figura 1.8: ***Capturas de pantalla del llamado en Postman del RPC client-side.***

1.5. Guía básica de uso del script de descarga de CSV de ThingsBoard

Se desarrolló un script en Node.js que permite la descarga desde ThingsBoard de los datos de un dispositivo durante un período de tiempo determinado en formato CSV y JSON. Para poder utilizar dicho script basta con llevar a cabo los tres pasos detallados a continuación.

1. **Instalación de dependencias:** Este paso es el más sencillo y consiste en ejecutar el comando *npm i* en el directorio raíz del proyecto del script (donde se encuentre el archivo package.json).
2. **Seteo de las variables de ejecución:** en el archivo *parametros_generales.js* hay una serie de variables que deben ser asignadas para poder llamar a la API de ThingsBoard y obtener los datos, estas variables se listan a continuación en el orden en que aparecen.
    - **JWT:** Este string contiene el bearer token que provee ThingsBoard, para saber como obtenerlo ver la Subsección 1.4.4.
    - **fechaInicio y fechaFinal:** Estos dos strings deben determinar el intervalo de tiempo del que se desea obtener mediciones, deben ser ingresados con el formato *YYYY-MM-DDThh:mm:ss*.
    - **idDispositivo:** Este string contiene el ID del dispositivo brindado por ThingsBoard, para saber como obtenerlo ver la Figura 1.3 (Inf. Med.).
    - **keys:** Este string contiene los nombres de los datos medidos por el dispositivo que se desean obtener, deben ser ingresados con el formato *claveDato1,claveDato2,...,claveDatoN*.

- **limite:** Este int determina el máximo número de datos a obtener, en general no se debe modificar.
- **endpointThingsboard:** Este string contiene la URL de la plataforma incluyendo hasta el puerto (de ser necesario especificarlo).

Nota: En el archivo *parametros_generales_ejemplo.js* se puede ver un ejemplo de las variables utilizadas para extraer datos durante el proyecto.

3. **Ejecución:** Se ejecuta el script con el comando *node index.js*, si todo funcionó correctamente en el directorio *salida-script* se podrán hallar los archivos con los datos bajo el nombre de: *datos-sensor-<segmento_inicial_de_-id_del_disp>-<YYYYMMDDThhmmss_de_inicio>-<YYYYMMDDThhmmss_de_fin>.[csv | json]*.

Apéndice 2

Procedimientos del capítulo 3

2.1. Estructura de proyecto PlatformIO de los sensores basados en ESP32

Los sensores basados en el chip ESP32 son: el sensor del caso uno (ambiente y calidad de aire), del caso tres (estación meteorológica) y del caso cuatro (sensor de huerta). A nivel de código estos tres sensores comparten la misma estructura en sus proyectos, variando únicamente en las funciones de las bibliotecas específicas de los sensores que utilizan así como las funciones que generan el objeto JSON a enviar en el caso de utilizar WiFi (*inicializarJsonVacio*) o la que crea la tira de caracteres que conforma el mensaje en caso de enviar por LoRa (*crearPaqueteCharArray*). Además de esto hay una única excepción en el sensor de estación meteorológica en el que hay una ligera variación en la estructura general, en este sensor a diferencia de los otros dos se utilizan los dos núcleos. El núcleo 0 que en los otros dos sensores no se utiliza, en la estación meteorológica se usa para escuchar las interrupciones generadas por el sensor de lluvia al activarse.

Los proyectos están desarrollados en C++ y dentro de cada uno de ellos encontramos encontramos una serie de directorios y un archivo principal llamado *platformio.ini* el cual contiene información general para compilar y grabar el proyecto en el dispositivo. Entre esta información encontramos los frameworks y bibliotecas de desarrollo usadas en el proyecto, el modelo de la placa a utilizar, el puerto serial al que está conectada la placa, las dependencias externas instaladas, las flags de build, etc.

Luego dentro de los directorios del proyecto encontramos *src* e *include* en donde se encuentran los archivos *.cpp* y *.h* respectivamente. Dentro de estos directorios

hay dos archivos a destacar los cuales son el *main.cpp* en src y el *GlobalData.h* en include.

En el archivo *main.cpp* está el script principal del proyecto en donde se encuentran las creaciones de las tasks de FreeRTOS con sus asignaciones a los distintos núcleos de la placa, las inicializaciones de los sensores a utilizar y además se hallan las funciones principales *envioWifi* y *envioLoraOtaa*, las cuales poseen el bucle en el que se censan los datos, se estructuran los mensajes y se envían a la plataforma.

Por otra parte, en *GlobalData.h* es donde se definen todas las constantes globales del proyecto que definirán muchos aspectos de su comportamiento. Es en particular **el único archivo que se debe modificar** antes de grabar un nuevo sensor. Dentro de las variables más importantes de GlobalData.h se encuentra:

- **URL_SERVIDOR:** Contiene el endpoint de la plataforma al que se enviaran los datos, Thingsboard utiliza una ID de dispositivo en la URL para recibir datos de un sensor por lo que cada uno debe tener un valor distinto en esta URL. El ID se puede obtener en el menú de la Figura 1.3 (Inf. Med.)
- **ANTENA_SELECCIONADA:** Contiene la tecnología de transmisión que se utilizara (WiFi, LoRa o alternar entre ambos)
- **RED_WIFI** y **CLAVE_WIFI:** Contienen las credenciales de la red WiFi
- **APP_EUI**, **DEV_EUI** y **APP_KEY:** Contienen las credenciales LoRa
- **DELAY_ANTES_ENVIO_WIFI** y **DELAY_ANTES_ENVIO_LORA:** Contienen el tiempo de espera entre envíos tanto para WiFi como para LoRa
- **USAR_SENSOR_X:** Denota que sensores utilizar, son solamente banderas booleanas para activar o desactivar los sensores definidos, en cada proyecto existen distintas de estas banderas según los sensores conectados.

Por último se hallan otros archivos que se encuentran en los tres proyectos, entre ellos están *WifiController* y *LoraController* en donde se hallan las funciones relacionadas al envío de los datos, en *SensorController* están las funciones relacionadas a la lectura de datos medidos por los sensores específicos así como la creación de los mensajes recopilando todos los datos para ser enviados y finalmente *LedController* y *ParsingController* con otras funciones auxiliares.

Como se nombra anteriormente, estos dispositivos envían un objeto JSON al enviar por WiFi y un string al enviar por LoRa. El string en el caso de LoRa tiene el formato mostrado en el Listing 2.1, por lo que es necesario implementar un procesamiento extra intermedio para esta información de modo que ThingsBoard pueda

almacenarla.

```
idDisp[6bytes]/dato1[8bytes]/dato2[8bytes]/.../datoN[8bytes]/00\0
```

Listing 2.1: Formato de los datos enviados por LoRa

Nuevamente se recomienda la lectura de (Veirana, 2021, [99]) así como de material en línea en el caso de que se desee abrir los proyectos con PlatformIO para modificar el código fuente.

2.2. Estructura del proyecto Node.js del sensor de procesamiento de imágenes

El proyecto para el sensor de conteo de estacionamientos con procesamiento de imágenes está pensado para ejecutar en una Raspberry Pi por lo que tiene una estructura completamente distinta a los anteriores. El proyecto está escrito en JavaScript y posee un único archivo de código fuente llamado *index.js*. Luego contiene dos directorios *yolov3* y *yolov7* en donde está el código fuente de ambos algoritmos de procesamiento de imágenes, en el caso de YOLOV3 ya viene compilado para ejecutar.

El archivo index.js posee una función principal que setea un cronJob para que ejecute la función *ejecucionPrincipal* cada cierto tiempo. En la parte superior del archivo se encuentran las constantes que hay que definir para la correcta ejecución del script, con la constante *algoritmoCatalogacionUsado* se puede seleccionar el algoritmo a utilizar (yolov3 o yolov7), con la constante *endpointTB* se declara el endpoint de ThingsBoard y en el resto se definen otros parámetros relacionados a Yolo. Entre estas constantes también está *regularidadEjecucion* para controlar la periodicidad del cronJob, la cual utiliza un string en formato cron. Dentro de la función *ejecucionPrincipal* desencadenada por el cronJob se ejecuta un comando para tomar una fotografía con la webcam y posteriormente se corre un comando para ejecutar el algoritmo de procesamiento de imágenes sobre esa fotografía. Finalmente el archivo de texto que contiene los resultados de la clasificación que dejan los modelos (cada uno tiene su formato) son procesados con una función específica para obtener el número de vehículos contados y se envía hacia la plataforma para terminar.

Para poder ejecutar este programa en una Raspberry Pi se deben usar los comandos listados en el Listing 2.2 mientras se está situado en el directorio raíz del

proyecto, habiendo instalado antes Python y Node.js.

```
$ sudo apt install –y zip htop screen fswebcam
$ npm i
$ cd yolov7
$ pip install –r requirements.txt
$ cd ..
$ node index.js
```

Listing 2.2: Instalacion y ejecucion del programa en una Raspberry Pi

2.3. Instalación de los algoritmos de procesamiento de imágenes para usarlos individualmente

En esta sección se disponen las guías de instalación por si se desean ejecutar los algoritmos de reconocimiento de objetos utilizados en el sensor de la Sección 3.4 de forma independiente.

2.3.1. Instalación de fswebcam

El paso previo opcional a la instalación de los modelos es el de la instalación del software que nos permitirá tomar las fotografías a procesar desde una webcam conectada a la Raspberry Pi. Para eso instalamos *fswebcam* con el siguiente comando.

```
$ sudo apt install fswebcam
```

Listing 2.3: Instalación de fswebcam

Nota: En caso de querer procesar imágenes ya tomadas, este paso puede omitirse.

2.3.2. Instalación de YOLOV3

Para instalar y ejecutar YOLOV3 de forma independiente se utilizó la guía presente en (Cantos, 2019, [19]). La secuencia de comandos se muestra en el Listing 2.4.

```
$ sudo apt–get install –y git pkg–config build–essential make
$ git clone https://github.com/AlexeyAB/darknet
$ cd darknet
```

```
$ sudo nano Makefile
  => En el archivo cambiar: OPENMP=0 a OPENMP=1
$ sudo nano cfg/yolov3.cfg
  => En el archivo, se pueden cambiar los campos
  => "width" y "height" en conjunto
  => Por ejemplo a: 224, 416 o 608
$ make
$ wget https://pjreddie.com/media/files/yolov3.weights
```

Listing 2.4: Instalación de YOLOV3

Habiendo instalado lo anterior, se ejecuta el algoritmo sobre la imagen *img.jpg* con el comando del Listing 2.5. En YOLOV3 las imagenes finales con las bounding boxes dibujadas se guardan siempre en el archivo *predictions.jpg*.

```
$./darknet detect cfg/yolov3.cfg yolov3.weights ./img.jpg > ./pred.txt
```

Listing 2.5: Ejecución de YOLOV3

2.3.3. Instalación de YOLOV7

Para instalar YOLOV7, se deben ejecutar los comandos del Listing 2.6.

```
$ sudo apt install –y zip htop screen
$ git clone https://github.com/WongKinYiu/yolov7.git
$ cd yolov7
$ sudo nano requirements.txt
  => En el archivo cambiar: protobuf<4.21.3 a protobuf==3.9.2

=> Opcion 1:
=> Instalar dependencias de Python en un virtual env:
$ python –m venv ./test–venv
$ source ./test–venv/bin/activate
$ pip install –r requirements.txt
=> Opcion 2:
=> Instalar dependencias de Python globalmente:
$ pip install –r requirements.txt

$ wget https://github.com/WongKinYiu/yolov7/releases
/download/v0.1/yolov7.pt
```

Listing 2.6: Instalación de YOLOV3

Habiendo instalado lo anterior, se ejecuta el algoritmo sobre la imagen *img.jpg* con el comando del Listing 2.7. En YOLOV7 las imagenes finales con las bounding boxes dibujadas se guardan siempre en el directorio *runs*.

```
$ python detect.py --weights yolov7.pt --source ./img.jpg
--img [640|1280] --save-txt
```

Listing 2.7: Ejecución de YOLOV7

2.4. Pruebas menores extra sobre las ESP32

Durante el estudio de las capacidades de comunicación de los chips ESP32 se realizaron distintas pruebas de conectividad de las que resultaron dos prototipos. Uno de ellos prueba la comunicación Bluethooth Low-Energy y el otro el protocolo ESP-NOW. Dentro de ambos prototipos encontramos dos proyectos PlatformIO para grabar en las dos placas ESP32 las cuales probarán la comunicación entre ellas.

2.4.1. Prototipo BLE

Para utilizar este proyecto se sugiere tener una PC para cada dispositivo ESP32 o de lo contrario de debe utilizar PlatformIO con los distintos puertos configurados correctamente en los archivos platformio.ini de cada proyecto.

Se dispondrá de un dispositivo (proyecto dispositivo-servidor) el cual emitirá mensajes al dispositivo cliente (proyecto dispositivo-cliente) que se le conecta al servidor y los recibe.

Los dispositivos tienen constantes en el archivo *GlobalData.h* en el cual se encuentran los IDs de los servicios (*UUID_SERVICIO* y *UUID_CARESTERSTICA*), los cuales deben coincidir entre cliente y servidor. Estas constantes ya vienen seteadas con valores coincidentes en ambos proyectos.

Una vez se tienen los valores de las variables listos ya es posible realizar el flash de los proyectos en cada dispositivo. Abriendo el monitor serial será posible ver que el servidor le envía al cliente un mensaje diciendo "Hola, te mando un <valor entero de un contador>". El valor del contador cambiará cada diez segundos, y se verá reflejado en el monitor del cliente.

Se utilizaron como referencia los artículos [97] y [72] para la realización de este prototipo.

2.4.1.1. Guía paso a paso para probar el prototipo

Para poner el ejemplo en funcionamiento correctamente, se recomienda el procedimiento descrito a continuación.

1. Conectar el USB del dispositivo servidor, realizar el flasheo del proyecto dispositivo-servidor en el mismo y abrir el monitor serial de PlatformIO.
2. Conectar el USB del dispositivo cliente, realizar el flasheo del proyecto dispositivo-cliente en el mismo y abrir el monitor serial de PlatformIO.
3. Reiniciar el dispositivo servidor con el botón _RST_ que se encuentra en la ESP32 sin desconectarlo. Luego de esto el monitor serial de PlatformIO comenzará de cero y en el monitor serial del cliente se podrá apreciar la conexión y los mensajes recibidos.

2.4.2. Prototipo ESP-NOW

Para utilizar este proyecto se sugiere tener una PC para cada dispositivo ESP32 o de lo contrario de debe utilizar PlatformIO con los distintos puertos configurados correctamente en los archivos platformio.ini de cada proyecto.

Se dispondrá de un dispositivo (proyecto dispositivo-1) el cual emitirá y recibirá mensajes junto con el otro dispositivo (proyecto dispositivo-2).

Se utilizó como referencia el artículo [71] para la realización de este prototipo.

2.4.2.1. Guía paso a paso para probar el prototipo

Se deben cambiar los valores de dos variables concretas para la correcta ejecución del ejemplo.

1. El nombre del dispositivo en la constante *ID_DISPOSITIVO*, ubicada en el archivo *GlobalData.h*.
2. La MAC del dispositivo al que le voy a enviar mensajes en la variable *uint8_t macReceptor[]*, ubicada en el archivo *main.cpp*.

Nota: Para saber la MAC de un dispositivo, esta se puede printear a partir del resultado de la función *WiFi.macAddress().c_str()*.

Una vez se tienen los valores de las variables listos ya es posible realizar el flash de los proyectos en cada dispositivo. Abriendo el monitor serial será posible ver que los dispositivos se envían entre ellos un mensaje diciendo "Soy el dispositivo <ID_DISPOSITIVO>".

ANEXOS

Anexo 1

Menciones especiales del proyecto

Participación del proyecto en Ingeniería deMuestra 2022

Este proyecto de grado participó en el evento de Ingeniería deMuestra 2022[48] presentando un stand con los sensores de calidad de aire funcionando en tiempo real así como con una pantalla mostrando las gráficas formadas por la plataforma ThingsBoard. Se recibió un público variado que iba desde estudiantes de secundaria y UTU, estudiantes de posgrado de la Facultad de Ingeniería, aficionados a la tecnología de todas las edades e incluso representantes de distintas empresas las cuales se desempeñan en ámbitos o tecnologías similares a las abordadas en el proyecto. En todos los casos mencionados es posible afirmar que la aceptación fue muy positiva y se notó un claro interés de todos en la temática del Internet de las Cosas.

Sumado a esto el proyecto tuvo el honor de recibir el segundo lugar en los premios otorgados por el tribunal de Ingeniería deMuestra.

Utilización de la plataforma del proyecto en el Workshop de gemelos digitales para los sistemas ciber-físicos 2022

Desde el 7 al 11 de Noviembre de 2022 se llevó a cabo el Workshop en sistemas ciber-físicos 2022 sobre *Gemelos Digitales para los Sistemas Ciber-Físicos*[44]. Este taller estuvo conformado por distintas charlas abiertas así como un curso para estudiantes de posgrado.

Para dar soporte a la parte práctica del workshop se disponibilizaron usuarios de la plataforma ThingsBoard usada en el prototipo del proyecto para los participantes del curso. Esto demostró una vez más la flexibilidad y utilidad de la plataforma incluso para diferentes proyectos llevados a cabo en la Facultad.

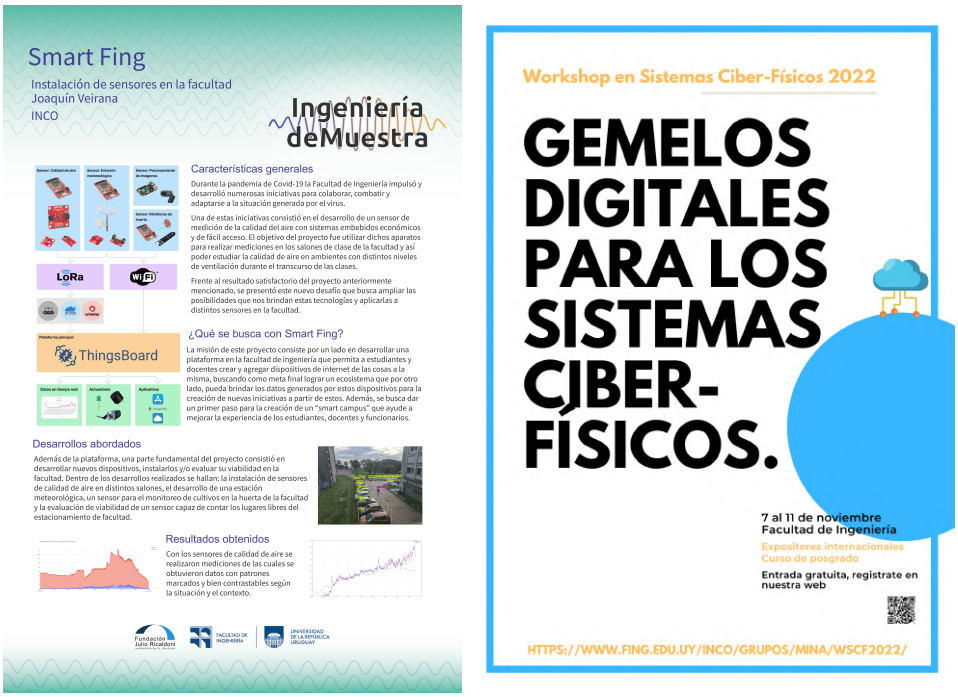

Figura 1.1: ***Izquierda: Poster del stand del proyecto en IdM 2022. Derecha: Poster del Workshop de gemelos digitales para los sistemas ciber- físicos 2022.***