UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Diseño, desarrollo y validación de un sistema de monitorizacion para Ethernet en sistemas embebidos enfocado al vehículo Hyperloop

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Zaballa Martínez, Felipe

Tutor/a: Molero Prieto, Xavier

CURSO ACADÉMICO: 2022/2023

Resum

Aquest treball desenvolupa les principals activitats de programari que s'elaboren a Hyperloop UPV. Aquest projecte universitari està centrat en el desenvolupament de prototips hyperloop completament funcionals, la finalitat dels quals és la verificació i validació de tecnologies aplicables a aquest concepte. Es troba adherit al programa Generació Espontània de la Universitat Politècnica de València i participa tots els anys a la competició universitària internacional European Hyperloop Week.

A la primera part del treball s'exposa la motivació de l'autor per seleccionar el tema, detallant els objectius del treball i les competències adquirides per aquest desenvolupament. A més, es descriuen les metodologies emprades per a l'elaboració de l'informe. Es descriu en profunditat el concepte hyperloop, destacant-ne els principals sistemes i la seua constant evolució. A més, es descriu breument la història d'Hyperloop UPV i la seva participació a l'European Hyperloop Week. Posteriorment, es detalla l'exercici del subsistema Software, aspecte clau d'aquest projecte.

Una de les principals parts es correspon amb l'arquitectura de xarxa de Hyperloop UPV. S'hi inclou una descripció detallada de les subxarxes internes del vehicle, la infraestructura i l'encarregada de la comunicació amb el públic. Per fer-ho, s'expliciten els requeriments necessaris, es documenta la justificació del disseny i finalment s'indiquen els processos de validació d'aquesta arquitectura. A continuació s'aborda el back-end, el programa encarregat de rebre la informació del vehicle i la infraestructura, processar-la i exposar-la a les diferents interfícies gràfiques. En aquesta part es detalla la justificació del disseny, descrivint l'arquitectura i els diferents mòduls implementats, destacant petits fragments de codi que tenen un interès especial per al treball. El front-end es correspon amb el conjunt d'interfícies d'usuari implementades per permetre la interacció amb els membres de l'equip i amb el públic. S'analitza de manera similar al back-end, descrivint-ne els requeriments, justificant l'arquitectura i cadascuna de les interfícies gràfiques desenvolupades. Es para una atenció especial a la interfície de testing "Ethernet-View" i a la interfície de control, "Control-front", que són les principals de l'equip.

Finalment es conclou el treball amb una avaluació global dels resultats obtinguts, es presenten un conjunt de propostes de millora per a cada bloc descrit i es fa una conclusió sobre la consecució dels objectius i les aportacions del document a la formació de l'autor i al projecte Hyperloop UPV.

Paraules clau: Hyperloop, Hyperloop UPV, programari, Generació Espontània, xarxes, back-end, Golang, front-end, React, Typescript

---

Resumen

Este trabajo desarrolla las principales actividades de software que se realizan en Hyperloop UPV. Este proyecto universitario está centrado en la elaboración de prototipos hyperloop completamente funcionales, cuya finalidad es la verificación y validación de tecnologías aplicables a este concepto. Se encuentra adherido al programa Generación Espontánea de la Universidad Politécnica de Valencia y participa todos los años en la competición universitaria internacional European Hyperloop Week.

En la primera parte del trabajo se expone la motivación del autor para seleccionar el tema, detallando los objetivos del trabajo y las competencias adquiridas por este desarrollo. Además, se describen las metodologías empleadas para la elaboración del informe. Se detalla el concepto hyperloop, destacando sus principales sistemas y su constante evolución y, se explica brevemente la historia de Hyperloop UPV y su participación en la

European Hyperloop Week. Posteriormente, se especifica el desempeño del subsistema Software, aspecto clave de este proyecto.

Una de las partes principales se corresponde con la arquitectura de red de Hyperloop UPV. En ella se incluye una descripción detallada de las subredes internas del vehículo, la infraestructura y la encargada de la comunicación con el público. Para ello se explicitan los requerimientos necesarios, se documenta la justificación del diseño y finalmente se indican los procesos de validación de esta arquitectura. A continuación se aborda el back-end, el programa encargado de recibir la información del vehículo y la infraestructura, procesarla y exponerla en las diferentes interfaces gráficas. En esta parte se detalla la justificación de su diseño, describiendo la arquitectura y los diferentes módulos implementados, destacando pequeños fragmentos de código que tienen especial interés para el trabajo. El front-end se corresponde con el conjunto de interfaces de usuario implementadas para permitir la interacción con los miembros del equipo y con el público. Se analiza de manera similar al back-end, describiendo los requerimientos, justificando la arquitectura y cada una de las interfaces gráficas desarrolladas. Se presta especial atención a la interfaz de testing ''Ethernet-View'' y a la interfaz de control, ''Control-front'', que son las principales del equipo.

Finalmente se concluye el trabajo con una evaluación global de los resultados obtenidos, se presentan un conjunto de propuestas de mejora para cada bloque descrito y se realiza una conclusión sobre la consecución de los objetivos y las aportaciones del documento a la formación del autor y al proyecto Hyperloop UPV.

Palabras clave: Hyperloop, Hyperloop UPV, software, Generación Espontánea, redes, back-end, Golang, front-end, React, Typescript

Abstract

This Bachelor's Thesis delves into the primary software activities carried out at Hyperloop UPV. This university project is focused on developing fully functional hyperloop prototypes, with the aim of verifying and validating technologies applicable to this concept. It is affiliated with the Generación Espontánea program at the Polytechnic University of Valencia and participates annually in the international university competition, European Hyperloop Week.

In the first part of this work, the author presents their motivation for selecting the topic, outlining the objectives of the project and the competencies acquired through this endeavor. Additionally, the methodologies employed for crafting this report are described. The hyperloop concept is examined in depth, highlighting its key systems and ongoing evolution. A brief history of Hyperloop UPV and its involvement in the European Hyperloop Week is also provided. Subsequently, the performance of the Software subsystem, a crucial aspect of this project, is detailed.

One of the central sections pertains to the network architecture of Hyperloop UPV. It includes a detailed description of the internal subnetworks of the vehicle, infrastructure, and communication with the public. This section specifies the necessary requirements, documents the rationale behind the design, and indicates the validation processes for this architecture. Then, the back-end, responsible for receiving information from the vehicle and infrastructure, processing it, and presenting it through various graphical interfaces, is addressed. This part elaborates on the justification for its design, describing the architecture and the different implemented modules, highlighting code snippets of particular relevance to the work. The front-end encompasses the set of user interfaces implemented to facilitate interaction with team members and the public. It is analyzed similarly to the back-end, describing requirements, justifying the architecture, and detailing each of

the developed graphical interfaces. Special attention is given to the “Ethernet-View" testing interface and the "Control-front" control interface, which are key components of the team’s work.

Finally, the work concludes with an overall evaluation of the results obtained. A set of improvement proposals for each described block is presented, and conclusions are drawn regarding the achievement of objectives and the contributions of the document to the author’s education and the Hyperloop UPV project.

Key words: Hyperloop, Hyperloop UPV, software, Generación Espontánea, networks, back-end, Golang, front-end, React, Typescript

Índice general

Índice de figuras

CAPÍTULO 1
Introducción

Este capítulo describe la motivación del autor por la selección del tema del documento, así como los objetivos establecidos para su desarrollo y la estructura general del informe. Además, se detallan las metodologías implementadas para la elaboración del Trabajo de Fin de Grado. Se destacan las principales asignaturas de la titulación con relevancia en el trabajo, al igual que las competencias transversales que se han visto reforzadas durante el proceso. Por último, se describe la utilización de la bibliografía y se menciona la relación del trabajo con los Objetivos de Desarrollo Sostenible.

1.1 Motivación

La selección de este tema para elaborar el Trabajo de Fin de Grado viene motivada por la creciente popularidad del concepto Hyperloop, que promete generar una transformación en el ámbito de los medios de transporte. Esta tecnología presenta avances significativos tanto en términos tecnológicos como medioambientales, superando las capacidades de los sistemas de transporte convencionales. Además, destaca por su elevada eficiencia y una velocidad no alcanzada por medios de transporte comerciales en la categoría terrestre.

El interés que me genera esta área de la tecnología provocó que me involucrase en el proyecto universitario Hyperloop UPV del programa Generación Espontánea en octubre de 2019, cuando me uní como el primer miembro del recién formado subsistema Economics. Mi función era gestionar los aspectos económicos del proyecto. En la edición siguiente, pasé a formar parte del equipo de Management, manteniendo este rol de responsabilidad durante el tercer año en el área económica.

En mi última participación en el equipo, durante el curso 2022-2023, decidí cambiar de área con el objetivo de enriquecer mi doble titulación también en el ámbito de la ingeniería informática, incorporándome al subsistema Software. Las labores de este subsistema son de gran interés para el desarrollo de un Trabajo de Fin de Grado sobre esta titulación, por lo que este documento trata sobre las labores realizadas durante esta etapa.

Además, existe una escasa orientación sobre cómo gestionar la navegación, el control y la comunicación de un vehículo autónomo hyperloop mediante software. Supone un desafío importante en el ámbito tecnológico, pero también en el ámbito de la seguridad. Esto representa una oportunidad para desarrollar un diseño íntegramente creado por el equipo para elaborar un programa que manipule un vehículo hyperloop.

Por las razones expuestas anteriormente, este Trabajo de Fin de Grado se enfoca en tres aspectos fundamentales:

- La arquitectura de red que comunica las diferentes placas del vehículo entre sí, con la infraestructura, con los responsables de su control, y con el público que esté presente durante las demostraciones.
- El back-end encargado de recibir toda la información proveniente del vehículo, procesarla, permitir su evaluación y el envío de órdenes para manipular el vehículo.
- Por último, el desarrollo de diversas interfaces de usuario diseñadas para la manipulación del vehículo durante las pruebas de validación y las demostraciones.

Cabe destacar que todos estos sistemas deben estar desarrollados siguiendo los requerimientos establecidos por la normativa de la European Hyperloop Week, la competición en la que participa el equipo todos los años contra universidades de todo el mundo.

1.2 Objeto del proyecto

Este Trabajo de Fin de Grado tiene como objetivo global **diseñar, desarrollar y validar un sistema de monitorización de los diferentes circuitos impresos del vehículo y de la infraestructura para procesar la información ofrecida por los diferentes sensores, representarla a través de una interfaz de usuario y permitir la manipulación del vehículo a través del envío de órdenes desde una estación de control.**

Para alcanzar este objetivo global es necesario establecer una serie de objetivos en cada una de las áreas de interés del documento:

- Desarrollar e implementar un diseño de la arquitectura de red que debe establecerse para garantizar la comunicación entre las diferentes placas del vehículo entre sí, con la infraestructura, y con la estación de control.
- Garantizar el funcionamiento de la comunicación del vehículo con la estación de control en tiempo real y sin pérdidas de paquetes.
- Ofrecer un punto de acceso inalámbrico para que se conecte el público durante las demostraciones y pueda tener acceso a los datos que facilita el equipo, permitiendo observar lo que ocurre en el interior de la infraestructura a través de cámaras.
- Diseñar y desarrollar un back-end funcional que haga de intermediario entre el vehículo y la infraestructura, y la estación de control. Debe permitir el procesado de toda la información originada y facilitar la comunicación de ambas partes de forma precisa y sin pérdidas.
- Validar los sistemas proporcionados por el back-end.
- Diseñar y desarrollar una interfaz de usuario que facilite la etapa de testing de las diferentes placas.
- Diseñar y desarrollar una aplicación que permita validar el funcionamiento del firmware y el hardware con la utilización de un sistema diseñado por el equipo llamado “Software and Hardware Unit Test Platform”.
- Diseñar y desarrollar una interfaz de usuario que permita el envío de órdenes al vehículo y la representación de la información importante durante las demostraciones, tanto para los miembros del equipo como para el público en general.
- Diseñar y desarrollar una interfaz adaptada a dispositivos móviles para mostrar las cámaras y la información básica del vehículo durante las demostraciones para el público presente.

1.3 Estructura de la memoria

A continuación se detalla brevemente el contenido de las diferentes partes de la memoria.

1. Introducción

En este capítulo inicial del documento se expone la motivación del proyecto y sus objetivos. Se proporciona una descripción de la metodología empleada en la elaboración del Trabajo de Fin de Grado, se explora la teoría relacionada con la titulación pertinente y se detalla el desarrollo de las competencias transversales involucradas y los Objetivos de Desarrollo Sostenible.

2. Marco teórico y antecedentes

2.1. Hyperloop: Enfocado en la descripción del concepto de hyperloop, abarca los principales sistemas que lo forman. Se aborda la problemática en el ámbito del transporte y se destacan las ventajas de esta idea. Además, se expone el origen del hyperloop, su evolución desde su concepción inicial y su situación actual en términos de desarrollo.

2.2. Hyperloop UPV: Historia y evolución del equipo Hyperloop UPV. Se realiza un breve resumen de cada generación del equipo. Se describe el origen y progresión de la competición European Hyperloop Week. También se aborda el programa universitario Generación Espontánea, que engloba este proyecto. Se trata la creación de la asociación juvenil que forma el equipo y, finalmente, se define la estructura del equipo y su organización general.

2.3. Subsistema Software: Se centra en la definición del subsistema Software, describiendo su estructura organizativa y las funciones que desempeña en el contexto de Hyperloop UPV.

3. Arquitectura de red

Se establecen los criterios necesarios para el desarrollo de la arquitectura, se explica el diseño elegido y cada una de las subredes elaboradas: la red interna del vehículo, la de la infraestructura, y por último la de comunicación con el público. Por último, se exponen los procesos de validación seguidos.

4. Back-end

Se especifican los requisitos del back-end y la justificación del diseño elegido. Se proporciona información general sobre el repositorio y las herramientas utilizadas, y se describe la arquitectura del back-end especificando cada uno de los módulos. Por último, se exponen las diferentes actividades utilizadas para la validación del software.

5. Front-end

Al igual que el capítulo anterior, se definen los requisitos de funcionamiento del front-end, se establece y se justifica el diseño elegido. Se describe la organización del front-end

y se detalla cada una de las diferentes interfaces de usuario, desde aquellas creadas para la etapa de testing y validación, como la general orientada a las demostraciones. Por último, se trata la validación de estos programas.

6. Resultados, propuestas de mejora y conclusiones

Se evalúan los resultados obtenidos en los programas y la infraestructura y se proponen mejoras para optimizar los diferentes sistemas, resaltando la contribución que pueden suponer para el equipo. Además, se evalúan los objetivos del trabajo y se analiza la consecución de los logros para el proyecto. Por último, se realiza una valoración global del trabajo.

1.4 Metodologías implementadas

La metodología aplicada para este Trabajo de Fin de Grado es una versión de una metodología habitual en el desarrollo de software llamada "Ingeniería de requisitos". Esta metodología se centra en identificar y analizar los requisitos de un proyecto de desarrollo de software para elaborar un diseño e implementar su arquitectura y soluciones [5].

La primera fase de esta metodología consiste en la identificación de requisitos, durante la cual se recopilan y documentan los requisitos del sistema. Se incluyen en esta fase las reuniones con los stakeholders, documentación necesaria existente, entrevistas, etc. En este documento, dada su extensión, no se ha realizado una descripción exhaustiva de cómo se han identificado los requisitos, que vienen impuestos por las obligaciones de la competición en la que participa el equipo (European Hyperloop Week) y por las necesidades que se detectan en el equipo para cumplir los objetivos de cada uno de los subsistemas.

Una vez identificados los requisitos, se especifican en un formato claro y se establecen las pautas de mayor importancia para realizar el diseño. Este apartado se encuentra desarrollado tanto para la arquitectura de red como para el back-end y los diferentes front-end elaborados.

A continuación se procede a realizar el diseño de la arquitectura de red y de los programas, describiendo las soluciones técnicas de mayor interés para cumplir los requisitos. Se describe cómo el diseño propuesto aborda los requisitos identificados.

Una vez confirmado el diseño se realiza la implementación de la arquitectura y el software de manera fiel al diseño.

Por último, se analiza cómo ha sido el desarrollo de su implementación y se valoran las posibles mejoras implementables para desarrollar una solución más eficiente.

1.5 Teoría aplicable

El Grado en Ingeniería Informática es fundamental para el desarrollo del Trabajo de Fin de Grado. A continuación, se describen las aportaciones de las asignaturas más influyentes sobre la elaboración de este documento:

Introducción a la informática y la programación, Programación: Este Trabajo de Fin de Grado está orientado al desarrollo de software y a la programación; estas asignaturas han servido para sentar las bases del código que se ha elaborado.

Ingeniería del software: Permite conocer la forma de organización del código en proyectos de gran envergadura, centrándose en el diseño y su posterior elaboración. Ha sido de mucha utilidad para organizar los diferentes repositorios y distribuirse el trabajo.

Computación paralela, Concurrencia y sistemas distribuidos: Los requisitos de estos sistemas requieren realizar numerosas tareas de forma simultánea, por lo que son necesarios conocimientos de computación paralela y concurrencia. Además, se hace uso de memoria compartida, donde se deben gestionar condiciones de carrera, utilizando mutex, locks, etc.

Estructura de Datos y Algoritmos: Esta asignatura sienta las bases de los tipos de estructuras más utilizados, sus ventajas e inconvenientes, que han sido de utilidad para elegir los diferentes tipos de datos que se utilizan tanto en el back-end como en el front-end, y poder realizar inserciones, búsquedas, etc. de manera óptima.

Interfaces persona-computador: El diseño de las interfaces del equipo ha sido elaborado por el subsistema de Software, por lo que ha sido importante esta asignatura para saber cómo realizar cada uno de los diseños con el objetivo de hacerlos simples, eficientes y atractivos. Además, los estilos CSS han sido trabajados en esta asignatura.

Lenguajes, tecnologías y paradigmas de la programación: Para la elaboración de este Trabajo de Fin de Grado he tenido que aprender a desarrollar en Golang, y a trabajar con la librería de JavaScript de React. Para agilizar el aprendizaje de estos lenguajes, han sido de utilidad los conceptos desarrollados en esta asignatura.

Tecnología de sistemas de información en la red: Ha sido la asignatura donde más se ha trabajado con JavaScript y, por lo tanto, ha sido de utilidad para aprender TypeScript y trabajar en React.

Redes de computadores: En esta asignatura se trabaja sobre los niveles de red y los diferentes protocolos. Ha servido para entender los protocolos UDP y TCP y hacer uso de ellos. También ha sido de gran interés para poder desarrollar servidores DNS, DHCP, etc.

Gestión y configuración de la arquitectura de los sistemas de información: Aquí se aprenden los conceptos fundamentales para desarrollar una arquitectura de red a partir del uso de switches y routers, además del cableado, puntos de acceso inalámbricos, etc.

Gestión de proyectos: Ha permitido gestionar un proyecto de manera eficaz, organizando grupos de trabajo, personas, plazos y tareas.

Fundamentos de computadores: De utilidad por el aprendizaje de numerosos conceptos básicos de la informática, y sobre todo, por permitir la comprensión en la decodificación de arrays de bytes binarios según el formato de datos correspondiente y la utilización del Big y Little Endian.

1.6 Competencias transversales

Las competencias transversales tienen el propósito de establecer un perfil de habilidades para los estudiantes dentro de un marco de referencia común que abarque todas las disciplinas académicas, centrándose en las soft skills. Estas competencias se desarrollan de manera mucho más completa en los proyectos universitarios. A continuación se describe cada una de ellas:

CT-01. Comprensión e integración

Este trabajo supone una integración de los conceptos desarrollados en la carrera, sobre todo en aquellos relacionados con la ingeniería de software, a raíz de la elaboración

del back-end y los diferentes front-end. También supone una integración de aquellos relacionados con redes dado que se gestiona la infraestructura de red del equipo.

CT-02. Aplicación y pensamiento práctico

Los miembros del equipo abordan los desafíos de sus respectivos subsistemas. Este documento se centra en la necesidad de desarrollar una infraestructura de red para la comunicación del vehículo con la estación de control y con el público, y además, crear unos programas robustos que permitan gestionar toda la información del vehículo y la infraestructura y enviar órdenes para que actúen como quiere el equipo. Se requiere la aplicación del pensamiento práctico para solucionar los requerimientos descritos en cada una de las secciones y conseguir un proyecto funcional.

CT-03. Análisis y resolución de problemas

Este documento se centra en la resolución de la metodología de comunicación con el vehículo, y la representación de los datos para los miembros y el público, desarrollando los programas que va a utilizar el equipo para realizar las demostraciones.

CT-04. Innovación, creatividad y emprendimiento

La resolución de los objetivos propuestos para el subsistema presenta infinidad de posibilidades y desarrollos. El subsistema se encarga de encontrar qué soluciones son las más ventajosas para el equipo. Por otra parte, el concepto hyperloop en sí obliga a una innovación y creatividad constante en todos los subsistemas, ya que no se cuenta con muchas información previa de cómo se debe desarrollar.

CT-05. Diseño y proyecto

Se diseñan soluciones para el proyecto Hyperloop UPV, integrando conocimientos y habilidades de diversas disciplinas. Para este documento, se debe diseñar la infraestructura, el back-end y cada una de las interfaces de usuario.

CT-06. Trabajo en equipo y liderazgo

El equipo consta de más de 40 estudiantes distribuidos en equipos que trabajan diariamente para desarrollar una idea conjunta. Hay perfiles encargados de liderar cada subsistema y el equipo completo y además, se realizan reuniones semanales para coordinar el trabajo de todas las personas.

CT-07. Responsabilidad ética, medioambiental y profesional

Hyperloop UPV se enfoca en mejorar la eficiencia de los medios de transporte actuales y reducir las emisiones de CO2. Por tanto, la defensa del medio ambiente es una implicación primordial del proyecto.

CT-08. Comunicación efectiva

Los estudiantes realizan presentaciones en público tanto en inglés como en español semanalmente para mejorar sus habilidades comunicativas. Además, para permitir la colaboración en el desarrollo de código es muy importante una comunicación constante con el subsistema y todos los grupos que se vean influidos por él. De esta manera, se asegura que los diferentes desarrollos sean compatibles entre sí.

CT-09. Pensamiento crítico

El objetivo del proyecto es obtener el sistema más completo posible. Para ello el subsistema Software debe estudiar con detenimiento cada uno de sus diseños y optimizarlos al máximo para suplir las necesidades en unos plazos de tiempo tan limitados.

CT-10. Conocimiento de problemas contemporáneos

Se comprenden las ventajas y limitaciones de los medios de transporte actuales, así como las diferencias que introduce hyperloop en la sociedad. En este caso se atiende so-

bre todo a la comunicación del vehículo con el exterior, la cantidad de transmisión de información de las infraestructuras de red de los medios de transporte, y la implementación de software en vehículos.

CT-11. Aprendizaje permanente

Las limitaciones temporales y económicas hacen que el subsistema Software tenga que reinventarse continuamente, logrando diseños muy eficientes que permitan alcanzar los objetivos del equipo y mejorando año tras año la comunicación con el vehículo gracias a la aplicación de nuevas tecnologías.

CT-12. Planificación y gestión del tiempo

En el proyecto los plazos son ajustados y los miembros del equipo deben aprender a planificar para cumplir con estas fechas y aplicar medidas correctivas si surgen dificultades que desvíen al equipo de su cumplimiento.

CT-13. Instrumental específico

Hyperloop representa un desafío tecnológico que requiere el uso de técnicas y herramientas actualizadas en ingeniería. En el caso del subsistema, el grupo adopta una filosofía similar para identificar mejoras potenciales que optimicen la eficiencia del software.

1.7 Manejo de la bibliografía

El manejo de la bibliografía está enfocado principalmente al apoyo teórico de los diferentes conceptos desarrollados en cada capítulo del documento.

Durante la sección de introducción, se utiliza la bibliografía para definir las metodologiás implementadas en el documento. Esta referencia permite acceder a información adicional sobre esta metodología de ingeniería de software, en caso de interés.

En la sección que trata sobre el marco teórico del trabajo, la bibliografía es utilizada principalmente para la descripción del concepto, sus sistemas y la evolución que ha tenido a lo largo de los años. En cambio, en la parte centrada en el equipo Hyperloop UPV y en su organización, no se hace referencia apenas a la bibliografía, simplemente se añade algún pie de página con alguna referencia del equipo que no tiene una importancia notable.

En los capítulos troncales del documento, la bibliografía permite explicar de manera clara la teoría necesaria para la comprensión del capítulo. Además, se hace referencia a la principal documentación, o los principales libros sobre las herramientas y lenguajes utilizados, permitiendo la profundización de cada uno de ellos en caso de interés.

Por otra parte, aquella información genérica obtenida de las páginas web de las instituciones, empresas, etc. son indicadas al pie de página a modo de referencia, debido a que no tienen un mayor valor en el desarrollo del Trabajo de Fin de Grado.

1.8 Objetivos de Desarrollo Sostenible

El concepto hyperloop propone el desarrollo de un medio de transporte sostenible que pueda reemplazar a los medios tradicionales, cuyas emisiones son un problema. Además, Hyperloop UPV se compromete activamente a abordar los desafíos globales establecidos por los Objetivos de Desarrollo Sostenible, esforzándose por minimizar cualquier posible impacto ambiental relacionado con la generación de residuos, el transporte

de materiales, y otros aspectos del proyecto. Por lo tanto, no solo es partícipe con la realización de un proyecto sostenible, sino que también por su forma de trabajo. Esta sección se ve ampliada en el Anexo I.

CAPÍTULO 2
Marco teórico y antecedentes

En esta sección, se presenta el concepto hyperloop y se explora su evolución a lo largo de la historia. Además, se introduce el proyecto universitario Hyperloop UPV, destacando sus logros en diferentes ediciones y su participación en la competición European Hyperloop Week. Se describe el programa Generación Espontánea que respalda este proyecto, así como la entidad legal establecida para su gestión. Por último, se ofrece una visión general de la estructura del equipo, se enfoca en el subsistema Avionics y se detalla el funcionamiento del subsistema Software, incluyendo su metodología de trabajo.

2.1 Hyperloop

2.1.1. Concepto

Hyperloop se refiere a un medio de transporte terrestre de muy alta velocidad orientado para desplazar tanto pasajeros como cargas. Se presenta como una alternativa para transporte interurbano, conectando ciudades y ofreciendo ventajas sobre los productos actuales de transporte tradicional, especialmente en recorridos que rondan entre los 500 y 1500 kilómetros.

Consiste en unas cápsulas propulsadas eléctricamente que levitan en el interior de un tubo a baja presión. De esta manera, al reducir en gran medida la resistencia y las pérdidas por fuerzas de rozamiento y fricción, pueden alcanzar velocidades superiores a los 1000 kilómetros por hora, aportando una eficiencia muy elevada. Además, no existe ningún tipo de emisión fósil directa derivada de su uso, por lo que ofrece una alternativa sostenible [16].

Para elaborar con éxito el hyperloop y lograr su incorporación de forma segura en los transportes actuales, es necesario el desarrollo de diversas tecnologías avanzadas y su integración en un mismo vehículo. Esto ha llamado la atención tanto de entidades públicas (EU Agency for Railways, Deutsches Zentrum für Luft- und Raumfahrt, etc.), como privadas (Zeleros, Hyperloop Transportation Technologies, Virgin Hyperloop, etc.). Pese a que su desarrollo lleva varios años activo y existe un gran número de empresas y proyectos distintos que trabajan en ello, todavía no se han cerrado estándares para el transporte y hay gran variedad de tecnologías. Actualmente, las tecnologías más instauradas que definen el concepto son las siguientes:

- Infraestructura: Se caracteriza por un tubo en cuyo interior se van a desplazar las cápsulas hyperloop; el principal sistema que tiene esta infraestructura es el de vacío, para proporcionar presiones muy inferiores a la del exterior, eliminando ineficiencias producidas por el rozamiento con el aire.

- Vehículo:
  - Sistema de guiado por levitación: Mediante campos electromagnéticos se evita contacto con ninguna superficie de la infraestructura, aumentando la eficiencia, reduciendo el desgaste y el impacto de vibraciones, mejorando la experiencia del pasajero.
  - Propulsión electromagnética: La tecnología predominante es la de motores de inducción lineal, que permiten, mediante la interacción de campos magnéticos y eléctricos generar fuerzas que impulsan y frenan a la cápsula. Proporciona una energía limpia y eficiente. Además, este tipo de sistemas de propulsión se encuentra en el vehículo, o como mucho, a inicios y finales de tramo de infraestructura, por lo que se abarata el coste de infraestructura, permitiendo mayor escalabilidad.

No son las únicas características básicas del medio de transporte, también son importantes los sistemas de suministro eléctrico, de comunicaciones, protecciones del interior del prototipo, paradas e intercambiadores durante el trayecto, etc., pero sí son las más exclusivas y determinantes.

El desarrollo de este conjunto de tecnologías se traduce en ventajas en eficiencia, sostenibilidad y velocidad frente a los transportes actuales en recorridos de media o larga distancia, es decir, en recorridos entre aproximadamente 500 y 1500 kilómetros [11].

Los sistemas existentes para transporte de larga distancia a alta velocidad son los ferrocarriles de alta velocidad y las aerolíneas comerciales. Por una parte, los aviones tienen unas velocidades que rondan los 900 km/h, una velocidad ligeramente inferior con la que se estima que viajará un hyperloop; sin embargo, las aerolíneas tienen grandes pérdidas de tiempo por el movimiento en el aeropuerto, despegue y aterrizaje, estancia en el aeropuerto, etc. En cambio, se estima que los tiempos extra de un viaje en hyperloop serán mucho más similares a los de un ferrocarril. Por otra parte, los trenes de alta velocidad han demostrado velocidades máximas de 575 km/h aunque los de funcionamiento comercial tienen velocidades inferiores.

Otro competidor es la tecnología Maglev, ligeramente más cercana al concepto de hyperloop, pero limitadas por el rozamiento del aire a velocidades de 600 km/h [13], además de por su elevado coste.

Otra ventaja que ofrece hyperloop es la utilización de energía limpia y la sustitución de combustibles fósiles utilizados por aviones, autobuses, coches, etc. Hyperloop no tiene emisiones de carbono directas [17], por lo que supone una clara ventaja frente a la mayoría de medios de transporte en funcionamiento. Sin embargo, la producción de la electricidad necesaria para que sea utilizada por el vehículo sí lleva asociadas emisiones de dióxido de carbono. Para analizar su viabilidad se debe tener en cuenta la intensidad de esta emisión en la generación de electricidad, que varía entre un país y otro, dependiendo de la proporción de los diferentes combustibles de generación de energía [12]. En algunos países como India, la estructura de generación de electricidad tiene unas emisiones tan elevadas que no producen una ventaja frente a vehículos con emisiones fósiles; sin embargo en muchos países, esto no es así. A medida que se vayan implantando más sistemas de generación de energía renovables, la viabilidad del hyperloop será aún mayor.

En cuanto a la eficiencia de este sistema de transporte respecto a sus competidores, el consumo de energía por asiento es muy inferior al de un avión, aunque es ligeramente superior al de un tren convencional (véase la Figura 2.1). Cabe destacar que la diferencia de velocidad entre los trenes y el hyperloop no es comparable, y el consumo es inferior al doble [14].

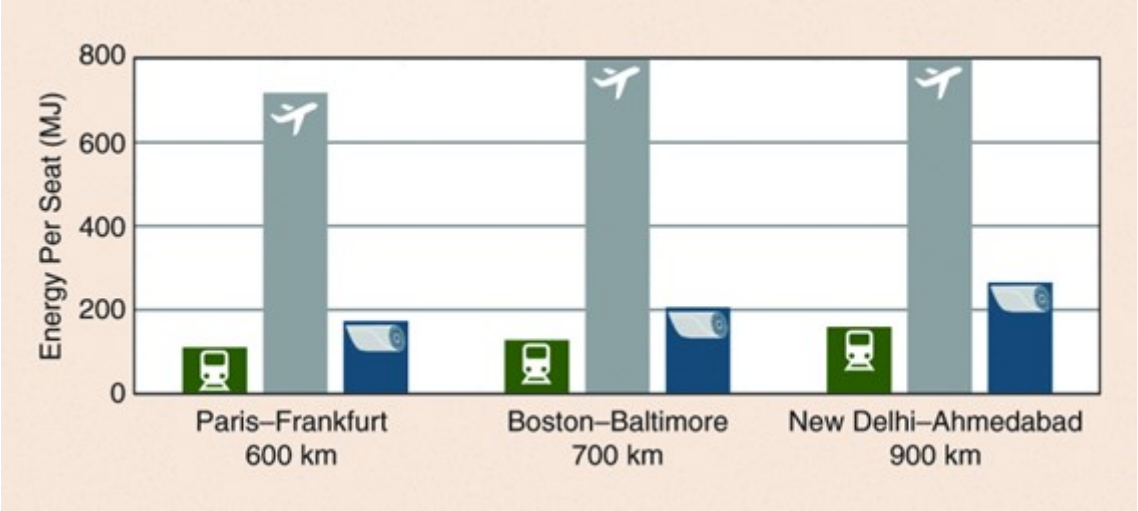

Figura 2.1: Consumo de energía por asiento en diferentes medios de transporte
Fuente: Evacuated-Tube, High-Speed, Autonomous Maglev (Hyperloop) Transport System for Long-Distance Travel: An overview

A pesar de su eficiencia, para recorridos de menor distancia el uso del hyperloop pierde interés. En estos viajes el tiempo en el que el vehículo se encuentra en velocidad de crucero (velocidad máxima mantenida constante durante el trayecto), que es el que destaca en eficiencia, es reducido, por lo que no proporciona un ahorro significativo. Además, los tiempos no son tan notorios frente a los del ferrocarril convencional de alta velocidad que ya se sitúa cercano a la hora.

En el límite contrario, superando los 1500 kilómetros, es más óptimo en términos de tiempo el avión supersónico. Pese a esta desventaja, se deben tener en cuenta las emisiones de este transporte frente a la sostenibilidad que ofrece hyperloop.

2.1.2. Origen

A pesar de parecer un concepto moderno y futurista, la idea de hyperloop surge hace más de 200 años. Las primeras menciones y descripciones datan del siglo XIX por el británico George Medhurst [7]. Desde entonces se han desarrollado múltiples diseños diferentes de trenes, y más adelante, de trenes de alta velocidad, tanto con tracción eléctrica como magnética. Durante este siglo surgieron diversos prototipos que se distanciaban de los trenes del momento. Sin embargo, la dificultad tecnológica que suponía el concepto de hyperloop para la época provocó que quedase estancado.

En 1909, Goddard publicó un artículo llamado "The Limit of Rapid Transit", donde se describía un viaje entre Boston y Nueva York en 12 minutos, y se mencionaron ideas básicas del hyperloop, como la levitación y el vacío [15].

Durante la segunda mitad del siglo XX surgieron nuevas investigaciones, como el Aerotrain de Jean Bertin, un vehículo similar a un tren con levitación pero con cojines de aire para la propulsión en vez de electromagnetismo. En los años 90, investigadores del Instituto Tecnológico de Massachusetts comenzaron un proyecto de desarrollo de un tren en tubos de vacío.

Durante el inicio del siglo XXI, el número de proyectos centrados en este desarrollo creció y, entre estos, se encuentra el consorcio estadounidense ET3 Global Alliance.

2.1.3. Estado actual

Durante estos últimos años, el alto desarrollo de los ferrocarriles ha sido frenado por la fricción con el aire. Esta problemática ha vuelto a traer al foco de atención a hyperloop, que sí propone soluciones para reducir este inconveniente.

Basándose en las ideas comentadas previamente, Elon Musk publica en agosto de 2013 "Hyperloop Alpha", en el que se traen al presente las ideas propuestas por Medhurst y muchos otros autores que trabajaron después que él. Propone un diseño que decide no patentar, utilizando el modelo de proyecto de código abierto.

A partir de aquí, surgen numerosas empresas, como Zeleros, Hyperloop Transportation Technologies, Virgin Hyperloop, etc. que comienzan a trabajar en el desarrollo de hyperloop. Se empiezan a construir zonas de prueba a escala real y a desarrollar con éxito los primeros ensayos. En 2020, Virgin Hyperloop llevó a cabo con éxito su primera prueba con pasajeros.

Además, en 2015, enfocado para el ámbito universitario, se crea la Hyperloop Pod Competition, una competición anual esponsorizada por SpaceX, en la que equipos universitarios se presentan con diseños de prototipos para probar la viabilidad de diversos aspectos de hyperloop. A raíz de esta competición surge el equipo universitario de la Universidad Politécnica de Valencia, Hyperloop UPV.

En 2021, cuatro equipos universitarios crean la European Hyperloop Week, una semana de divulgación, demostración y competición a nivel mundial que toma su primera sede en Valencia.

2.2 Hyperloop UPV

Hyperloop UPV es un equipo universitario de la Universitat Politècnica de València adherido al programa Generación Espontánea que lleva trabajando en esta tecnología desde 2015. El equipo consta de alrededor de 40 estudiantes y se encarga de construir prototipos completamente funcionales para la verificación y validación de tecnología aplicable a hyperloop, demostrando su viabilidad y promoviendo su desarrollo.

El equipo es completamente multidisciplinar, esta formado por estudiantes de la gran mayoría de centros de la universidad, desde diversas ingenierías (Mecánica, Informática, Telecomunicaciones , Aeroespacial, etc.) hasta otras de ámbito empresarial (Administración y Dirección de Empresas) o artístico (Bellas Artes). La estructura del equipo se basa en el coworking, y está organizada en distintos grupos de trabajo llamados subsistemas, cada uno enfocado a un área determinada del desarrollo. Además, el equipo cuenta con dos profesores asesores: Vicente Dolz, Doctor en Ingeniería Mecánica, y Tomás Baviera, doctor en Periodismo.

Todo este entorno favorece a los miembros en el desarrollo de competencias en su carrera, pero también las de carácter transversal.

2.2.1. Historia

H1: Primera generación

En 2015, cinco estudiantes de la Universitat Politècnica de Valencia decidieron participar en la SpaceX Design Weekend celebrada en Texas en enero de 2016. Presentaron una propuesta conceptual única en comparación con otras universidades y el equipo fua galardonado con los premios a Mejor Diseño de Concepto y Mejor Diseño de Propulsión

(véase la Figura 2.2). Este grupo es considerado H1, la primera generación de estudiantes del equipo[1].

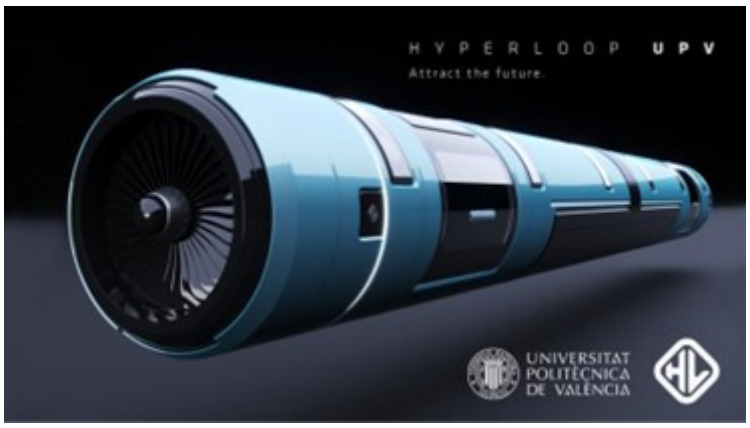

Figura 2.2: Diseño conceptual de hyperloop por Hyperloop UPV
Fuente: LinkedIn Hyperloop UPV

H2: Segunda generación

Después del triunfo en su primera competición, Hyperloop UPV recibió más de 300 solicitudes de estudiantes para participar en la Hyperloop Pod Competition II. Como resultado, H2 aumentó su tamaño hasta alcanzar los 25 miembros con el objetivo de construir un prototipo real (véase la Figura 2.3). En esta competición el equipo trabajó en colaboración con la Universidad de Purdue de Estados Unidos, y juntos lograron construir un prototipo que destacó entre los 10 mejores del mundo.

Figura 2.3: Equipo H2 Hyperloop UPV, curso 2016-2017
Fuente: Hyperloop UPV (2017)

Atlantic II (véase la Figura 2.4) fue el prototipo construido para esta competición, el primer vehículo elaborado por el equipo, en colaboración con la Universidad de Purdue de Estados Unidos. Su característica principal era la del uso de dos esquís magnéticos que permitían que levitase.

[1]Información obtenida de convocatorias económicas anuales de cada generación de Hyperloop UPV

Figura 2.4: Prototipo Atlantic II
Fuente: Hyperloop UPV y Purdue Hyperloop 1 (2017)

H3: Tercera generación

El equipo (véase la Figura 2.5) logró clasificarse nuevamente para la final de la Hyperloop Pod Competition III en 2018, que fue celebrada en Los Ángeles. Durante esta competición, ahora en solitario, volvieron a situarse entre los diez mejores equipos de la competición, la cual reunió a más de 300 universidades de todo el mundo.

Figura 2.5: Equipo H3 Hyperloop UPV, curso 2017-2018
Fuente: Hyperloop UPV(2018)

El prototipo diseñado para H3 fue Valentia (véase la Figura 2.6). Es el primer prototipo diseñado y construido completamente por el equipo de la UPV. El aspecto más característico de su diseño son sus dos motores eléctricos de gran tamaño. Además, se da una importante evolución en el cuidado y desarrollo del diseño exterior.

H4: Cuarta generación

En 2019, se vio necesario incrementar el equipo a 35 miembros tras una renovación de la mayor parte de la plantilla (véase la Figura 2.7). El equipo diseñó y construyó el prototipo Turian, que les llevó a alcanzar el puesto número 8 frente a más de 700 universidades participantes, obteniendo el Innovation Award debido a su diseño estructural e integración de componentes. Otras universidades premiadas junto a la UPV fueron el Instituto Tecnológico de Massachussets (MIT), la Escuela Técnica Federal de Zurich (ETH) y la Universidad Técnica de Munich (TUM).

Figura 2.6: Prototipo Valentia
Fuente: Hyperloop UPV(2018)

Figura 2.7: Equipo H4 Hyperloop UPV, curso 2018-2019
Fuente: Hyperloop UPV(2019)

Turian (véase la Figura 2.8) significó un contraste de gran magnitud frente al anterior prototipo debido a que se cambiaron los dos grandes motores eléctricos de Valentia por doce motores eléctricos independientes. Este prototipo tenía un tamaño y peso mucho menor, con una estructura en fibra de carbono. Con este nuevo diseño se consiguió un vehículo mucho más veloz que los anteriores.

Figura 2.8: Prototipo Turian
Fuente: Hyperloop UPV(2019)

H5: Quinta generación

En H5 (véase la Figura 2.9), durante el curso 2019-2020, el equipo se preparaba para presentarse a la Hyperloop Pod Competition V, con unas expectaciones altas creadas por SpaceX que se planteaba cambios que podrían aplicarse a la competición, como un hipotético tubo curvo de 10 kilómetros. Sin embargo, esta competición no llegó a publicarse. El inicio de la pandemia provocó momentos de incertidumbre muy grande sobre el futuro del equipo, sin una competición de prestigio que supusiese la meta para el equipo. A partir de aquí, surgió la idea de la European Hyperloop Week, pero por motivos evidentes debidos a la pandemia, esta idea debió posponerse para futuras generaciones.

Figura 2.9: Equipo H5 Hyperloop UPV, curso 2019-2020
Fuente: Hyperloop UPV(2020)

El trabajo de este año se centró en optimizar el prototipo Turian, mejorando sus prestaciones y fiabilidad, y completando la etapa de testing que no se pudo realizar satisfactoriamente el año anterior. Además, durante esta etapa se empezó a investigar en las tecnologías que se deseaba implantar, para acercar a la realidad el concepto de hyperloop, por ello se empezaron investigaciones de levitación y propulsión electromagnética, entre otros.

H6: Sexta generación

En el curso 2020-2021, H6 (véase la Figura 2.10) se encontró con una situación complicada, en una época post pandémica, con gran cantidad de restricciones sanitarias, y con un número elevado de limitaciones para la agrupación de personas. La cancelación de la Hyperloop Pod Competition desvaneció el primer objetivo del equipo. Sin embargo, Hyperloop UPV se unió a tres de los equipos universitarios europeos más importantes del momento para organizar la European Hyperloop Week, una competición en la que participan equipos de todo el mundo y tuvo como primera sede Valencia, en julio de 2021. De esta forma, el equipo participó como organizador, y a la vez y de forma independiente, como participante en la competición. H6 logró situarse en el top 3 en esta primera edición.

El prototipo diseñado y construido por el equipo es Ignis (véase la Figura 2.11). La principal novedad que conlleva este vehículo es su sistema de propulsión, un motor de inducción lineal, que permite desplazar el vehículo sin necesidad de hacer ningún con-

Figura 2.10: Equipo H6 Hyperloop UPV, curso 2020-2021
Fuente: Hyperloop UPV (2021)

tacto con una superficie, simplemente a través de campos electromagnéticos. Este diseño da el primer salto real hacia el concepto original de hyperloop.

Figura 2.11: Prototipo Ignis
Fuente: Hyperloop UPV(2021)

H7: Séptima generación

Con la European Hyperloop Week ya instaurada, el curso 2021-2022 de H7 empieza con objetivos muy claros: obtener un buen resultado en la EHW. Con la experiencia obtenida durante las ediciones anteriores, el equipo (véase la Figura 2.12) se plantea la construcción de un prototipo que añada numerosas tecnologías del concepto original de hyperloop, como ya se había empezado a desarrollar durante el curso anterior. Además, en este periodo se da el salto a nivel legal más importante hasta el momento: se decide crear una asociación juvenil sin ánimo de lucro para gestionar de manera más cómoda y rápida el presupuesto obtenido a través de organizaciones externas a la universidad.

Figura 2.12: Equipo H7 Hyperloop UPV, curso 2021-2022
Fuente: Hyperloop UPV(2022)

El prototipo diseñado durante este año es Auran (véase la Figura 2.13). Este vehículo representa un gran avance respecto a las ediciones anteriores gracias a la experiencia obtenida, que permite la realización de mejoras significativas e innovaciones tecnológicas. Auran es el primer vehículo que levita, mantiene propulsión mediante un motor de inducción lineal más sofisticado, y tiene un tubo como infraestructura. Además, se prueba su levitación en un sistema de vacío.

Por otra parte, durante el diseño del prototipo se ha prestado especial atención al diseño del interior, incluido el espacio para los pasajeros. Esto supone una novedad en la competición porque proporciona una sensación realista de vehículo a pequeña escala. Con esta apuesta el equipo fue el más galardonado de toda la competición.

Figura 2.13: Prototipo Auran
Fuente: Hyperloop UPV(2022)

H8: Octava generación

Durante el curso presente, el equipo (véase la Figura 2.14) se presenta a la tercera edición de la European Hyperloop Week, con el objetivo de mantener su posición como equipo más galardonado. Este año el foco del equipo está centrado en la fiabilidad y op-

timización, con el fin de lograr una robustez propia de un medio de transporte comercial. Además, se ha propuesto añadir vacío al proyecto, el último de los principales conceptos que faltaban para desarrollar la idea original de hyperloop. La adición de vacío resuelve o minimiza la mayor dificultad de los medios de transporte terrestres actuales para obtener velocidades superiores, la fricción con el aire.

Figura 2.14: Equipo H8 Hyperloop UPV, curso 2022-2023
Fuente: Hyperloop UPV(2023)

Durante este curso se construye Kenos (Figura 2.15). Este prototipo desarrolla las virtudes de Auran, añadiendo potencia y velocidad. Sin embargo, Auran no dispone de la implementación completa del vacío, uno de los conceptos fundamentales de hyperloop. Por lo tanto, el principal objetivo del año es crear un vehículo que levite, se propulse sin contacto con ninguna superficie, y que además funcione en un ambiente de muy baja presión.

Otro aspecto importante es la separación del vehículo en dos módulos diferentes: una base que alberga todos los sistemas del vehículo y una cápsula superior donde se encuentran los vagones para la carga y los pasajeros. Al independizar ambas partes, el vehículo es más seguro para las personas y se asemeja a un vehículo comercial en funcionamiento, como puede ser un tren.

Además, dada la importancia de la infraestructura para este año, se ha decidido asignar un nombre propio al tubo, Atlas (Figura 2.16). Atlas está capacitada para producir ambientes de muy baja presión en su interior, cuenta con un raíl sobre el que puede trabajar el motor de inducción lineal del vehículo, y supone una estructura óptima para la levitación de Kenos.

2.2.2. European Hyperloop Week

La European Hyperloop Week (EHW) es un evento anual que reúne a equipos de estudiantes, empresas, investigadores e instituciones interesados en el desarrollo de la tecnología hyperloop de todo el mundo[2]. A lo largo del evento, que tiene como duración una semana, se realizan presentaciones y pruebas de prototipos de los equipos que han llegado a la fase final. Además, por otra parte, se realizan conferencias y debates sobre

[2]EHW (marzo de 2021). European Hyperloop Week. Recuperado el 18 de abril de 2023, de https://hyperloopweek.com/event/

Figura 2.15: Prototipo Kenos
Fuente: Hyperloop UPV(2023)

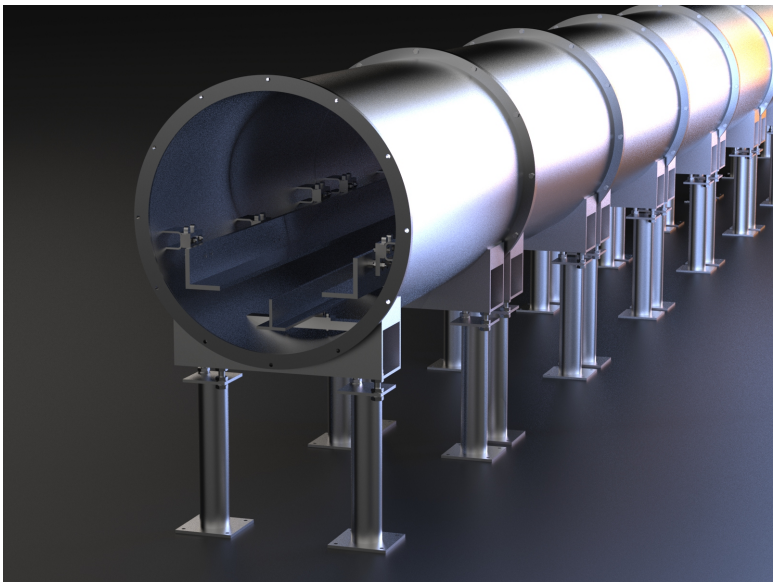

Figura 2.16: Infraestructura Atlas
Fuente: Hyperloop UPV(2023)

esta tecnología. El objetivo principal de la organización es fomentar la colaboración y la innovación en las diferentes áreas de hyperloop, para acelerar su desarrollo y adopción.

El desarrollo de la competición no abarca solo aspectos técnicos, de eficiencia y de velocidad, sino que se exponen aspectos legales, económicos y sociales, a través de debates y charlas, incluyéndose en alguno de los premios que otorga la competición.

Este evento surgió como la principal alternativa a la Hyperloop Pod Competition organizada por SpaceX antes del Covid. Tras este acontecimiento, cuatro equipos universitarios europeos finalistas en las competiciones celebradas en Estados Unidos decidieron agruparse y organizar el primer evento mundial de hyperloop celebrado en Europa. Los cuatro equipos organizadores fueron Hyperloop UPV (Valencia, España), Swissloop (Zurich, Suiza), Delft Hyperloop (Delft, Países Bajos) y HYPED (Edimburgo, Escocia). La primera edición de esta competición tuvo lugar en Valencia en julio de 2021 (véase la Figura 2.17). Al año siguiente el evento se realizó en Delft, Países Bajos, en julio de 2022

(véase la Figura 2.18). Este curso 2022-2023 supone la tercera edición de la competición y se celebra en Edimburgo entre el 17 y el 23 de julio de 2023.

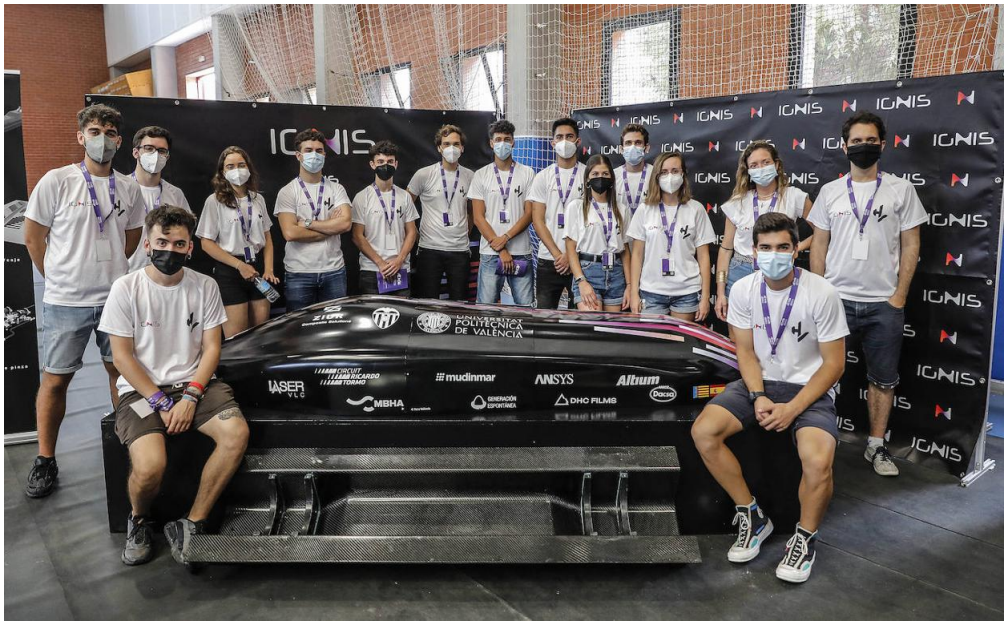

Figura 2.17: EHW 2021
Fuente: Las Provincias (2021)

Figura 2.18: EHW 2022
Fuente: Hyperloop H7

Para poder formar parte del evento durante la semana de la EHW debes clasificarte para la fase final, la clasificación consta de varias etapas en las que se van cribando las solicitudes [4]. Se puede aplicar al evento de tres formas diferentes: como una exposición de tecnología (actividad denominada "Showcase"), como una demostración de tecnología desarrollada por el equipo y su puesta en funcionamiento ("Demonstrate"), o bien, como presentación de la investigación realizada ("Research submission"). Todas estas opciones siguen una estructura similar y tienen las mismas etapas de clasificación para la fase final. A continuación se presentan las etapas de "Demonstrate" a modo de ejemplo:

La primera etapa se realiza alrededor de diciembre y en ella se entrega un documento llamado "Intent to Demonstrate". Este documento se utiliza como una primera solicitud

para el evento y presenta los tipos de demostraciones que el equipo pretende realizar. Este describe el estado actual de cada actividad y las próximas realizaciones sobre el sistema. Se debe explicar cómo se probará el sistema antes del evento.

La segunda etapa presenta el grueso de la aplicación para el evento, se entrega alrededor de marzo y se llama "Final Demonstration Documentation". Con este documento, los solicitantes dan detalles más exactos de cada uno de los sistemas presentados. Es una documentación más técnica y precisa, en ella se debe demostrar que el sistema está diseñado correctamente y se puede utilizar de forma segura. Se deben describir detalladamente los ensayos de los sistemas.

Por último se realiza una tercera etapa en la que se garantiza la seguridad de los vehículos y de cualquier actividad que se vaya a realizar durante el evento. Este documento se llama "Testing and Safety Documentation" y se entrega alrededor de un mes antes de la competición. En este se describen detalladamente los ensayos que se han realizado, mostrando los resultados, mediciones y datos para demostrar su seguridad. Se aconseja enviar documentación gráfica de estas pruebas. A partir de este documento el Comité de la EHW definirá las condiciones de funcionamiento permitidos de cada sistema durante la demostración. Cualquier manipulación errónea de estos métodos de prueba conllevan la expulsión inmediata del evento.

Aquellos equipos que superen todas las etapas tendrán derecho a participar en el evento celebrado en julio de 2023. Los mejores sistemas expuestos o demostrados son premiados. Existen ocho premios diferentes que otorga un jurado imparcial, cada uno de estos enfocado a un área concreta de hyperloop. Además también se reparten diversos premios al mejor diseño entregados por empresas patrocinadoras del evento.

2.2.3. Generación Espontánea

El programa de GE es una lanzadera para iniciativas que provienen enteramente de los estudiantes de la universidad. Se crea en el 2014 [3], dependiente del Vicerrectorado de Alumnado, Cultura y Deporte, y tiene como objetivo fomentar y apoyar actividades extracurriculares llevadas a cabo por alumnos de la UPV. De esta forma se complementa el desarrollo formativo de las titulaciones y se mejora la adquisición de competencias transversales, proporcionando a los proyectos una mayor capacidad económica, administrativa, legal y operativa. Además, con este programa se permite representar a la universidad en competiciones y certámenes.

La filosofía de Generación Espontánea es "aprender haciendo", aproximando los contenidos de la universidad a un entorno aplicado y más cercano al mundo laboral, donde los estudiantes pueden aplicar los conocimientos adquiridos a proyectos de gran variedad. Este programa resulta muy apropiado para dar cabida a estudiantes con ganas de aportar en proyectos innovadores.

Generación Espontánea cuenta con más de 2000 estudiantes y más de 50 grupos, donde se encuentra Hyperloop UPV como uno de los proyectos emblema del Programa.

2.2.4. Asociación juvenil

Durante la generación H7 se decidió establecer una entidad legal para el equipo con el objetivo de gestionar de manera algo más independiente y ágil los patrocinios recibidos por las decenas de empresas que apoyan el proyecto.

[3] Generación Espontánea (s.f.). Recuperado el 20 de abril de 2023, de https://generacionespontanea.upv.es/nosotros/

La entidad elegida fue una asociación juvenil sin ánimo de lucro. Esta organización tiene como objetivo promover y fomentar actividades y proyectos de interés general para la juventud y la sociedad, sin obtener beneficios económicos para sus miembros. El objetivo de esta organización es promover el desarrollo de tecnología hyperloop.

Para facilitar su gestión, el equipo trabaja con la asesoría Naolex Asesores C.B. que se encarga de llevar al día la contabilidad y asesorar al equipo ante temas legales.

2.2.5. Estructura del equipo

La estructura del equipo se basa en el coworking. Para facilitar esta forma de trabajo, el equipo se divide en diferentes grupos separados llamados subsistemas, cada uno de los cuales se centra en un área diferente de desarrollo.

El equipo está formado por 44 miembros estudiantes de la universidad que estudian tanto titulaciones de grado como másteres. Dada la multidisciplinariedad del proyecto, en Hyperloop UPV se ven representados la mayoría de centros de la universidad.

Los subsistemas que conforman el equipo están divididos en dos grandes bloques. Por una parte el área de ingeniería, centrada en el desarrollo tanto del vehículo como de la infraestructura; por la otra, el área de operaciones, encargada de la financiación, administración y marketing:

Área de ingeniería

- **Hardware:** La tarea principal de este subsistema es la de diseñar las placas de circuitos impresos del equipo, realizar su montaje y testear que funcionan correctamente. Se desarrollan placas de potencia, de control, etc. Trabajan con software de diseño y de simulación de circuitos, concretamente utilizan Altium Designer, LTspice y Saturn PCB Design, Inc., entre otros. Los perfiles más habituales en este subsistema son ingenieros electrónicos, de telecomunicaciones o industriales.

- **Firmware:** Se encarga de controlar el funcionamiento y la comunicación de las placas del vehículo. Realizan la programación de sistemas embebidos y trabajan con protocolos de comunicación entre placas. A pesar de utilizar diversos lenguajes, el más habitual es C++. Los perfiles más habituales para firmware son ingenieros electrónicos, de telecomunicaciones, informáticos, industriales y robótica.

- **Software:** Este subsistema se encarga de la comunicación del vehículo con el exterior. Para ello, software diseña, implementa y valida un sistema de monitorización de los sensores del vehículo. Desarrolla un back-end que recibe paquetes de las placas del vehículo con información sobre temperaturas, voltajes, posición, etc., identifica y trata esta información y la muestra a través de diversas interfaces de usuario. Estas interfaces gráficas permiten hacer testing del vehículo, enviar órdenes y visualizar cómo se desarrollan las pruebas. Para el desarrollo del back-end se utiliza el lenguaje Golang, mientras que para el front-end se hace uso de la librería de javascript React, junto con el lenguaje de programación typescript. Los perfiles más habituales para este subsistema son ingenieros informáticos y de telecomunicaciones.

- **Structures & Mechanisms:** Este subsistema se encarga de la parte mecánica del vehículo y de la infraestructura. Mechanics (como se llama el subsistema dentro del equipo) diseña, simula y ensambla el chasis, los sistemas de frenado, los guiados, el carenado, la infraestructura, etc. Para el diseño y simulaciones trabajan con Ansys y

Solidworks, entre otros. Los perfiles más habituales son ingenieros aeroespaciales, industriales, mecánicos, civiles, etc.

- **Electromagnetics:** Se encarga del diseño y validación de los sistemas electromagnéticos del vehículo, y de los sistemas de control. Para ello hace utilización de software especializado como Solidworks, JMAG, Matlab y Simulink, entre otros. Están centrados principalmente en la tracción y la levitación del vehículo. Los perfiles son similares a los expuestos en el subsistema de Structures & Mechanisms.

Área de operaciones

- **Partners & Logistics:** Este subsistema se centra en la financiación del equipo. Con un presupuesto anual que supera los 150k €, de lo que solo alrededor de un 10 % procede de Generación Espontánea. Resulta necesario un subsistema que se encargue de conseguir patrocinadores, gestionar las relaciones y financiar el proyecto. Además, son la cara más visible en eventos públicos. Este subsistema es muy multidisciplinar y no requiere de una formación universitaria específica.

- **Economics:** Se encarga de elaborar los presupuestos, gestionar el dinero del equipo, asegurarse que los procedimientos satisfacen la normativa de la universidad y se cumplen los plazos. Además presenta las convocatorias para recibir dinero de la universidad y gestiona la asociación juvenil. Para este subsistema resulta conveniente cursar estudios relacionados con Administración y Dirección de Empresas.

- **Outreach:** Es el subsistema encargado del marketing del equipo. Además, se encarga de la elaboración de renderizados del vehículo e infraestructura, gestiona redes sociales, realiza el diseño integral del interior de la cápsula, etc. Es habitual que los miembros de este grupo estudien ingeniería de diseño industrial, marketing, diseño gráfico, etc.

Cada subsistema tiene un Project Manager (PM) que gestiona al subsistema. El conjunto de los PM de cada subsistema conforma Management. Y por encima de Management se encuentra Direction, formado por dos directores técnicos, uno de la parte mecánica, y el otro de la parte aviónica, y una directora de operaciones.

Con el objetivo de asesorar a cada subsistema, el equipo cuenta con 13 colaboradores ex miembros que han trabajado en el equipo en generaciones anteriores y cuentan con experiencia en diversas áreas relacionadas con hyperloop. De esta forma guían y ayudan a los miembros ante las dificultades que surjan durante el desarrollo del año.

Además, el equipo cuenta con dos profesores asesores: Vicente Dolz, doctor en Ingeniería Mecánica, y Tomás Baviera, doctor en Periodismo.

2.2.6. Subsistema Avionics

El subsistema Avionics engloba a los tres subsistemas que se encargan de la electrónica e informática del equipo: Hardware, Firmware y Software (descritos anteriormente). Dada la envergadura de su trabajo, este año se decidió separar y gestionar de manera independiente cada una de las disciplinas, aunque necesitan de continua comunicación para desarrollar su trabajo.

Avionics requiere de una jerarquía definida que permita la coordinación entre cada uno de los subsistemas. Este grupo está gestionado por uno de los tres miembros de dirección del equipo, por debajo de este se encuentra un Project Manager general que

dirige los tres subsistemas y se encarga de su coordinación, y además, cada uno de los subsistemas cuenta con un Project Manager que lidera el área.

La forma de trabajo de estos subsistemas es similar y se centra en metodologías ágiles, aunque cada uno lo personaliza a sus necesidades. Se realizan reuniones generales con un intervalo de entre dos semanas y un mes en las que se coordinan e integran las diferentes actividades, aunque hay tareas comunes que requieren de comunicación constante.

El subsistema de interés para este trabajo de fin de grado es Software.

2.3 Subsistema Software

Uno de los principales aspectos en el desarrollo de un vehículo hyperloop es el diseño de un sistema software de control robusto. La principal tarea del subsistema es la de monitorizar y controlar todos los sistemas del vehículo. Este software tiene una importancia vital en el proyecto, ya que un fallo en cualquiera de los sistemas controlados por este puede tener graves consecuencias, y más en el caso de la utilización de elementos que funcionan a altas tensiones, altas presiones, etc. La tarea secundaria del subsistema es compartir toda esta información de forma visual para los integrantes del equipo, el jurado y el público, de forma que todos puedan participar en las demostraciones de la European Hyperloop Week.

Con el objetivo de desarrollar estas actividades se han definido diversas áreas de trabajo diferentes:

- La arquitectura de red, que permite a los diferentes sistemas utilizados durante las demostraciones comunicarse con la Control Station.
- El back-end de la aplicación principal “Control Station“ y de la aplicación de testing “Ethernet-View”, que gestiona todo el procesado de datos y la comunicación con los diferentes sistemas.
- El respectivo front-end de cada una de estas aplicaciones, que son las interfaces de usuario que muestran la información del back-end y permiten interaccionar con el vehículo.
- La herramienta de envío de órdenes y perturbaciones de testing llamada HIL-GUI.

El subsistema Software está formado por tres miembros: Sergio Moreno, Software Lead del subsistema y encargado de su gestión, participa en el desarrollo de todas las áreas de trabajo; Juan Martínez, encargado de la parte del back-end; y el autor de este trabajo de fin de grado, Felipe Zaballa, encargado del front-end. Esta clasificación es formal pero realmente las tareas realizadas se mezclan y son variadas, según las necesidades del equipo y del subsistema.

En las siguientes secciones se desarrolla cada una de estas áreas de trabajo del subsistema.

2.3.1. Metodologías de trabajo

Existen gran variedad de propuestas metodológicas que inciden en diferentes dimensiones en el proceso de desarrollo [2]. Por una parte se encuentran las propuestas tradicionales en las que se establece una rigurosa definición de roles, actividades involucradas y herramientas a utilizar. Estas metodologías son útiles, especialmente en proyectos de

gran tamaño, pero también cuentan con defectos. A medida que se van detectando estos defectos, se pueden añadir nuevos artefactos y herramientas, pero esto generalmente produce un desarrollo más complejo. El tiempo invertido en este tipo de proyectos es elevado y proporcionan poca flexibilidad.

Los principales inconvenientes que presentan las metodologías convencionales son los siguientes:

- Los requisitos impuestos por el cliente deben ser conocidos en su totalidad desde el inicio del proyecto.
- Dificultad de cambiar los requisitos iniciales: a medida que avanza el proyecto las modificaciones y correcciones son más costosas.
- Los mecanismos de control se establecen desde el inicio del proyecto y producen cierta rigidez.
- Excesiva documentación y tiempo dedicado para elaborarla.
- Metodología lenta y que centra sus dificultades al final del proyecto, que produce retrasos de entregas.

Por otra parte existen las metodologías ágiles. Estas dan mayor valor al individuo y al desarrollo incremental del software con iteraciones muy cortas, contando con la opinión y colaboración del cliente a medida que va avanzando el proyecto. Estas metodologías están ganando popularidad, mostrando efectividad en proyectos con requisitos cambiantes y con cierta incertidumbre, permitiendo reducir los tiempos de desarrollo pero manteniendo la calidad. Están orientadas principalmente a proyectos pequeños y suponen una simplificación que mantiene las prácticas esenciales del desarrollo de software. Estas metodologías tienen una fuerte proyección industrial.

El término "ágil" apareció aplicado al software por primera vez en febrero de 2001 en una reunión celebrada en Utah (EEUU). EL objetivo de esta reunión era ofrecer una alternativa a los procesos de desarrollo tradicionales, rígidos y dirigidos por la documentación. Aquí se creó The Agile Alliance, una organización dedicada a promover los conceptos relacionados con las metodologías ágiles. Estos conceptos se recogen en el documento llamado "El Manifiesto Ágil", entre los que destacan:

- Satisfacer al cliente con continuas entregas que aporten valor, con software que vaya añadiendo funcionalidad al proyecto. Se valora al individuo y las interacciones del equipo de desarrollo sobre el proceso y herramientas.
- Desarrollar software funcional priorizado frente a una buena documentación.
- Colaboración con el cliente: no se trata simplemente de una negociación de contrato.
- Responder a los cambios permitiendo alteraciones en el plan.
- Comunicación y diálogo constante entre los miembros del equipo.
- Atención constante a la calidad técnica y al buen diseño, centrándose en la simplicidad.

La metodología SCRUM

La metodología SCRUM forma parte de las metodologías ágiles y define un marco para la gestión de proyectos [20]. Sus características principales se resumen en dos. La primera consiste en que el desarrollo se realiza mediante iteraciones llamadas sprints, que tienen duraciones cortas, de aproximadamente 30 días. El resultado de cada sprint es un MVP (Producto Mínimo Viable), un producto ejecutable que añade cierta funcionalidad. La segunda característica es la constante comunicación y el número de reuniones realizadas, siendo habitual una reunión diaria de 15 minutos centrada en la coordinación del equipo.

Esta metodología es adecuada para los proyectos que se caracterizan por los cambios constantes y la incertidumbre. Es conveniente para grupos centrados en la autoorganización, con un control moderado y centrados en la transmisión de conocimiento. La metodología SCRUM promueve el movimiento de personas entre proyectos y actividades.

Esta metodología de trabajo lleva implementada en el subsistema desde el inicio del curso. Se trabaja por MVPs definidos para tareas de entre dos semanas y un mes. Se realizan reuniones semanales para dar continuidad al trabajo, informar al resto de compañeros de los avances y dificultades, y para estimar previsiones de plazos y resultados. Además, el trabajo se realiza en su mayor parte en el local habilitado para Hyperloop UPV en el Edificio 8N, por lo que hay comunicación constante y colaboración entre los componentes del grupo. Todos los días se realiza un "daily" para ver los objetivos para cada día y analizar cómo se están cumpliendo los plazos.

A nivel conjunto en el subsistema de Avionics, se realizan reuniones con una periodicidad similar a la de los sprints, durante estas reuniones se pone en común el cumplimiento de los objetivos generales, las dificultades encontradas y los cuellos de botella. Además, a nivel global se realiza una breve exposición por subsistemas en la reunión general realizada todos los viernes a las 19:30h. De esta manera, todos los integrantes del grupo son conscientes de la situación en la que se encuentra cada subsistema. Esta exposición se aplica tanto a los subsistemas del área técnica, como los del área de operaciones.

A continuación, se expone una breve descripción de los MVPs y sprints realizados por el subsistema a lo largo de la edición. Dada la extensión de las actividades desarrolladas, se agrupa en secciones mayores con el objetivo de que quede explicado cada una de las etapas de forma genérica en su conjunto. Se ha de tener en cuenta que cada bloque contiene MVPs intermedios funcionales. Las fechas son orientativas y han sufrido alteraciones durante el curso. Por otra parte, siguiendo la definición de las metodologías ágiles, estos periodos no se corresponden con proyectos fijos, estáticos y con un comienzo y final muy definido, estas fechas orientan la elaboración de un producto viable, funcional y aceptable para el desarrollo del año. Sin embargo, a medida que se puede ajustar el tiempo se siguen desarrollando y mejorando algunos MVP que tienen mayor margen de mejora. En todo caso, antes del periodo final de estas fechas ya se cuenta con un mínimo producto viable funcional, a partir del cual se va iterando. En la Figura 2.19 se puede observar la distribución a nivel general de la edición:

Durante la primera etapa del año se estableció un mes y medio de formación y aprendizaje, este periodo contó con 3 sprints diferenciados (véase la Figura 2.20). En primer lugar, se realizó formación sobre el lenguaje de programación golang y se pusieron como objetivos varias actividades de programación genérica, websockets, etc. para facilitar el desarrollo durante el año de estas competencias. El segundo sprint estuvo centrado en React, tratando también las generalidades de la librería y desarrollando diversas actividades. La última etapa estuvo dedicada a aprender sobre las dificultades surgidas a nivel

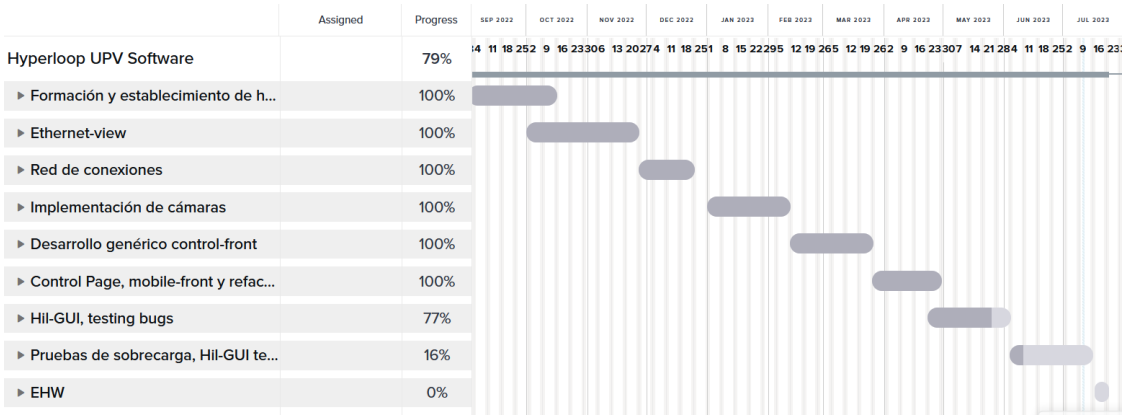

Figura 2.19: Planificación anual Software
Fuente: Elaboración propia

más personalizado, y a investigar de las principales librerías y tecnologías que debían utilizarse para lograr los objetivos establecidos al comienzo de la edición. Además, durante esta etapa se realiza el diseño a través de figma del Ethernet-View, el primer programa que va a desarrollar el subsistema. Al final de este bloque teníamos un sprint de mayor dimensión con las herramientas a utilizar por el equipo y una concreción mayor de las principales actividades a desarrollar durante cada etapa.

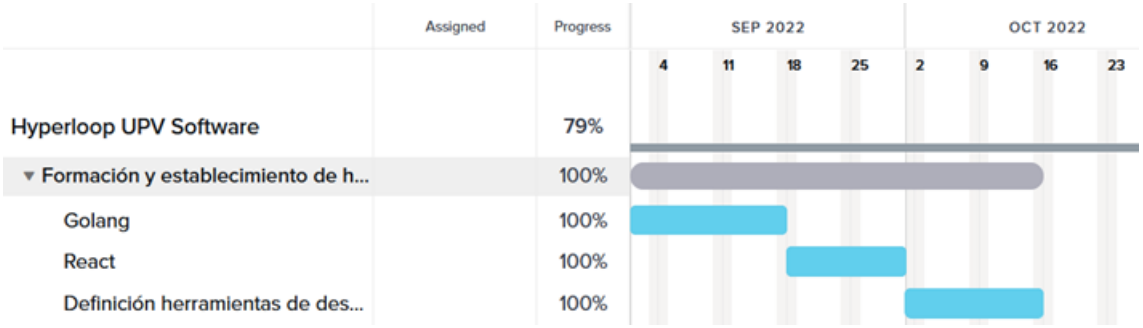

Figura 2.20: Formación y establecimiento de herramientas
Fuente: Elaboración propia

El siguiente bloque comienza a inicios de octubre y desarrolla el programa Ethernet-View, encargado de permitir el testeo de las diferentes placas del vehículo, la emisión de órdenes y el feedback constante del estado del vehículo (véase la Figura 2.21). Este bloque cuenta con diversos sprints y MVPs funcionales en los que se va mejorando el desarrollo del backend y el correspondiente frontend. En cada una de los MVP se añaden funcionalidades adicionales que van desde características básicas del backend, como es el caso del Excel manipulable por el resto de Avionics para personalizar la configuración, hasta detalles del frontend, como por ejemplo el "drag and drop" de gráficas o la animación de carga.

Tras definir el Ethernet-View, se puede dar paso al siguiente periodo, que está centrado en la red de conexión de las diferentes placas del vehículo con los programas desarrollados por software y toda la conexión del vehículo con el exterior (véase la Figura 2.22). De forma paralela también se gestiona la red de conexiones con el público para la competición. A su vez, se inicia el diseño de los programas principales a través de figma.

A mediados de enero se comienza con la implementación de las cámaras. El objetivo de este bloque es permitir visualizar el funcionamiento del vehículo y el recorrido que realiza por dentro de la infraestructura cuando está cerrado a bajas presiones, para me-

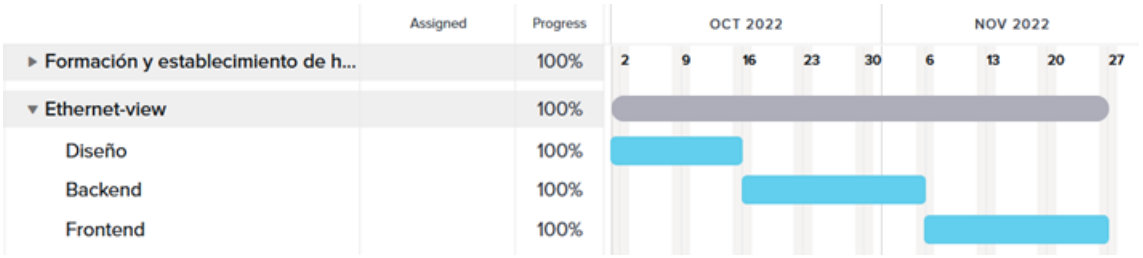

Figura 2.21: Ethernet-view
Fuente: Elaboración propia

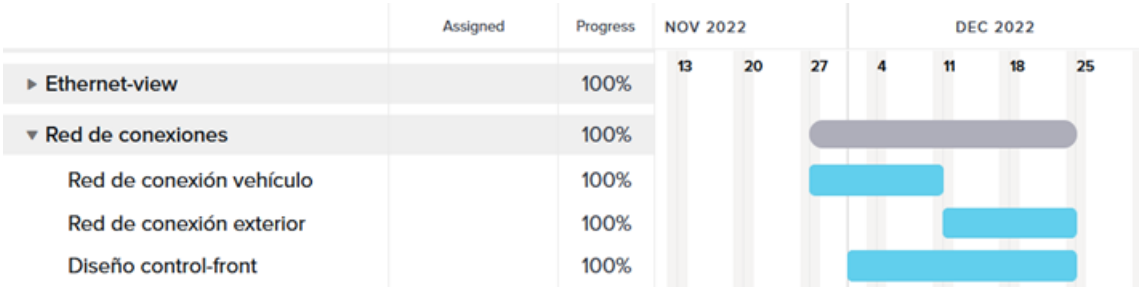

Figura 2.22: Red de conexiones
Fuente: Elaboración propia

jorar la experiencia de usuario del público. En estos sprints se desarrolla la creación del servidor donde operan las cámaras, la simulación de clientes y la red establecida para conectar las cámaras del vehículo y la infraestructura con la página del control-front (véase la Figura 2.23). Además, se realiza el diseño de las interfaces de usuario para las cámaras.

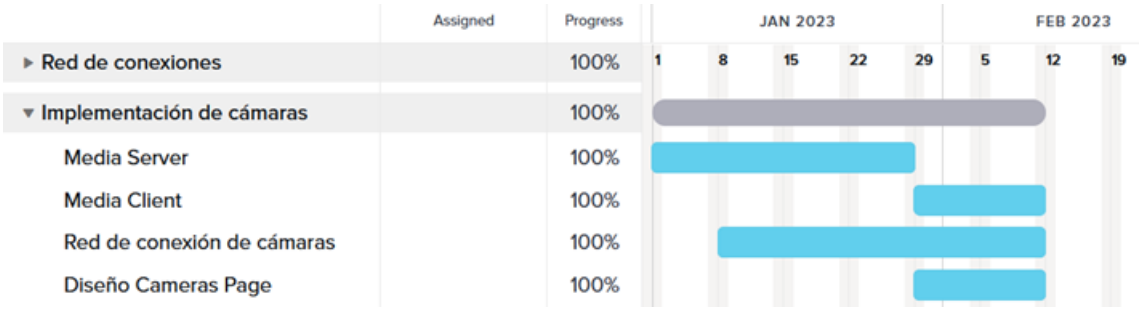

Figura 2.23: Implementación de cámaras
Fuente: Elaboración propia

Tras definir productos viables en cada una de las secciones anteriores, es el momento de comenzar con el programa más extenso del subsistema, la página del Control-Front de la Control Station (véase la Figura 2.24). Durante este periodo se definen sprints para realizar la estructura principal del repositorio y el front-end. Además, se desarrolla la pestaña principal, la página con información del tubo y la página de las cámaras diseñada durante el bloque anterior.

A partir de abril, el grueso del trabajo del subsistema es funcional. La tarea más crítica para este periodo es el diseño y la implementación de la página de control del Control-Front. Además, una vez realizadas las primeras pruebas con este nuevo programa, es el momento de analizar las mejoras necesarias sobre el back-end y hacer una refactorización y simplificación de los diferentes casos de uso. Por otra parte, se realiza el desarrollo de la página web para móviles que será utilizada con el público (véase la Figura 2.25).

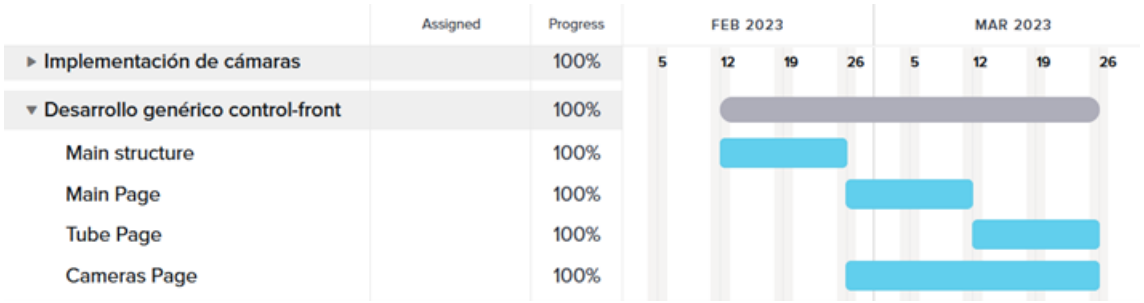

Figura 2.24: Desarrollo genérico control-front
Fuente: Elaboración propia

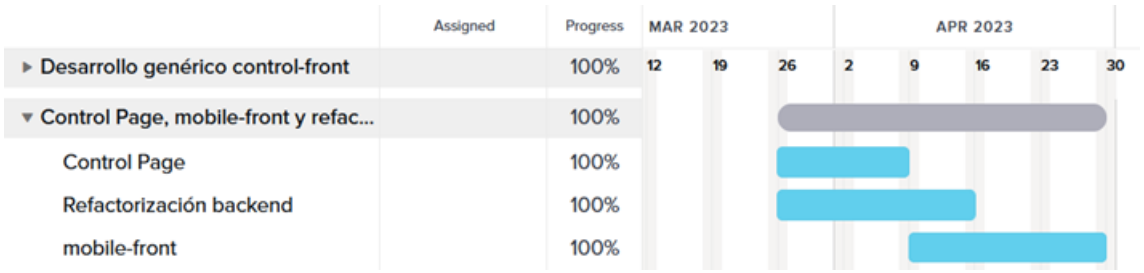

Figura 2.25: Control Page, mobile-front y refactorización
Fuente: Elaboración propia

Es el momento de desarrollar la principal página de testing del equipo, la HIL-GUI. En ella se representa el vehículo a partir de la información del estado que se facilita a través del HIL que interpreta los sensores, y desde esta página se envían perturbaciones y órdenes para ver cómo se comporta. Se realizan tres diferentes sprints: uno para el back-end, otra para el front-end, y un último de testeo de la funcionalidad de la aplicación a través de un módulo que simula el funcionamiento del vehículo (véase la Figura 2.26). Durante todo el periodo se ha de corregir cualquier error que salga en los diferentes programas mientras se hace testing del vehículo.

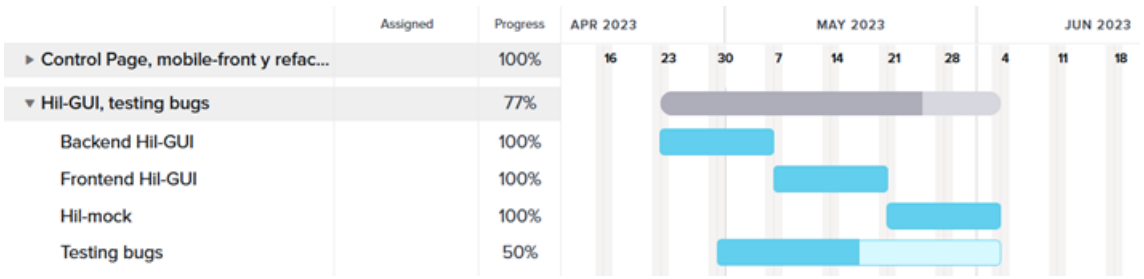

Figura 2.26: HIL-GUI y resolución de bugs
Fuente: Elaboración propia

Durante el último mes previo a la competición se realiza el testeo de todos los sistemas del vehículo. Se debe identificar cualquier inconveniente o bug de los programas y corregirlo. Además, es importante atender las sugerencias que proponen el resto de los subsistemas para su etapa de pruebas y cambiar cualquier implementación que sea necesaria para mejorar la actuación del resto de grupos. Otro sprint principal para este periodo es la prueba de sobrecarga de red para poner a pruebas los servidores con simulaciones superiores al número de clientes esperados durante la competición (véase la Figura 2.27).

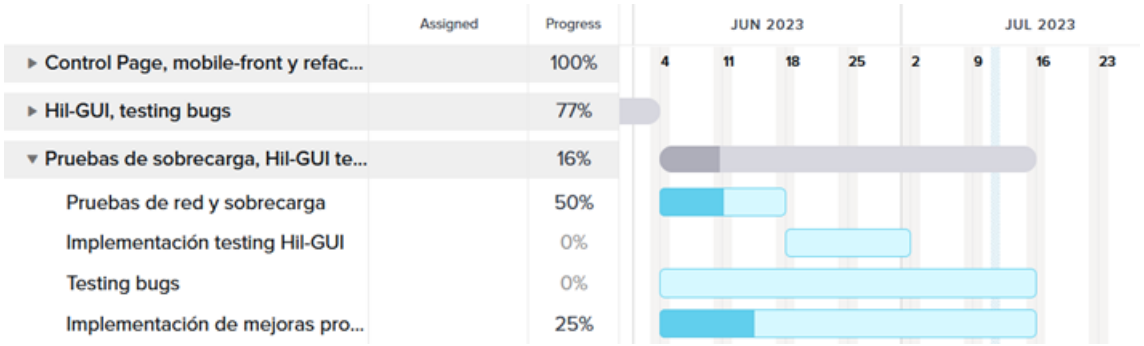

Figura 2.27: Pruebas de sobrecarga y testing
Fuente: Elaboración propia

Una vez superado este periodo, todas las actividades del subsistema han sido cubiertas y estamos preparados para la European Hyperloop Week.

2.3.2. Desarrollo de código

Para el desarrollo de código dentro del subsistema se hace uso de Github. Github es una plataforma de alojamiento de código para el control de versiones que permite la colaboración de diferentes integrantes de manera paralela en proyectos [18]. Github utiliza Git como sistema de control de versiones, permite alojar y compartir los repositorios de Git de manera remota, que son espacios donde se administra el código fuente y los archivos de un proyecto, además, proporciona herramientas adicionales. Es una de las herramientas más utilizadas en desarrollo de software para estas funcionalidades.

Git es un sistema de control de versiones distribuido disponible en todas las plataformas de desarrollo a través de una licencia de software libre, tiene la capacidad de gestionar copias privadas y combinarlas con los repositorios remotos. Permite realizar el seguimiento de los cambios realizados en un archivo a lo largo del tiempo. De esta manera se puede trabajar de manera conjunta entre todos los integrantes del equipo, y mantener copias de seguridad de las diferentes versiones. Además, ofrece una gran cantidad de funcionalidades que permite un control de versiones preciso.

Hyperloop UPV ha creado en Github una organización llamada "HyperloopUPV-H8" y en ella se almacenan los repositorios de cada uno de los subsistemas. El subsistema Software cuenta con seis:

- **HyperloopUPV-H8/h8-backend:** Incluye la implementación del back-end del equipo para establecer conexiones con el vehículo, recibir la información obtenida de los sensores, procesarla, darle formato y servírsela a las interfaces de usuario.

- **HyperloopUPV-H8/ethernet-front:** Consiste en una interfaz gráfica de usuario utilizada para realizar el testing de las placas y los distintos módulos del vehículo, con este programa se comprueba que el comportamiento del vehículo es adecuado. En él se gestionan los posibles errores y advertencias que puedan surgir. Además, permite enviar órdenes y valores para interactuar con las placas.

- **HyperloopUPV-H8/control-front:** Esta interfaz gráfica de usuario es la principal del equipo y se corresponde con la Control Station, en ella se desarrollan diferentes funcionalidades. En primer lugar, se permite el envío de órdenes al vehículo para manejarlo. En segundo lugar, se representa la información del estado del vehículo, corrientes, temperaturas, airgaps, etc. para conocer el estado en todo momento. En

tercer lugar, se dispone de una pestaña diseñada para representar la información de la infraestructura, los valores de presión del tubo, temperatura, etc. Además, hay una última pestaña que incluye una serie de cámaras situadas tanto en el tubo como en el vehículo para observar cómo actúa. Una versión limitada de esta aplicación se facilita al público durante el día de la competición para que permitan conocer el estado de la demostración en todo momento y mejore la experiencia de usuario.

- **HyperloopUPV-H8/common-front:** Repositorio creado para almacenar en él todos los componentes comunes de React que se utilizan en las diferentes interfaces de usuario que desarrolla el equipo. Este repositorio permite la reutilización de código de manera efectiva a través de su importación.

- **HyperloopUPV-H8/hil-gui:** Cuenta con tres módulos diferentes en su interior y engloba toda la lógica del testing del vehículo para las etapas finales del año. Cuenta con un back-end en golang que recibe información del vehículo a través de un programa intermedio llamado “HIL”, el back-end se encarga de procesar esta información y enviársela al front-end de este repositorio. Además, gestiona las órdenes que recibe del front-end para enviárselo al HIL y que estas se apliquen sobre el vehículo, y permite la posibilidad de simular perturbaciones para ver cómo reacciona el vehículo. El segundo módulo se corresponde con el front-end en React que recibe la información, la representa y permite enviar las órdenes. Cuenta además con representación 3D del funcionamiento del vehículo en tiempo real. Por último, este repositorio cuenta con un módulo en golang que simula el funcionamiento del vehículo y permite hacer testing del funcionamiento de este repositorio. De esta forma se simula el envío de información del vehículo, y permite la recepción de las órdenes enviadas por el usuario, obteniendo un feedback completo del funcionamiento de los tres módulos.

- **HyperloopUPV-H8/mobile-front:** Consiste en una versión limitada del control-front que adapta la visualización de los componentes para un dipositivo móvil con el objetivo de mejorar la experiencia de usuario del público durante la competición.

Además, se han creado diversos repositorios adicionales para testear y cumplimentar diferentes funcionalidades, por ejemplo, para procesamiento del vídeo de las cámaras, para la simulación de tráfico de la red o para la simulación de órdenes de perturbación del vehículo.

La metodología de trabajo en el desarrollo de código de Software es idéntica para todos los repositorios, se centra en el modelo alternativo de ramificación de Git llamado "Giflow". En este modelo, cada desarrollador crea una rama llamada “feature” donde realiza el trabajo necesario, y no es combinada con la rama principal hasta que se ha completado y revisado [10].

En este sistema de ramificación, en vez de tener una única rama “main”, se utilizan dos ramas diferentes para registrar el historial del proyecto. La rama “main” almacena el historial oficial de las diferentes versiones, mientras que la rama “develop” se utiliza como la rama de integración de las ramas “feature”. Los commits de la rama “main” se corresponden con diferentes versiones del programa y se guardan con un número.

Para iniciar el proyecto a través de la rama principal se crea una primera rama “develop” vacía, esta contendrá el historial del proyecto. A partir de este momento todos los desarrolladores pueden clonar el repositorio y empezar a crear ramas “feature” a partir de “develop” para empezar a desarrollar su contenido. Una vez el desarrollo de esta rama secundaria es terminada se realizará un merge con “develop” de nuevo para fusionar el contenido de ambas, en ningún caso deben interactuar con “main”.

Una vez "develop" ha adquirido una funcionalidad suficiente, es el momento de realizar un "release" para generar una nueva versión en la rama principal. Durante esta etapa, no se pueden añadir funcionalidades, tan solo se pueden corregir errores, generar documentación, etc. Una vez la rama "release" está lista para ser adherida a "main", se hace un merge para combinarlo, y se añade un número de versión. Además, se debe realizar un merge de nuevo con "develop", para combinar el desarrollo de ambas ramas. La utilización de una rama "release" adicional permite que el resto de los desarrolladores puedan seguir programando sobre "develop", pero es muy importante realizar el merge de vuelta.

En el caso de que los "releases" que se lleven a "main" contengan algún tipo de error, se crea una rama "hotfix" para tratarlo. Son las únicas ramas en los que se debe realizar un fork directamente sobre main. Luego debe fusionarse sobre "main" y sobre "develop" para conservar toda la funcionalidad añadida. Con estas ramas no es necesario interrumpir el workflow del equipo.

A continuación, en la Figura 2.28 se muestra un esquema que resumen el funcionamiento de este modelo de ramificación:

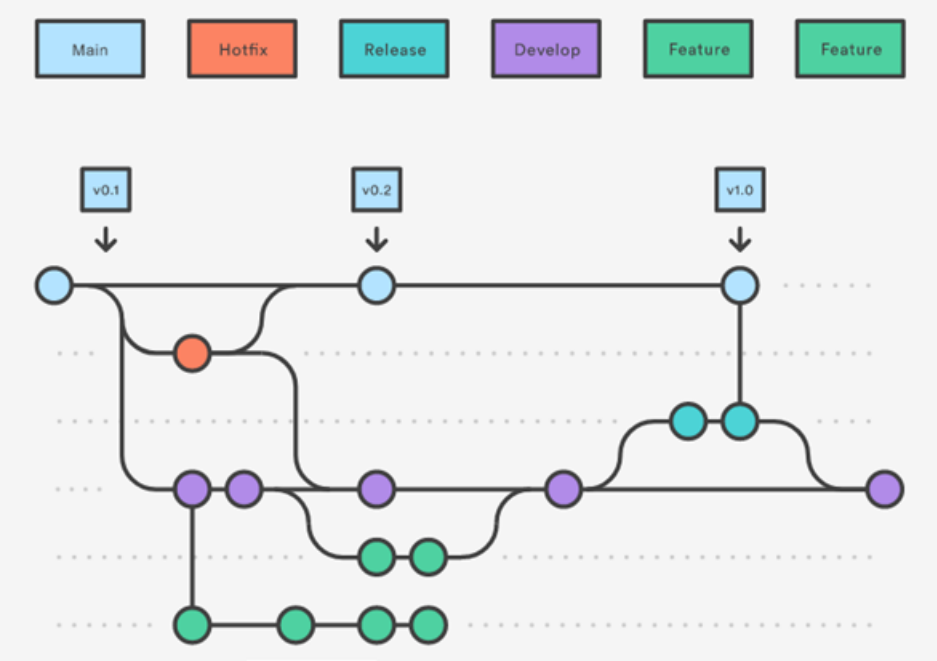

Figura 2.28: Diagrama del modelo de ramificación Gitflow
Fuente: Atlassian Git Tutorial [10]

Para el desarrollo de esta metodología por parte del equipo, se elaboran unos pequeños detalles adicionales con el objetivo de aumentar el orden y facilitar la gestión. En primer lugar, cada creación de una rama "feature" viene nombrada de la siguiente forma: "features/DEV-1", donde el número identifica a la rama y va incrementando a medida que se crean. Estas ramas numeradas se añaden en el Trello del subsistema, que es una herramienta que permite a los equipos gestionar el proyecto y su flujo de trabajo. En el Trello se indica su momento de creación, el proceso de desarrollo en el que se encuentra y cualquier observación de la que sea necesaria dejar constancia. A modo de ejemplo, en la Figura 2.29 se observa parte del Trello:

Una vez se ha completado la funcionalidad de esta rama, se procede a realizar un pull request. Un pull request es una petición del propietario de la rama de incorporar los commits de esta a la nueva rama donde se quiere fusionar el contenido realizando un merge. De esta manera, los integrantes del equipo revisan el código, comentan aquello

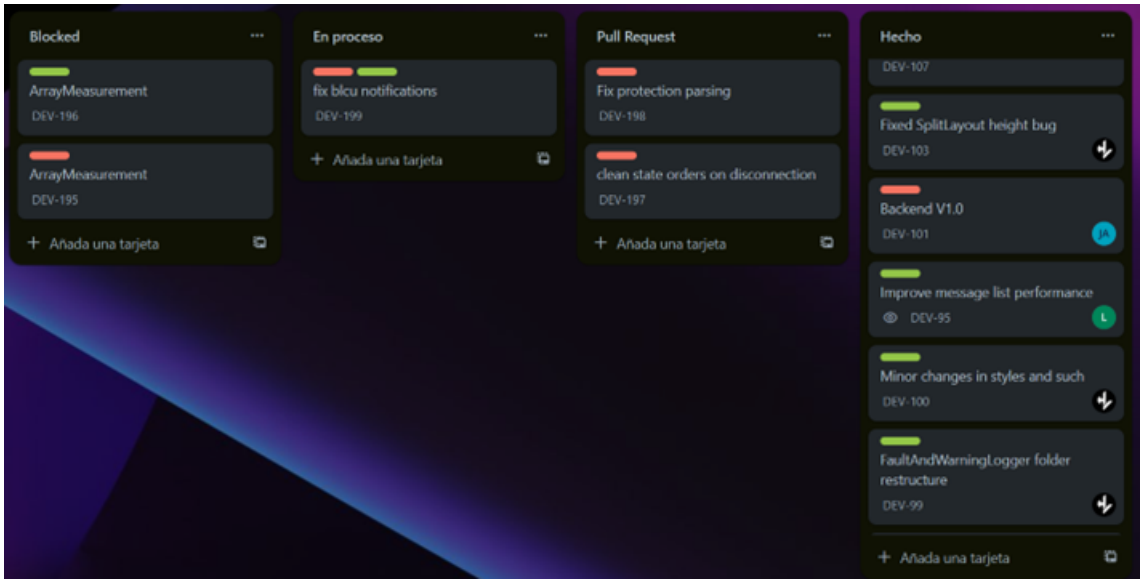

Figura 2.29: Trello del subsistema Hyperloop UPV
Fuente: Elaboración propia

que consideren, proponen cambios, y si están de acuerdo, aceptan el pull request. Cuando ha sido aceptado por el número de personas que se ha establecido en el proyecto (en el caso de los proyectos del subsistema software son dos personas, los otros dos integrantes del grupo), esta rama está lista para realizar el merge e incorporar el contenido a "develop". De esta forma todos los integrantes de Sotware son conscientes del trabajo realizado por el resto y revisan el código y su funcionalidad antes de incorporarlo a las ramas principales.

CAPÍTULO 3
Arquitectura de red

Este capítulo se centra en la planificación de la arquitectura de red utilizada por el equipo para la comunicación entre el vehículo, la infraestructura, la estación de control y, además, para proporcionar acceso a diversos usuarios y al público en general. En este proceso, se establecen los requisitos necesarios para el diseño, se justifica la elección de dicho diseño y se describen los métodos empleados para verificar su eficacia.

3.1 Introducción

Las redes proporcionan la infraestructura necesaria para que diferentes dispositivos se conecten e interactúen entre sí. Estos dispositivos finales, llamados hosts, establecen la comunicación entre direcciones que los identifican dentro de la red. Una vez identificados, dividen el mensaje en pequeños paquetes para facilitar las transmisión y los envían a través de la red. Esta comunicación está estandarizada utilizando protocolos que definen cómo la información debe ser gestionada. El protocolo más utilizado para este tipo de comunicación es el TCP/IP.

A lo largo de la red se encuentran dispositivos intermedios que recogen los paquetes generados por los hosts y los retransmiten a los hosts remotos. El dispositivo de red más simple utilizado en la actualidad es el switch, que permite que todos los dispositivos conectados físicamente intercambien paquetes entre sí. Está funcionalidad es limitada, por lo que para ampliar su alcance, se utilizan routers que toman decisiones más avanzadas para elegir la ruta óptima de cada paquete. Además, si la conexión por cable no es posible entre dos dispositivos, existe la posibilidad de la utilización de los puntos de acceso (NAP) que proporcionan conexión inalámbrica.

Existen más dispositivos de red, pero estos son los más utilizados. Para el sistema propuesto, en el que se conecta el vehículo, la Control Station y el público, no es necesario utilizar otros tipos de dispositivos de red.

3.2 Requerimientos

La red debe ser fiable en ambos sentidos entre el vehículo y la Control Station, así como la conexión con el público y el jurado. Para garantizar la seguridad de las demostraciones del vehículo, el equipo ha diseñado ciertos requerimientos:

- Un enlace inalámbrico de baja latencia inferior a 100 ms entre el vehículo y la Control Sation con un ancho de banda de al menos 100 Mb/s para garantizar que todo

el intercambio de información llega a tiempo. Este requisito viene impuesto por la regulación de la European Hyperloop Week directamente.

- Establecer el menor número de conexiones posible con la estación de control, aliviando la carga de las placas en el envío de datos e interceptando el tráfico intercambiado.
- Permitir la conexión de una cantidad elevada de clientes (se estima que es necesario un mínimo de 40 personas, un valor de 200 conexiones en paralelo es el idóneo para la competición) y permitir la transmisión de datos del vehículo y cámaras. Los datos transmitidos deben ser reducidos para evitar congestiones. Todas estas conexiones no deben alterar el comportamiento con el vehículo.
- Aislar la red de Internet para evitar posibles ataques.

3.3 Justificación del diseño

La arquitectura de red elegida está dividida en tres subredes independientes con el objetivo de que cada una de ellas se centre en una tarea, facilitando en gran medida la gestión y la configuración de la red global. En la Figura 3.1 se observa la división establecida: la subred del vehículo, la subred de la infraestructura y la subred del público.

Figura 3.1: Visión general de la infraestructura de red
Fuente: Hyperloop UPV H8

3.3.1. Unidades de control del vehículo

El vehículo cuenta con siete unidades de control encargadas de asegurar el correcto funcionamiento y manipulación del vehículo y sus sistemas. Estas unidades deben poder comunicarse entre sí y con el exterior, por lo que están incluidas en la subred del vehículo. Alguna de las unidades de control cuenta con más de una placa asociada. La función de cada una de las unidades de control es la siguiente:

- **Vehicle Control Unit (VCU):** Es la unidad responsable de centralizar la comunicación y gestionar los sistemas de navegación. Participa como la placa maestra en la comunicación de las placas.
- **Bootloader Control Unit (BLCU):** El vehículo cuenta con un Bootloader inalámbrico que ofrece una forma segura de actualizar el firmware de las unidades de control sin necesidad de acceder físicamente a las placas y tener que desconectar cables de alto voltaje. Con esta placa se consigue un método fiable para esta operación inalámbrica.
- **Battery Management System Low-Voltage (BMSL):** Su función es asegurar un suministro de energía estable y seguro para todas las unidades de control que se alimentan a través de la batería de bajo voltaje.

- **Battery Management System High-Voltage (BMSH):** Es la placa responsable de monitorizar el estado de carga y de salud de las baterías de alto voltaje y equilibrar las celdas para garantizar un funcionamiento seguro y eficaz. Esta unidad se conecta con la OBCCU, por lo que no ocupa ningún puerto del switch.
- **On Board Charger Control Unit (OBCCU):** Es la placa responsable de controlar todos los elementos de protección del circuito de distribución de energía, medir el aislamiento entre las baterías y el chasis, obtener información sobre el estado de las baterías de alto voltaje y medir las corrientes. Además, también se encarga de controlar la unidad de alimentación del cargador a bordo (OBCPU).
- **Levitation Control Unit (LCU):** Recibe la información de todos los datos del control de levitación y los sensores de seguridad. Además, aplica los algoritmos de control implementados para manipular la levitación. Este sistema está formado por dos placas, la master y la slave, ambas se incluyen y comunican en la subred del vehículo, la placa slave envía la información exclusivamente a la master, y es esta la que se comunica con el resto.
- **Propulsion Control Unit (PCU):** Ejecuta algoritmos de control de la propulsión para gestionar las dos unidades de alimentación de la propulsión con las que cuenta el vehículo (PPU).

3.3.2. Subred del vehículo

Esta subred es la encargada de la comunicación de todas las placas de control, establece la conexión entre las siete placas del vehículo y el resto de la red. La Figura 3.2 muestra un diagrama de estas conexiones:

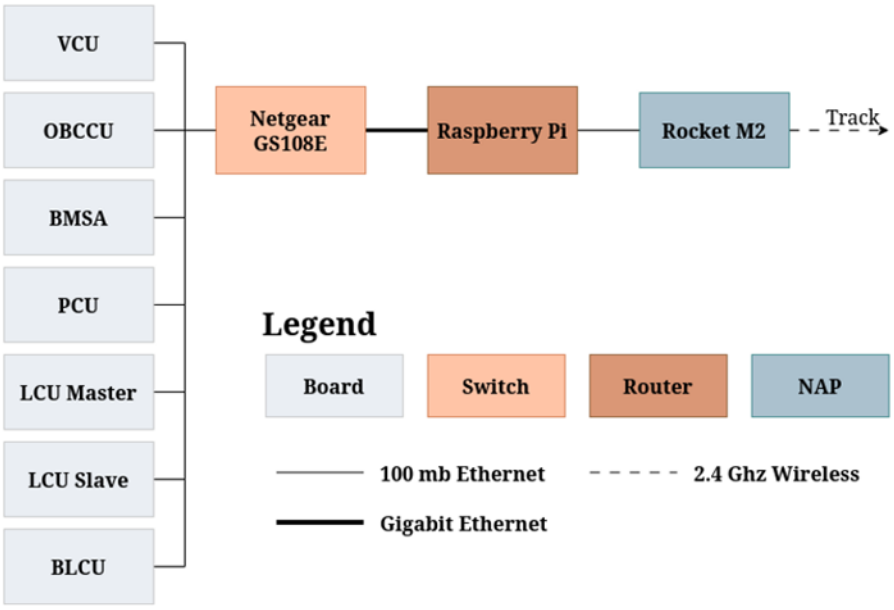

Figura 3.2: Visión general de la subred del vehículo
Fuente: Hyperloop UPV H8

Todas las placas utilizan Fast Ethernet, por lo que su ancho de banda está limitado a 100 Mb/s cumpliendo el requisito obligatorio de la competición. Además, en algunos casos, este valor se excede.

La información no se envía directamente a la Control Station, los mensajes intercambiados entre las placas necesitan ser interceptados y enviados a una diferente subred. La

solución para esto es utilizar Encapsulated Remote Switch Port Analyzer (ERSPAN) en el switch. ERSPAN funciona duplicando todos los paquetes de entrada en ciertos puertos del switch y enviándolos a través de un túnel. De esta forma, los paquetes pueden ser recibidos en una subred diferente. Un túnel crea una conexión virtual entre dos dispositivos de la red utilizando un protocolo de tunelización. El subsistema utiliza el protocolo IP in IP (IPIP), que es uno de los protocolos más simples pero efectivos para la tunelización. Cuando un paquete pasa a través de un túnel IPIP, todos los datos y su cabecera IP anterior se colocan por detrás de una nueva cabecera IP con el origen y el destino establecidos en los dos puntos del túnel (Figura 3.3).

Figura 3.3: Encapsulación IP in IP
Fuente: Hyperloop UPV H8

Algunos switches Cisco proporcionan este servicio, sin embargo, las alternativas comerciales existentes no se ajustaban a las necesidades del equipo, o bien por su volumen o peso, o por su precio. Por esa razón, se ha optado por replicar ERSPAN manualmente con otros dispositivos.

El switch elegido para conectar las siete unidades de control del vehículo entre sí es el Netgear ProSafe GS108E v3. Contiene ocho puertos que proporcionan gestión en la capa de enlace, la Figura 3.4 muestra el dispositivo. Con él, se puede configurar "port mirroring" para duplicar todo el tráfico entre las placas al puerto restante, donde se encapsula para atravesar las diferentes subredes.

Figura 3.4: Switch Netgear ProSafe GS108E v3
Fuente: Hyperloop UPV H8

Este último puerto del switch se conecta a una Raspberry Pi 4 B+. La función de este ordenador monoplaca (Single Board Computer, SBC) consiste en interceptar todo este tráfico utilizando el modo promiscuo y renvía los paquetes por el túnel IPIP para llegar a la Control Station. Además, este dispositivo cumple con la funcionalidad de router y cortafuegos, impidiendo que lleguen paquetes no deseados a las placas y se comuniquen con el vehículo y la subred Track, en la Figura 3.5 se muestra la Raspberry utilizada.

El funcionamiento básico de este dispositivo se representa en la Figura 3.6. El diagrama está simplificado para una mejor comprensión y describe las decisiones que toma cuando recibe un paquete a través de sus conexiones. Básicamente, se comprueba si los paquetes que llegan de alguno de los puertos de las unidades de control son duplicados; si alguno lo es, quiere decir que va dirigido al exterior y lo envía hacia la Raspberry

Figura 3.5: Raspberry Pi 4 B+
Fuente: Hyperloop UPV H8

Pi. Los paquetes que no se han duplicado se enrutan según las cabeceras originales y se dirigen donde es necesario; de igual manera se gestionan los paquetes que vienen de la Raspberry Pi. Se ha de tener en cuenta que se ha prestado especial atención a las reglas de cortafuegos y otras políticas para asegurar la red contra paquetes maliciosos.

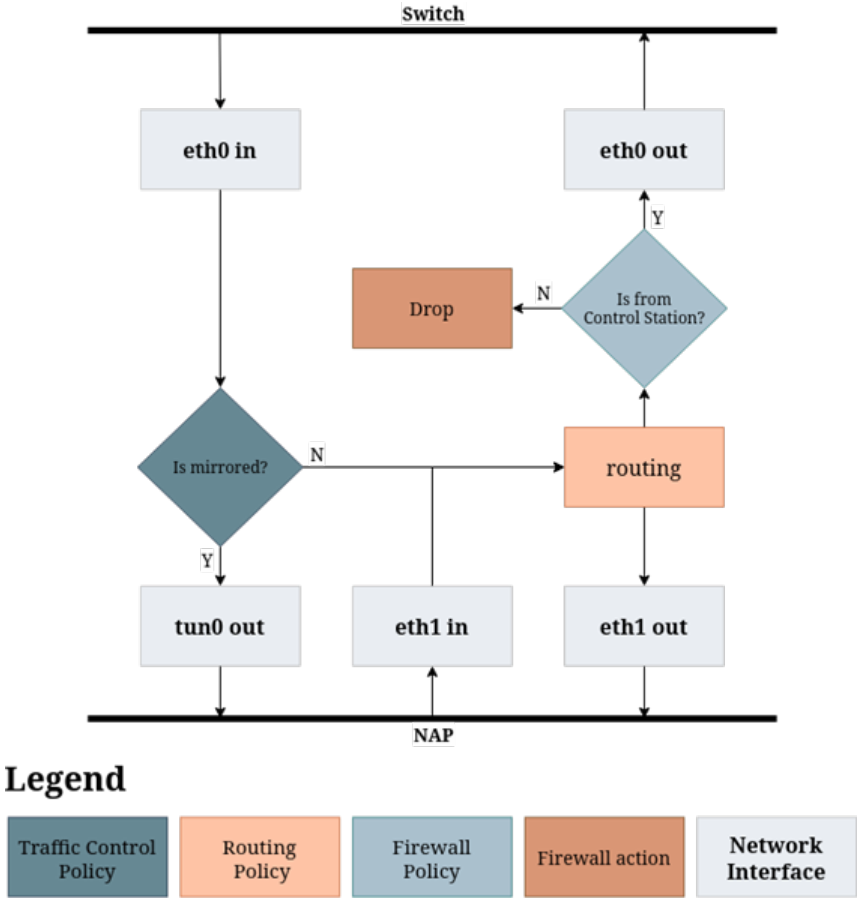

Figura 3.6: Diagrama de decisión de la Raspberry Pi
Fuente: Hyperloop UPV H8

Es necesario un punto de acceso que conecte de forma inalámbrica la subred del vehículo con el exterior, y concretamente, con la subred Track. Para ello se ha utilizado el punto de acceso inalámbrico Point-to-Point Ubiquiti Rocket M2 NAP, de esta forma toda la subred es accesible desde el exterior de la infraestructura, aunque esta se encuentra

cerrada en condiciones de bajas presiones. En la Figura 3.7 se muestra este dispositivo, que completa la subred del vehículo.

Figura 3.7: Punto de acceso inalámbrico Ubiquiti Rocket M2
Fuente: Hyperloop UPV H8

3.3.3. Subred de la infraestructura

La subred de la infraestructura o subred Track es la subred troncal de la arquitectura. En ella se encuentra la Control Station donde se ejecuta la aplicación de control "control-front" y las aplicaciones de testing, desde donde el equipo manipula el vehículo. Además esta subred gestiona el streaming de cámaras y la placa Tube Control Unit (TCU), que controla todos los sensores, actuadores y elementos situados en la infraestructura. La distribución de la subred está representada en la Figura 3.8.

Gran parte de la subred soporta Gigabit Ethernet, por lo que provee un ancho de banda holgado para la retransmisión de todos los datos de las cámaras y datos del vehículo.

En cuanto a la selección de cámaras, inicialmente se barajaron alternativas con retransmisión en formato comprimido. Sin embargo, las mejores alternativas con un precio económico introducen una latencia de al menos 150 milisegundos, superando los requisitos impuestos por la competición. Dado el presupuesto limitado se prescindió de gamas más altas y se optó por Elgato Facecam (véase la Figura 3.9), estas cámaras retransmiten en un formato sin comprimir, YUV 420, a través de USB 3.0. Esta retransmisión debe ser difundida a la Control Station y a los diferentes dispositivos tanto de la audiencia como del jurado. La retransmisión de un vídeo sin comprimir es inviable, se ha comprobado a través de numerosas actividades de testing que cada cámara puede generar hasta 238 Mb/s, esta transferencia de datos es completamente excesiva si se desea manipular numerosas conexiones. Por ello, cada cámara tiene asignado un computador monoplaca que utilizan FFMPEG para codificar el vídeo a un protocolo de tiempo real (RTP), que luego se envía a un servidor WebRTC que gestiona las conexiones entre pares, de esta forma se comprime mediante H.264 limitando la transferencia a una tasa de bits objetivo de 1 Mb/s, permitiendo un gran número de conexiones simultáneas.

La Control Station sirve la interfaz de control, la interfaz pública y aloja el back-end. Para evitar que usuarios no autorizados se hagan con el control del vehículo, la interfaz de control principal se sirve en localhost o, si es necesario que varios hosts visualicen los

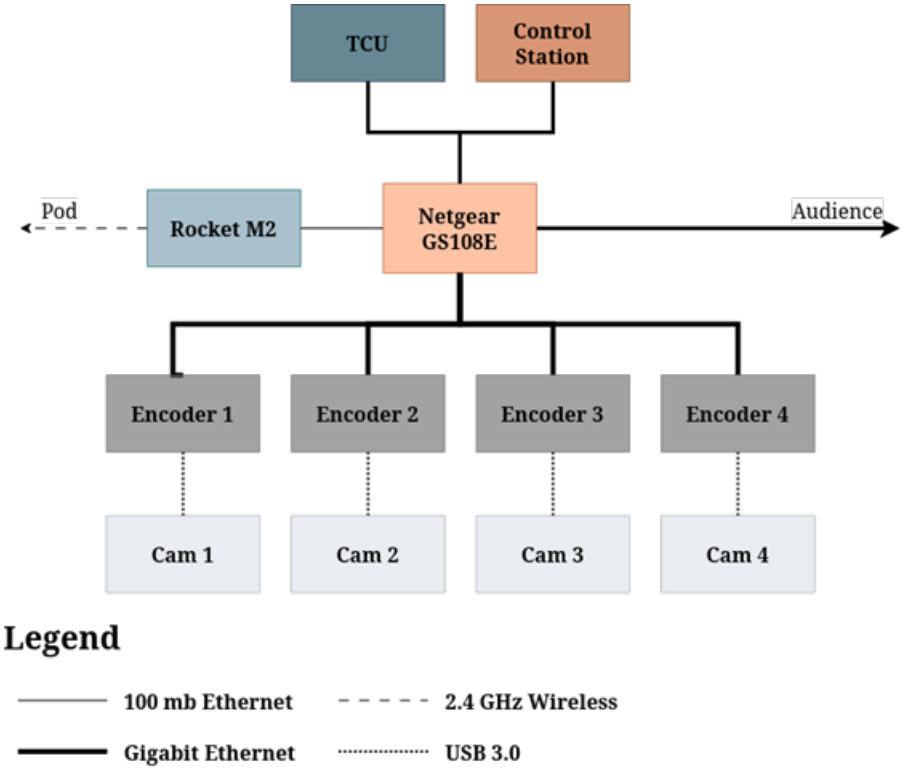

Figura 3.8: Visión general de la subred de infraestructura
Fuente: Hyperloop UPV H8

Figura 3.9: Elgato Facecam
Fuente: elgato.com

datos de control, en una tarjeta de interfaz de red (NIC) diferente solo accesible por personas autorizadas. Estas aplicaciones no tienen conexión a Internet para evitar ataques.

3.3.4. Subred de la audiencia

Esta subred proporciona un punto de acceso inalámbrico accesible en los alrededores de la infraestructura donde se va a realizar la demostración durante la competición, con el objetivo de que el público y el jurado se pueda conectar para observar las cámaras y acceder a la información del vehículo en tiempo real. El punto de acceso inalámbrico es proporcionado por tres Ubiquiti Rocket M2 en una configuración de malla, y proporcio-

nan una WIFI pública para que todos los asistentes pueden conectarse (véase la Figura 3.10). El número de puntos de acceso (NAP) ha sido elegido según las estimaciones y pruebas realizadas por el equipo. Sin embargo, este número es modificable según la necesidad real, pudiendo añadir más puntos de accesos para expandir la red o disminuir el número. En el caso de que el número de conexiones aumente y sea necesaria mayor potencia, se estudiará la opción de sustituir la malla de los puntos de accesos actuales por un dispositivo nuevo, se ha estudiado el punto de acceso Ubiquiti U6 Lite que permite Gigabit Ehternet y cubre un área mucho mayor que el de la infraestructura.

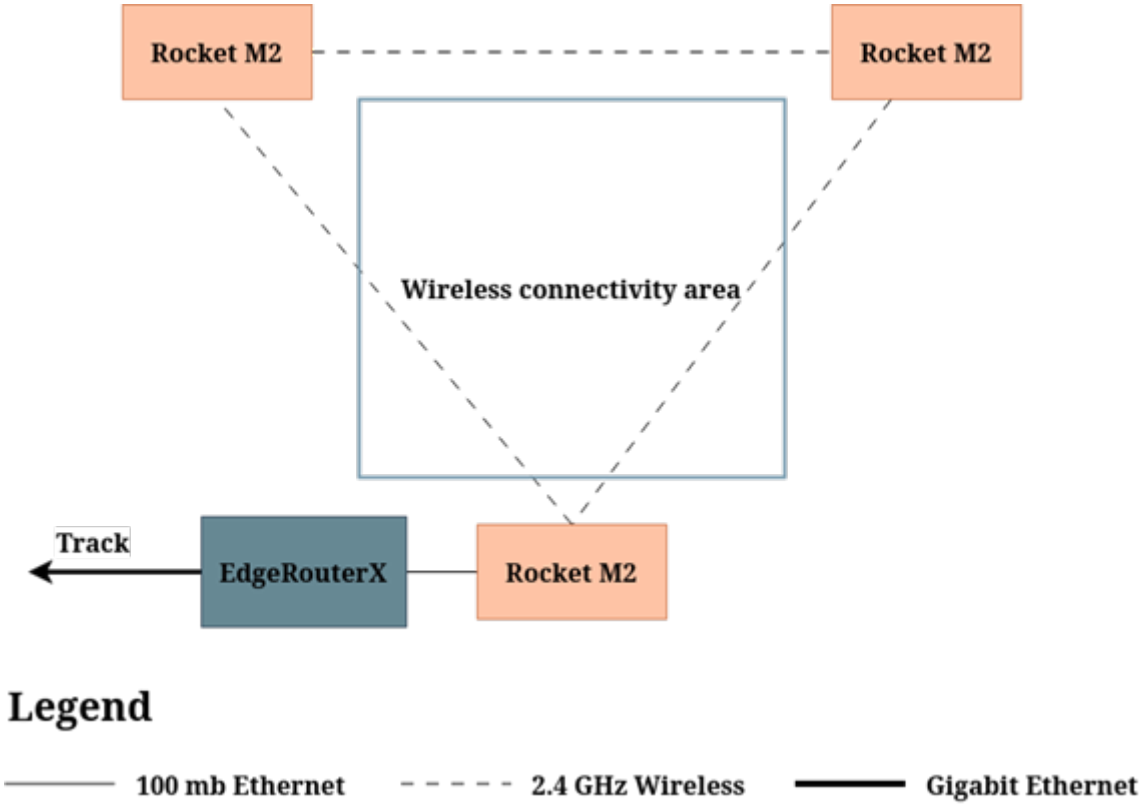

Figura 3.10: Visión general de la subred de audiencia
Fuente: Hyperloop UPV H8

Para conectar la subred de la audiencia con el de la infraestructura se hace uso del router Ubiquiti EdgeRouter X (véase la Figura 3.11). A este router se le pueden incorporar los puntos de acceso inalámbricos que sean necesarios, con la opción "bonding" se facilita su acceso a través de una única interfaz y una única SSID, conectándose al punto de acceso que mayor señal proporcione, de esta manera se mejora el rendimiento total y no supone una dificultad añadida para el público que no tiene que elegir qué punto de acceso utiliza y la distancia a la que se encuentra de él.

Este router incorpora un servidor DHCP (Dynamic Host Configuration Protocol) para que se asignen las direcciones IP automáticamente a los dispositivios que se conecten a la red, y un servidor DNS (Domain Name System) para traducir las direcciones IP de los servidores a nombres de dominio. De esta forma los dispositivos se pueden conectar a la red sin tener que realizar ningún tipo de configuración y se permite su acceso a la interfaz web que se proporciona.

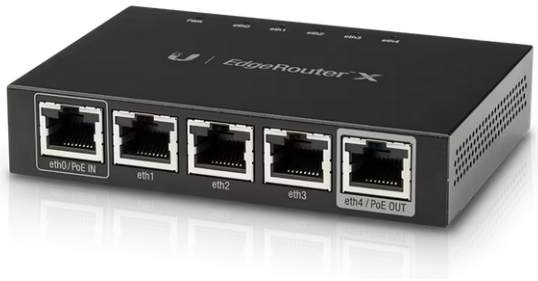

Figura 3.11: Ubiquiti EdgeRouter X
Fuente: ui.com

3.3.5. Validación de la red

La única forma de asegurarse que esta red funciona correctamente es a través de testing. Para ello, se ha realizado pruebas generando información simulada y comprobando las pérdidas de paquetes, analizando la aceptación y respuesta los diferentes dispositivos.

Aunque se puede simular la red al completo, dado el enfoque modular que se ha desarrollado en las tres diferentes subredes, se pueden realizar pruebas de manera individualizada, asegurando su funcionamiento de manera aislada al resto. Una vez se comprueba su correcto funcionamiento, se puede realizar la parte de validación de la red completa.

Los requerimientos funcionales que deben ser alcanzados por cada una de las subredes son las siguientes:

- La información que se transmite desde la subred del vehículo hacia la Control Station es de aproximadamente 90 Mb/s, y el ancho de banda máximo de la subred es de 100 Mb/s, por lo que se cumple. Se ha analizado que los picos de transferencias no provocan retardos ni sobrecargas.

- La subred Track gestiona la información y las cámaras transmiten a cada conexión de la audiencia 1 Mb/s de datos del vehículo, y aproximadamente 1 Mb/s por la retransmisión de cada una de las cámaras, es decir, un total de 5 Mb/s para cada conexión. En esta comunicación el elemento de la red que hace de cuello de botella es la conexión de los puntos de acceso inalámbricos, ya que el resto utilizan Gigabit Ethernet. Como están dispuestos en malla conectados al router, se puede gestionar en paralelo alrededor de 300 Mb/s teóricos. Por lo tanto, se prevé una conexión adecuada de un máximo de 60 personas. Si finalmente es necesario aumentar ese valor, el punto de acceso alternativo descrito anteriormente dispone de Gigabit Ethernet, permitiendo un máximo de 200 personas.

Para automatizar los tests, se utiliza software de generación de tráfico que puede reemplazar parte del equipamiento y simplificar los tests. A través de un ordenador conectado a la red mediante varias tarjetas de interfaz de red se ejecutan múltiples pruebas enviando diferentes datos similares a una demostración real y comprobando que la respuesta coincide con la esperada.

Se han realizado pruebas comparativas en la red, principalmente con pruebas de estrés, en ellos se simularon hosts hipotéticos mediante ordenadores y se supervisó todo el tráfico a través de Wireshark. Se ha de garantizar que incluso con cargas elevadas, la red funciona correctamente.

Los resultados obtenidos muestran que incluso si Kenos utiliza la totalidad de los 100 Mb/s del ancho de banda del que dispone la Control Station, no se pierde ningún dato. Por parte de la conexión con el público, se ha comprobado el mayor soporte con una comunicación fluida y sin pérdidas, que es de alrededor de 55 conexiones y se valora la viabilidad de comprar el punto de acceso alternativo para aumentar este valor.

CAPÍTULO 4
Back-end

El back-end es la parte de la aplicación que gestiona toda la lógica y el procesamiento de datos. En el caso del back-end diseñado para la Control Station, es el software que recibe todos los paquetes duplicados de la red y los procesa para gestionar estructuras de información que pueden ser registrados en archivos o mostrados en el front-end. Además, sirve la web que se utiliza como front-end y gestiona las conexiones con los clientes. De esta forma, no solo la Control Station puede visualizar los datos, sino que también está abierto para dispositivos del equipo, el público y el jurado.

4.1 Requerimientos

El diseño de una aplicación con esta funcionalidad presenta una serie de requerimientos que se analizan en esta sección.

Los requerimientos del back-end están centrados en el correcto procesamiento de la información recibida de la red anteriormente descrita y sus requisitos para compartirlo con todos los usuarios:

- Las placas se comunican utilizando Ethernet, que permite unas velocidades de comunicación de hasta 100 Mb/s dentro del vehículo. Toda esta información se duplica y envía al back-end, por lo que debe ser capaz de procesar la totalidad de los datos recibidos, sin perder ningún paquete.
- La Control Station debe ser capaz de enviar órdenes al vehículo, para ello es necesario establecer una conexión TCP que se conecte con la placa del vehículo VCU (Vehicle Control Unit).
- Es necesario enviar archivos .hex a la placa BLCU, que es la placa encargada de cargar ese código en cualquiera de las placas que sea necesario. Para ello es necesario establecer una conexión TFTP.
- El back-end debe funcionar como servidor del front-end tanto para el equipo como para la audiencia, por lo tanto, debe ser capaz de gestionar peticiones HTTP para facilitar la página web.
- El front-end se actualiza a través de conexiones WebSocket, el back-end se encarga de gestionar todas estas conexiones (una conexión por cliente) y compartir los datos con todo el mundo.
- Se espera que el número de conexiones el día de la competición sea de alrededor de 200. Debe ser capaz de gestionar todo este tráfico y compartir a cada conexión

una cantidad mínima de información que garantice una experiencia fluida pero completa.

4.2 Justificación del diseño

Para cumplimentar todos los requerimientos, el back-end se ha diseñado siguiendo una arquitectura modular y concurrente. El código se divide en módulos independientes que realizan una tarea concreta. Cada módulo contiene sus propios modelos y ofrece una Interfaz de Programación de Aplicaciones (API) para operar con su funcionalidad en el exterior, esta "librería" que ofrece cada módulo es mínima para facilitar su utilización y reducir el acoplamiento. De esta manera, solo se hacen públicas las funciones y propiedades necesarias, ocultando los detalles de implementación que son innecesarios. La modularidad establecida permite que la arquitectura se adapte con facilidad a los cambios de implementación que surjan durante el desarrollo del proyecto, un cambio en uno de los módulos debe ser indiferente para el resto.

La concurrencia del código permite que el programa ejecute diversas tareas al mismo tiempo en vez de realizarlo de forma secuencial. De esta manera, el programa salta a través de diferentes rutinas, ejecutándolas cuando es necesario. Cada rutina es independiente del resto y por lo tanto todas pueden ser ejecutadas de forma simultánea. Si dos rutinas necesitan compartir información, hacen uso de canales o memoria compartida. Las principales rutinas del back-end son: lectura de paquetes duplicados de la red ("sniffing"), análisis de los paquetes, registro la información, servicio al front-end y gestión de todas las conexiones con los clientes.

4.2.1. Información general del repositorio

El repositorio del back-end está ubicado en la organización "HyperloopUPV-H8" de Github y tiene como nombre "h8-backend"[1]. Cuenta con cuatro contribuidores formados por el subsistema de software del equipo y un colaborador de la edición anterior. El repositorio consta de 348 commits y 121 ramas. El lenguaje predominante es Golang con un 98.8 %, el resto está desarrollado en Shell para algún script específico. El back-end ha tenido varias refactorizaciones y modificaciones menores desde su creación en octubre de 2022 para adaptarse a las nuevas necesidades surgidas en el equipo; la versión descrita en este documento no tiene por qué ser la definitiva y puede estar sujeta a ligeros cambios. La organización ha decidido dejar el repositorio open source, por lo que es accesible por cualquier usuario. Cuenta con ciertos aspectos de mejora no prioritarios que se han dejado sin implementar dada la carga de trabajo y la limitación de personal. El grueso del desarrollo de este repositorio se ha realizado entre octubre de 2022 y enero de 2023.

4.2.2. Herramientas utilizadas

Dadas las características necesarias para el desarrollo de este código, se ha optado por desarrollarlo en el lenguaje Golang (Go) 4.1. Go es un lenguaje de programación de alto rendimiento con soporte integrado para la concurrencia. Cuenta con una amplia librería de paquetes, tanto oficiales como desarrollados por la comunidad, que simplifican el proceso de desarrollo, por ejemplo para el desarrollo de las conexiones WebSocket. Además, tiene una curva de aprendizaje suave que permite una rápida adaptación al lenguaje.

[1] El enlace del repositorio de Github "h8-backend" es https://github.com/HyperloopUPV-H8/h8-backend

Figura 4.1: Lenguaje Golang
Fuente: go.dev

La elección de Golang para el back-end de esta edición ha supuesto un cambio respecto al año anterior, donde se utilizó Rust. Las principales razones para este cambio han sido:

- **Simplicidad y facilidad de aprendizaje:** Go se caracteriza por una sintaxis simple y legible, tiene una curva de aprendizaje mucho más suave que Rust. Para un subsistema donde la mayoría de los miembros se han incorporado este año y no cuentan con un periodo de adaptación, permite una productividad mucho mayor y más rápida, y un mejor desarrollo del proyecto.
- **Concurrencia y paralelismo sencillo de controlar:** Go tiene un soporte nativo para la concurrencia mediante goroutines y canales. Esto facilita mucho el trabajo del equipo, ya que necesita de procesamiento simultáneo de múltiples tareas, además, es muy útil para gestionar las conexiones del servidor web.
- **Nivel de eficiencia y rendimiento adecuado para el objetivo:** Go destaca por su tiempo de ejecución altamente optimizado y su gestión eficiente de la concurrencia a través de goroutines. Permite construir sistemas escalables y con alta concurrencia.
- **Amplia biblioteca estándar:** Go incluye una biblioteca estándar rica que abarca numerosos aspectos del desarrollo de aplicaciones. Evita que se tengan que utilizar gran cantidad de dependencias externas, por lo tanto, acelera el desarrollo.
- **Comunidad y ecosistema:** Go cuenta con una comunidad activa grande y un ecosistema creciente de bibliotecas y herramientas, que facilitan el desarrollo.

Rust es un lenguaje muy poderoso, más cercano al hardware y con un enfoque más estricto en la seguridad y el control del sistema y los recursos. Sin embargo, las ventajas que aporta este lenguaje no son sumamente importantes en el proyecto, y la curva de aprendizaje es más pronunciada. Por lo tanto, el cambio de lenguaje a Golang permite desarrollar un programa que cumpla perfectamente con los requerimientos necesarios, pero suponiendo una menor complicación que la edición anterior.

Además, para el desarrollo de este programa se ha hecho uso de Google Drive y sus Hojas de Cálculo para la configuración de los tipos de paquetes. La validación para las descargas de los ficheros de Google se han realizado a través de archivos JSON. Por otra parte, las variables de entorno del programa se han gestionado inicialmente con un archivo “.env”, que han sido sustituidas por ”config.toml” en las etapas finales.

4.3 Arquitectura del back-end

La funcionalidad del back-end está centrada en la gestión de las conexiones con el vehículo y la subred, la recepción de mensajes, el procesamiento de la información, y

la conexión con los clientes que van a hacer uso de esta información. Estas tareas están formadas por múltiples módulos independientes que facilitan su implementación y manejo. El objetivo de una arquitectura modular es la de independizar secciones de código no relacionados. Esta división facilita el trabajo de los desarrolladores, minimiza los cambios tras alteraciones en el diseño y permite un código reutilizable y más predecible. Esta modularidad supone un aumento de la sobrecarga del código para la integración de estas secciones, sin embargo, este inconveniente es insustancial comparado con las ventajas que ofrece. En la Figura 4.2 se puede observar la distribución de los módulos y la relación entre ellos, es un esquema simplificado, algunos módulos cuentan a su vez con submódulos desacoplados para mejorar la independencia de los fragmentos de código. A continuación se describe cada uno de los módulos que describen el back-end.

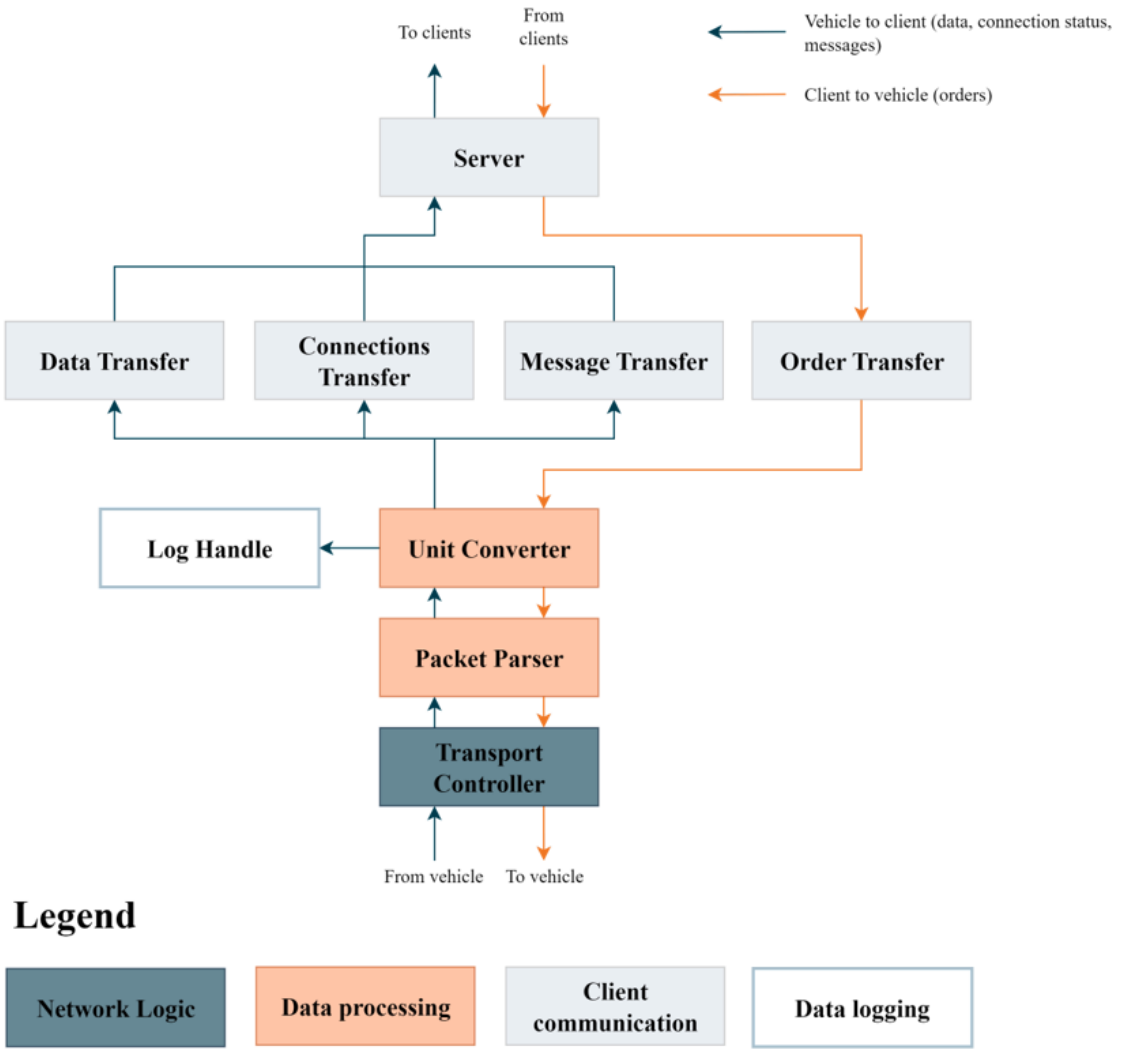

Figura 4.2: Estructura de los módulos del back-end
Fuente: Hyperloop UPV H8

4.3.1. Descripción de módulos

Transport Controller

Es el módulo más cercano a la capa de transporte, está dividido en dos partes lógicas: el Sniffer y el Pipe Handler. El sniffer se encarga de leer todo el tráfico duplicado de la subred del vehículo, como por ejemplo, cualquiera de los paquetes que envía una placa a

otra. Para ello, elimina la encapsulación mencionada anteriormente y filtra los paquetes según su origen y destino IP, cuando ambos se corresponden con las placas del vehículo, se envía esta información a través de un canal para que sea procesado por el resto de módulos. El Sniffer utiliza pcap, la misma tecnología que tienen otras aplicaciones de monitorización de red como Wireshark.

Se ha definido la estructura "Sniffer", este objeto cuenta con el atributo “source” del tipo *pcap.Handle perteneciente a la librería "github.com/google/gopacket/pcap".

```
type Sniffer struct {
    source *pcap.Handle
    filter string
    config Config
    trace  zerolog.Logger
}
```

El objeto ofrece la interfaz para capturar y procesar paquetes de red utilizando libpcap, una biblioteca de captura de paquetes en red multiplataforma. Concretamente se utiliza pcap.Handle que proporciona una variedad de métodos para abrir la interfaz de captura de paquetes, filtrar paquetes mediante reglas de Berkeley Packet Filter, obtener contenido de los paquetes, etc. Esta estructura “Sniffer” cuenta con un filtro de tipo string para seleccionar los paquetes de interés, esta cadena de texo es introducida en el pcap.Handle.SetBPFFFilter para que compile un BPF filter en esta estructura, de esta manera se previene de capturar tráfico no deseado. Este string se crea con métodos personalizados del equipo tanto para el encapsulamiento IPIP como para las conexiones TCP y UDP , que son seleccionados a través de métodos realizados por el equipo. A modo de ejemplo se detalla el método que crea la cadena de texto para el filtro TCP:

```
func getTCPFilter(addrs []net.IP, serverPort uint16,
clientPort uint16) string {
    ports := fmt.Sprintf("tcp port %d or %d", serverPort, clientPort)
    notSynFinRst := "tcp[tcpflags] & (tcp-fin | tcp-syn | tcp-rst) == 0"
    notJustAck := "tcp[tcpflags] | tcp-ack != 16"
    nonZeroPayload := "tcp[tcpflags] & tcp-push != 0"

    srcAddresses := common.Map(addrs, func(addr net.IP) string {
        return fmt.Sprintf("(src host %s)", addr)
    })

    srcAddressesStr := strings.Join(srcAddresses, " or ")

    dstAddresses := common.Map(addrs, func(addr net.IP) string {
        return fmt.Sprintf("(dst host %s)", addr)
    })

    dstAddressesStr := strings.Join(dstAddresses, " or ")

    filter := fmt.Sprintf("(%s) and (%s) and (%s) and (%s) and (%s) and (%s)",
    ports, notSynFinRst, notJustAck, nonZeroPayload, srcAddressesStr,
    dstAddressesStr)
    return filter
}
```

Además, el sniffer cuenta con una estructura propia con el nombre Config que guarda etiquetas del cliente, servidor, interfaz, etc. Por último, se ha creado a partir de la librería

”github.com/rs/zerolog” un trace que imprime por consola los avisos y errores e información genérica para mejorar la experiencia del usuario.

Al inicio del main se crea el sniffer inicializándolo con su respectivo constructor. Una vez creado, se puede acceder a todos los métodos públicos, como por ejemplo "Listen", para obtener toda la información de los paquetes recibidos y dirigirlos al resto de módulos. Esta práctica de instanciar el objeto en el programa main y utilizar sus métodos públicos se realiza de igual forma en el resto de módulos.

El Pipe Handler gestiona las conexiones TCP establecidas con las placas y proporciona información sobre el estado de estas conexiones. Para ello se ha creado un método que devuelve un mapa con clave string y valores de tipo Pipe con información de las direcciones, lectores e información del estado de la conexión, para cada una de las direcciones de las placas que envían información que recibe el back-end a través del sniffer, cada "pipe-epresenta una conexión TCP individual, con un puntero a un TCPConn, si este puntero es nulo es que la placa está desconectada. De esta forma se facilitan métodos de lectura para cada conexión y también se pueden enviar arrays de bytes a través de las conexiones TCP que mantiene la estructura Pipe en su atributo “conn”, para comunicarse con las placas. La información de lectura se mantiene almacenada en el canal (chan) “output”, que es un buffer que acumula las estructuras definidas en su inicialización, en este caso packet.Packet. El resto de módulos acceden a este canal para manipular los paquetes.

```
type Pipe struct {
    conn *net.TCPConn

    laddr *net.TCPAddr
    raddr *net.TCPAddr

    readers map[uint16]common.ReaderFrom

    isClosed bool
    mtu      int

    output             chan<- packet.Packet
    onConnectionChange func(bool)

    keepaliveInterval *time.Duration
    writeTiemout      *time.Duration

    trace zerolog.Logger
}
```

Packet Parser

Este módulo es utilizado para transformar la información de los paquetes en estructuras de datos del back-end, para ello, recibe la información capturada por el sniffer y la transforma. Cada paquete se identifica con un id único en sus dos primeros bytes. Con este identificador, el módulo packet parser identifica el tipo de información en unas tablas alojadas en un documento Excel preparado por Avionics, montando la estructura correspondiente. Su funcionamiento incluye las acciones en sentido contrario, a partir de una estructura de datos, el módulo puede transformarlo en un paquete de bytes que puede entender el vehículo, este sentido es utilizado en el envío de órdenes.

El struct definido utiliza el mismo nombre del módulo, "PacketParser". El campo "structures" asigna el identificador de un paquete a una lista de "ValueDescriptor", que describe las propiedades, su nombre y el tipo de datos que contiene. El atributo "valueParsers" asigna los tipos de datos a partir de su nombre como cadena de texto a una interfaz llamada "parser" que tiene los métodos de decodificar y codificar de cada tipo. Además cuenta con un objeto de tipo "Config" que almacena configuraciones específicas y etiquetas.

```
type PacketParser struct {
    structures   map[uint16][]packet.ValueDescriptor
    valueParsers map[string]parser
    config       Config
}
```

Con el objetivo de describir el funcionamiento de la decodificación y codificación de los paquetes se describe el siguiente fragmento de código a modo de ejemplo:

```
type enumParser struct {
    descriptors map[string][]string
}

func (parser enumParser) decode(descriptor packet.ValueDescriptor,
order binary.ByteOrder, data io.Reader) (packet.Value, error) {
    enum, ok := parser.descriptors[descriptor.Name]
    if !ok {
        return packet.Enum("Default"), fmt.Errorf("enum descriptor for %s
        not found", descriptor.Name)
    }

    var value uint8
    err := binary.Read(data, order, &value)
    if err != nil {
        return packet.Enum("Default"), err
    }

    if len(enum) < int(value) {
        return packet.Enum("Default"), fmt.Errorf("option slice has length %d
  but index is %d", len(enum), value)
    }

    return packet.Enum(enum[value]), nil
}

func (parser enumParser) encode(descriptor packet.ValueDescriptor,
order binary.ByteOrder, value packet.Value, data io.Writer) error {
    enum, ok := parser.descriptors[descriptor.Name]
    if !ok {
        return fmt.Errorf("enum descriptor for %s not found", descriptor.Name)
    }

    var index uint8
    for i, v := range enum {
        if v == string(value.(packet.Enum)) {
            index = (uint8)(i)
            break
```

```
        }
    }

    return binary.Write(data, order, index)
}
```

EnumParser es una estructura que implementa la interfaz “parser”, esta estructura contiene el campo “descriptors”, un mapa que asigna valores de tipo enumerado para mostrar las opciones posibles. Por ejemplo, si tienes un valor de tipo enumerado llamado ”AxisRotation” con opciones ”XRotation”, ”YRotation” y ”ZRotation”, el mapa se vería así: ”{”AxisRotation”: [”XRotation”, ”YRotation”, ”ZRotation”]}”.

El método “decode” toma como atributos un “ValueDescriptor” con nombre y tipo de dato, un “binary.ByteOrder” que define si está en Big Endian o Little Endian y un Reader para acceder a los datos. Este método decodifica un valor de tipo enumerado a partir de ese Reader. En primer lugar obtiene el valor enumerado asociado el nombre del descriptor, por ejemplo ”AxisRotation”, comprueba que el enumerado se encuentra en el mapa y si no es así devuelve un “Default” y un mensaje de error. Luego se inicializa un byte donde se lee el índice del valor enumerado dentro del mapa, comprueba que ese índice existe y devuelve el valor de enumeración correspondiente, que es el valor decodificado. En el caso de que fuese ”YRotation” para este ejemplo, el valor del índice sería 1, como el tamaño del enumerado es de 3, sí está incluido. Si cualquiera de los pasos anteriores falla devuelve un “Default” junto a un error. El método “encode” realiza el proceso contrario, encuentra el enumerado correspondiente ”AxisRotation”, busca en ese enumerado el string que coincida con el valor que se ha pasado como atributo, ”YRotation” y escribe ese índice para enviar el paquete, en este caso 1.

Unit Converter

Todas las medidas que se reciben del vehículo se obtienen en unas unidades particulares, que no tienen por qué ser las deseadas por el equipo; este módulo se encarga de esta conversión. Algún ejemplo puede ser conversions de radianes a grados centígrados, cambios de unidades al Sistema Internacional, etc. Este módulo también funciona en ambos sentidos, pudiendo modificar las unidades utilizadas por los usuarios para que sean comprensibles para el vehículo.

Para este módulo se utiliza un struct llamado “Operation” que representa una operación aritmética, con un operador y un operando. Cuenta con los métodos “convert” y “revert” para aplicar y deshacer la operación representada. Se define también un slice de “Operation” con el nombre “Operation” para representar una lista de operaciones. Para formar estas estructuras se recibe una cadena de texto que representa operaciones aritméticas y se utilizan dos funciones, “operationExp” y “NewOperations”. A partir de expresiones regulares con Regex se identifican las operaciones y operandos y se construyen las diferentes “Operations”.

```
const decimalRegex = `[-+]?(\d*\.)?\d+(e[-+]?\d+)?`

var operationExp = regexp.MustCompile(fmt.Sprintf(`([+\-\/*]{1})(%s)`,
decimalRegex))

func NewOperations(literal string) Operations {
    matches := operationExp.FindAllStringSubmatch(literal, -1)
    operations := make([]Operation, 0)
    for _, match := range matches {
```

```
        operation := getOperation(match[1], match[2])
        operations = append(operations, operation)
    }
    return operations
}
func getOperation(operator string, operand string) Operation {
    numOperand, err := strconv.ParseFloat(operand, 64)
    if err != nil {
        log.Fatalln("units: operations: getOperation:", err)
    }
    return Operation{
        Operator: operator,
        Operand:  numOperand,
    }
}
```

Además, el módulo tiene con otro archivo "UnitConverter" que cuenta con funciones para realizar conversiones de unidades basadas en operaciones aritméticas. Tiene una función constructora que toma el nombre de tipo de unidad, una lista de placas y un mapa que relaciona nombres de unidades con secuencias de operaciones.

Logger

El módulo Logger registra todos los datos que recibe en archivos de valores separados por coma, CSV. Este módulo identifica los distintos tipos de valores y medidas y registra cada una de ellas en archivos independientes para facilitar su análisis. Estos archivos pueden utilizarse con software como Matlab para representar gráficamente los valores y realizar una evaluación exhaustiva de la información facilitada por el vehículo.

El directorio principal del módulo es "logger_handler", implementa un sistema de registro de logs que permite habilitar o deshabilitar la generación de logs en función de los clientes conectados al WebSocket. Define la estructura "CSVFile" y proporciona funciones para crear un nuevo fichero CSV, escribir datos en el archivo, vaciar el buffer y cerrarlo. Con la estructura "Logger" se proporcionan métodos para la generación de logs, y contiene un manejador de cada "Logger", "LoggerHandler".

```
type CSVFile struct {
    fileMx *sync.Mutex
    file   *os.File
    writer *csv.Writer
}

type Loggable interface {
    Id() string
    Log() []string
}

type Logger interface {
    Ids() common.Set[string]
    Start(basePath string) chan<- Loggable
}

type LoggerHandler struct {
    loggers       map[string]Logger
```

```
    loggableChan   chan Loggable
    currentClient *wsModels.Client
    isRunning      bool
    isRunningMx    *sync.Mutex
    config         Config
    trace          zerolog.Logger
}
```

Este módulo cuenta con numerosos logger diferentes, según la estructura de datos o el archivo que se necesite registrar (file_logger, info, message_logger, order_logger, packet_logger, state_space_logger, value_logger). En cada uno de ellos se define un constructor público, unos métodos de identificación, de registro y de escritura.

Además, un gran número de objetos cuentan con el campo "trace"; este atributo permite dejar un registro de la traza de ejecución que sigue el programa en la línea de comandos y en un fichero, además incluye información más específica, advertencias y errores. Para obtener esta funcionalidad se parte de la librería "github.com/rs/zerolog". El programa principal llama a la función "initTrace" indicando el nivel de detalle que quiere en la traza y el nombre del fichero donde se va a registrar. Se crea un mapa que asigna los diferentes niveles con los reales de la librería, se configura además el formato de los números de línea, nombres, formatos de fechas, de tiempo y de errores. Además, se establece un logger global con la asignación del nivel de detalle elegido, y desde le resto de módulos se incluye la información necesaria para cada uno de los niveles. El resultado es el representado en la Figura 4.3. En la línea de comandos y en un fichero previamente establecido queda registrada la traza de todo lo que ocurre en el programa para facilitar la comprensión por el usuario, y permitir realizar debug de manera mucho más rápida.

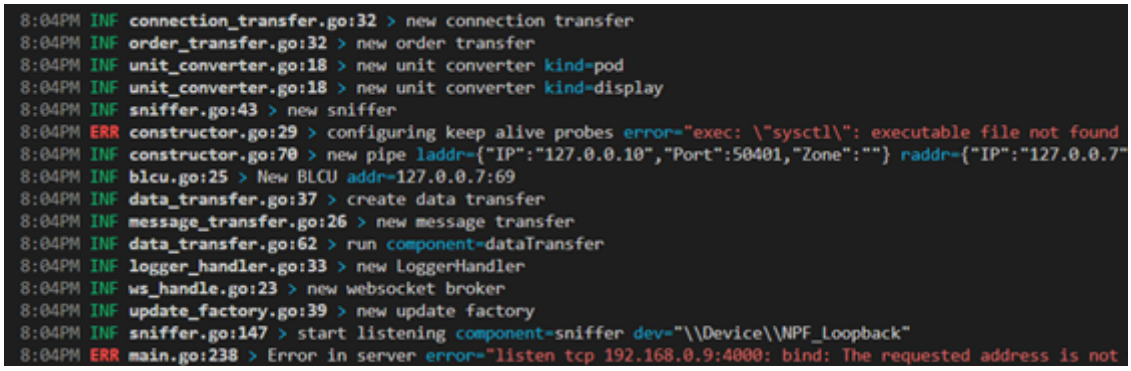

Figura 4.3: Traza por línea de comandos
Fuente: Hyperloop UPV H8

Transfer modules

Estos módulos sirven de puente entre el back-end y el front-end. La gestión de comunicación de cada tipo de mensaje tiene unos requerimientos diferentes por lo que estos módulos engloban varios directorios para cada uno de los tipos de datos:

- **Módulo "data_transfer":** Envía los mensajes de las placas al front-end después de haber sido procesadas por el resto de módulos. En vez de enviar toda la información en tiempo real que podría colapsar la red, almacena todos los paquetes recibidos en un map según su id. Si varios paquetes son recibidos con el mismo id, solo se almacena el último, el más actualizado, el resto no son necesarios; cada cierto tiempo, esta lista es enviada al cliente para que reciba los valores más actualizados de todos

los paquetes. La estructura ”DataTransfer” cuenta con un mutex para asegurar el correcto acceso concurrente, un mapa de “Update", que es una representación de datos de una actualización, con su identificador, conteo de actualizaciones, tiempo de ciclos y valores de medición actualizados. Además cuenta con un “ReplayObservable” utilizado para almacenar y enviar actualizaciones de los datos a los suscriptores interesados, un ticker para enviar las actualizaciones del buffer de forma periódica, una cadena de texto que indica donde se publican las conexiones, y por último un “trace” para registrar los distintos mensajes del proceso.

```
type Update struct {
   Id         uint16                 `json:"id"`
   HexValue   string                 `json:"hexValue"`
   Count      uint64                 `json:"count"`
   CycleTime  uint64                 `json:"cycleTime"`
   Values     map[string]UpdateValue `json:"measurementUpdates"`
}

type DataTransfer struct {
   bufMx            *sync.Mutex
   updateBuf        map[uint16]models.Update
   updateObservable observable.ReplayObservable[map[uint16]models.Update]
   ticker           *time.Ticker
   updateTopic      string
   trace            zerolog.Logger
}
```

Para esta estructura se facilitan métodos para actualizar los datos, enviar los mensajes, etc. Además, se facilita el método “Run” que ejecuta un bucle para enviar periódicamente (según intervalos definidos por el atributo “ticker”) las actualizaciones almacenadas en el buffer a los suscriptores. Este método se ejecuta en segundo plano a través de una “goroutine”.

```
func (dataTransfer *DataTransfer) Run() {
    dataTransfer.trace.Info().Msg("run")
    for {
        <-dataTransfer.ticker.C
        dataTransfer.trySend()
    }
}

func (dataTransfer *DataTransfer) trySend() {
    dataTransfer.bufMx.Lock()
    defer dataTransfer.bufMx.Unlock()

    if len(dataTransfer.updateBuf) == 0 {
        return
    }

    dataTransfer.sendBuf()
}

func (dataTransfer *DataTransfer) sendBuf() {
    dataTransfer.trace.Trace().Msg("send buffer")
    dataTransfer.updateObservable.Next(dataTransfer.updateBuf)
    dataTransfer.updateBuf = make(map[uint16]models.Update,
```

```
    len(dataTransfer.updateBuf))
}
```

- **Módulo "order_transfer":** Recibe las órdenes de los clientes para que puedan ser dirigidas al vehículo, aprovechando las conexiones TCP establecidas por el Pipe Handler del módulo Transport Controller. La estructura de transferencia de órdenes es muy similar al resto, se diferencia en el atributo "channel" que incluye un canal para enviar órdenes a través de la aplicación y que sean recibidas en otros módulos del back-end para procesarlas. Se facilitan los métodos similares al resto de manejadores de transferencia para gestinoar actualizaciones, envíos, etc., como es el caso de "updateMessage" y "handleSuscription". Adicionalmente, se añaden los métodos de "AddStateOrders" y "RemoveStateOrders" para agregar y eliminar órdenes en el mapa "StateOrders", actualizando la lista de órdenes que están disponibles para cada placa y notificar a los suscriptores de los cambios realizados. "ClearOrders" limpia por completo el mapa para restablecer el estado de las órdenes.

```
type OrderTransfer struct {
   stateOrdersMx         *sync.Mutex
   stateOrders           map[string][]uint16
   stateOrdersObservable observable.ReplayObservable[map[string][]uint16]
   channel               chan<- vehicle_models.Order
   trace                 zerolog.Logger
}
```

- **Módulo "connection_transfer":** Informa del estado de las conexiones TCP entre el back-end y el vehículo, este módulo también utiliza el Pipe Handler para obtener esta información. Proporciona métodos para recibir mensajes, manejar suscripciones y actualizar el estado de las conexiones, notificando a los suscriptores interesados. La estructura es similar al resto de estructuras del módulo, pero cuenta con un mapa de "Connection" con el estado de las conexiones.

```
type Connection struct {
    Name        string `json:"name"`
    IsConnected bool   `json:"isConnected"`
}

type ConnectionTransfer struct {
    writeMx               *sync.Mutex
    boardStatus           map[string]Connection
    boardStatusObservable observable.ReplayObservable[[]Connection]
    updateTopic           string
    trace                 zerolog.Logger
}
```

- **Módulo "message_transfer":** Es el módulo responsable de enviar todos los mensajes de estado generados por el vehículo, informando de errores y advertencias que provocan que el vehículo no funcione correctamente. La estructura del manejador es más sencilla que el resto de manejadores explicados. Proporciona métodos para enviar mensajes a los suscriptores interesados a través de "messageObservable".

```
type MessageTransfer struct {
    updateTopic       string
    messageObservable observable.NoReplayObservable[any]
```

```
    trace              zerolog.Logger
}
```

HTTP Server

Este módulo implementa una infraestructura para manejar servidores web con configuraciones y endpoints diferentes. Proporciona funcionalidades para servir archivos estáticos, JSON y establecer conexiones WebSocket. Con este módulo se gestionan las peticiones HTTP enviadas por el front-end. La estructura se define con varios tipos de datos diferentes:

- **"ServerConfig":** Configuración para un servidor en particular, contiene campos como la dirección, el número máximo de conexiones, los endpoints y la ruta al directorio de archivos estáticos.
- **"EndpointConfig":** Configuración de endpoints para diferentes recursos del servidor.
- **"EndpointData":** Un contenedor para los datos de los diferentes endpoints, de tipo any, que indica que puede contener cualquier tipo de datos.
- **"Config":** Un mapa que asocia el nombre de un servidor con la configuración de tipo "ServerConfig".

```
type Config map[string]ServerConfig

type ServerConfig struct {
    Addr           string
    `toml:"address" default:"localhost:4000"`
    MaxConnections *int32         `toml:"client_limit,omitempty"`
    Endpoints      EndpointConfig `toml:"endpoints"`
    StaticPath     string         `toml:"static" default:"./static"`
}

type EndpointConfig struct {
    PodData           string
    `toml:"pod_data" default:"/podDataStructure"`
    OrderData         string
    `toml:"order_data" default:"/orderStructures"`
    ProgramableBoards string
    `toml:"programable_boards" default:"/uploadableBoards"`
    Connections       string `toml:"connections" default:"/backend"`
    Files             string `toml:"files" default:"/"`
}

type EndpointData struct {
    PodData           any
    OrderData         any
    ProgramableBoards any
}
```

- **"WebServer":** Respresenta un servidor web individual con un router, un manejador de conexiones, un contador atómico para mantener las conexiones activas de manera concurrente y su configuración.

```
type WebServer struct {
    name        string
    router      *mux.Router
    connnHandler ConnectionHandler
    connected   *atomic.Int32
    config      ServerConfig
}
```

Para crear cada uno de los WebServer se ha creado el método "NewWebServer" que inicializa el objeto y contiene funciones como "serveJSON", "serveFiles" y "serveWebSocket" para gestionar los diferentes tipos de solicitudes en los endpoints correspondientes.

```
func NewWebServer(name string, connectionHandle ConnectionHandler,
staticData EndpointData, config ServerConfig) (*WebServer, error) {
    server := &WebServer{
        name:        name,
        router:      mux.NewRouter(),
        connnHandler: connectionHandle,
        connected:   &atomic.Int32{},
        config:      config,
    }

    headers := map[string]string{
        "Access-Control-Allow-Origin": "*",
    }

    err := server.serveJSON("/backend"+config.Endpoints.PodData,
    staticData.PodData, headers)
    if err != nil {
        return nil, err
    }

    err = server.serveJSON("/backend"+config.Endpoints.OrderData,
    staticData.OrderData, headers)
    if err != nil {
        return nil, err
    }

    err = server.serveJSON("/backend"+config.Endpoints.ProgramableBoards,
         staticData.ProgramableBoards, headers)
    if err != nil {
        return nil, err
    }

    upgrader := &websocket.Upgrader{
        CheckOrigin: func(r *http.Request) bool { return true },
    }
    server.serveWebsocket(config.Endpoints.Connections, upgrader,
    headers)
    server.serveFiles(config.Endpoints.Files, config.StaticPath)

    if err != nil {
        return nil, err
```

```
    }

    return server, nil
}
```

La función "serveWebSocket" es un método de la estructura "Webserver" con cierta complejidad que se utiliza para configurar y manejar las conexiones WebSocket del servidor. En ella se define un manejador HTTP utilizando la librería "net/http" y concretamente la función "http.HandleFunc". Dentro del handler se configura las cabeceras de respuesta HTTP utilizando un mapa "headers". Estas cabeceras permiten establecer el encabezado "Access-Control-Allow-Origin" como "*" para permitir el acceso cruzado desde cualquier origen. Después se comprueba que el número máximo de conexiones está configurado y no se ha alcanzado; si ha ocurrido, se rechaza la conexión. Una vez ha comprobado que debe aceptar la conexión, intenta actualizar la solicitud HTTP a una conexión WebSocket con el método "Upgrade". Se configura el controlador del cierre de la conexión WebSocket para informar la salida y se actualice el contador de conexiones. Se llama al método "server.connHandler.Add" para agregar la conexión WebSocket al controlador de conexiones que mantiene el seguimiento de estas. Por último, se configura el router del servidor para manejar las solicitudes WebSocket en el path especificado utilizando el handler. A continuación se muestra el código de la función:

```
func (server *WebServer) serveWebsocket(path string, upgrader
*websocket.Upgrader, headers map[string]string) {
    handler := http.HandlerFunc(func(w http.ResponseWriter,
    r *http.Request) {
        trace.Info().Str("server", server.name).Msg("new
        websocket connection")
        for key, value := range headers {
            w.Header().Set(key, value)
        }

        if server.config.MaxConnections != nil && server.connected.Load()
        >= *server.config.MaxConnections {
            trace.Error().Msg("websocket connection after maximum
            connections reached")
            http.Error(w, "Max connections reached",
            http.StatusTooManyRequests)
            return
        }

        conn, err := upgrader.Upgrade(w, r, w.Header())
        if err != nil {
            return
        }

        conn.SetCloseHandler(func(code int, text string) error {
            trace.Info().Msg("websocket connection closed")
            server.connected.Add(-1)
            return nil
        })

        err = server.connHandler.Add(conn)
        if err != nil {
```

```
            return
        }
        server.connected.Add(1)

    })

    server.router.Handle(path, NoCacheMiddleware(handler))
}
```

Además, cuenta con una función "ListenAndServe" que inicia el servidor y devuelve un canal de errores. Hace uso de una goroutine para no bloquear la ejecución del programa.

```
func (server *WebServer) ListenAndServe() <-chan error {
    errs := make(chan error, 1)

    go func() {
        errs <- http.ListenAndServe(server.config.Addr, server.router)
        close(errs)
    }()

    return errs
}
```

- **"Handler":** Manejador para múltiples servidores web. Contiene un mapa con el nombre de los servidores y su respectiva instancia de "WebServer". Además, cuenta con un mutex para controlar la concurrencia.

```
type Handler struct {
    config   Config
    serverMx *sync.Mutex
    servers  map[string]*WebServer
}
```

Para poner en funcionamiento cada uno de los manejadores que inicializan los servidores se utiliza la función "ListenAndServe", que añade una goroutine para dejar escuchando y recibiendo solicitudes en paralelo para cada uno de los servidores sin bloquear el programa principal. Este método llama a la función "ListenAndServe" de cada uno de los WebServer, esta función ha sido comentada anteriormente. Se devuelve un canal que se utiliza para recibir todos los posibles errores de cada servidor; para ello cuenta con la función "consumeErrors" que recibe los errores del servidor web y los maneja, si ocurren errores se elimina el servidor con el método "RemoveServer".

```
func (handler *Handler) ListenAndServe() <-chan error {
    errs := make(chan error, len(handler.servers))

    for name := range handler.servers {
        go handler.consumeErrors(name,
        handler.servers[name].ListenAndServe(), errs)
    }

    return errs
}
```

```
func (handler *Handler) consumeErrors(name string, serverErrs
<-chan error, errs chan<- error) {
    for err := range serverErrs {
        errs <- err
        handler.RemoveServer(name)
    }
}

func (handler *Handler) RemoveServer(name string) {
    handler.serverMx.Lock()
    defer handler.serverMx.Unlock()

    delete(handler.servers, name)
}
```

Websocket Handler

Los websocket se utilizan para compartir información entre el back-end y el front-end, es un protocolo de comunicación bidireccional full-duplex [8]. Este protocolo permite al back-end enviar continuamente actualizaciones a los clientes sin necesidad de realizar un petición nueva cada vez. Los mensajes enviados están en formato JSON (Javascript Object Notation), un formato que consiste básicamente en un archivo de texto que sigue la sintaxis de objetos de JavaScript. Este módulo contiene definiciones de estructuras y funciones relacionadas con la manipulación de mensajes y la gestión de clientes para el uso de WebSockets en tiempo real.

Solo se establece una conexión WebSocket entre cada dispositivo cliente y el back-end, optimizando el rendimiento. Para diferenciar los diferentes tipos de mensajes, estos tienen una propiedad llamada ”topic”. El back-end es el encargado de identificar la propiedad para determinar la forma del mensaje y su contenido.

Se define la estructura cliente que representa un cliente conectado a una conexión WebSocket, tiene un atributo ”id” generado con el paquete ”github.com/google/uuid”, una conexión WebSocket creada con el paquete ”github.com/gorilla/websocket” y dos mutex para asegurar la concurrencia en la lectura y en las escrituras. Para esta estructura se definen métodos como “Id” para devolver su identificador, “Read”, “Write”, “Ping” que envía un mensaje ping a través de la conexión WebSocket para comprobar que sigue conectado, y “Close”. A partir del método “NewClient” se crea y devuelve una nueva instancia de cliente.

```
type Client struct {
    id string

    conn    *websocket.Conn
    readMx  *sync.Mutex
    writeMx *sync.Mutex
}
```

Se define una interfaz “MessageHandler” con los métodos “UpdateMessage” que gestiona los mensajes recibidos por el servidor, y “HandlerName” que devuelve el nombre del manejador. Este manejador se usa en el fichero principal de este módulo.

```
type MessageHandler interface {
```

```
    UpdateMessage(client Client, message Message)
    HandlerName() string
}
```

Además, la estructura "Message" define un mensaje enviado a través de un WebSocket, contiene un campo "topic" con el tipo de mensaje y un "payload" con el contenido del mensaje en formato "json.RawMessage". Cuenta con dos constructores, "NewMessage" y "NewMessageBuf", la única diferencia es que el segundo devuelve el mensaje codificado para un búfer de bytes.

```
type Message struct {
    Topic   string          `json:"topic"`
    Payload json.RawMessage `json:"payload"`
}
```

Una vez definidas estas estructuras, se puede definir "WebSocketBroker", contiene un mapa que almacena los manejadores para los diferentes topics con la interfaz "MessageHandler" mencionada anteriormente, y un mapa que almacena los clientes conectados, con un mutex para cada uno de los dos mapas. Además, cuenta con un trace para dejar registro de la traza y la información.

```
type WebSocketBroker struct {
    handlersMx *sync.Mutex
    handlers   map[string][]models.MessageHandler
    clientsMx  *sync.Mutex
    clients    map[string]models.Client
    trace      zerolog.Logger
}
```

Tras crear una instancia de esta estructura con el método "New", cada vez que se cree una nueva conexión se hace uso del método "Add", que crea un nuevo cliente e inicia dos goroutines para lectura de mensajes ("broker.readMessages") y un mensaje de ping periódico para comprobar el estado de la conexión ("broker.ping"). También cuenta con métodos públicos para eliminar manejadores ("RemoveHandler") y cerrar las conexiones WebSocket ("Close"). Para estos métodos se hace uso de métodos privados a fin de separar el código y permitir una mejor comprensión.

Document Adapter

La mayoría de los módulos necesitan información sobre los paquetes recibidos del vehículo. Por ejemplo, el módulo Packet Parser junto con el Unit Converter necesitan conocer el tipo de paquete y las medidas recibidas para saber cómo realizar la conversión. Toda esta información está almacenada en un documento de Google Sheet en el Google Drive de la organización llamado "Avionics Description Excel", que es descargado y analizado por este módulo.

En este archivo se configura la información de todos los tipos de unidades que pueden llegar del vehículo, números de puerto, identificadores de tablas, paquetes y direcciones IP de las placas, cámaras, back-end, routers, etc. Para ello, el documento cuenta con una primera hoja donde se incluye la información genérica y cualquier tipo de metadato. En la Figura 4.4 se puede observar un fragmento de dos de las tablas que tiene esta primera hoja y que conforma el campo "Info" en la estructura "ADE" del back-end.

[TABLE] addresses	
Board	IP
VCU	192.168.1.3
LCU_MASTER	192.168.1.4
LCU_SLAVE	192.168.1.5
PCU	192.168.1.6
BLCU	192.168.1.7
BMSL	192.168.1.8
OBCCU	192.168.1.9
TCU	192.168.0.14
TBLCU	192.168.0.15
Backend	192.168.0.9

[TABLE] units	
Unit	Conversion
V	
A	
ºC	
m	
mm	/1000
deg	
Hz	
Percentage	
m/s	
m/ss	

Figura 4.4: Fragmento de información genérica de "Avionics Description Excel"
Fuente: Hyperloop UPV H8

A continuación, se incluye una hoja por cada una de las placas del vehículo o infraestructura que necesitan enviar información, describiendo el tipo de paquetes que pueden llegar, con su identificador, las medidas, las variables y las estructuras. Existen tres tipos de tablas para cada placa:

- **"Packet":** Define los paquetes que pueden ser enviados por la placa. Cada paquete es identificado por un ID y puede contener uno o varios valores. La Figura 4.5 muestra los tipos de paquetes que pueden llegar de la placa Vehicle Control Unit (VCU) y cómo identificarlos.

[TABLE] Packets		
ID	Name	Type
209	vcu_hardware_reset	order
210	vcu_set_regulator_pressure	order
211	vcu_regulator_packet	data
212	vcu_reed_packet	data
213	vcu_bottle_temperature_pack	data
214	vcu_pressure_packet	data
215	vcu_environmental_packet	data
216	brake	order
217	unbrake	order
218	disable_emergency_tape	order
219	enable_emergency_tape	order
220	healthcheck_and_load	stateOrder
221	healthcheck_and_unload	stateOrder
222	start_static_lev_demostration	stateOrder
223	start_dynamic_lev_demostrati	stateOrder
224	start_traction_demostration	stateOrder
225	stop_demostration	stateOrder
226	take_off	stateOrder
227	landing	stateOrder
228	CloseContactors	order
229	OpenContactors	order
231	start_crawling	stateOrder
232	states_packet	data
233	traction_data	stateOrder
234	traction_end_data	stateOrder

Figura 4.5: Fragmento de la tabla de paquetes de la VCU
Fuente: Hyperloop UPV H8

- **"Measurement":** Define cada valor que puede enviar la placa, pueden pertenecer a medidas de los sensores de la placa, referencias calculadas por los sitemas de control, etc. Esta tabla también incluye información sobre el tipo de cada variable en el código, las unidades en que se envían, cómo deben ser mostradas, e incluso pueden aparecer rangos de seguridad. Es importante destacar que "podUnits" se refiere a las unidades de los valores enviados por el vehículo y "DisplayUnits" a las unidades en las que se muestran en el front-end. La Figura 4.6 muestra las medidas que pueden incluir cada uno de estos paquetes de la VCU.

[TABLE] Measurements

ID	Name	Type	PodUnits	DisplayUnits	SafeRange	WarningRange
new_reference_pressure	New Reference Pressure	float32	bar	bar	[0,10]	[0,10]
reference_pressure	Reference Pressure	float32	bar	bar	[0,10]	[0,10]
actual_pressure	Actual Pressure	float32	bar	bar	[0,10]	[0,10]
valve_state	Valve State	enum(CLOSED, OPEN)				
reed_state	Reed State	enum(EXTENDED, RETRACTED)				
bottle_temp_1	Bottle Temperature 1	float64	ºC	ºC		
bottle_temp_2	Bottle Temperature 2	float64	ºC	ºC		
high_pressure	High Pressure	float32	bar	bar	[0,300]	[0,300]
low_pressure_1	Low Pressure 1	float32	bar	bar	[0,10]	[0,10]
low_pressure_2	Low Pressure 2	float32	bar	bar	[0,10]	[0,10]
environment_pressure	Environment Pressure	float32	bar	bar	[0, 1.2]	[0, 1.2]
environment_temperature	Environment Temperature	float32	ºC	ºC	[0, 50]	[10, 40]
general_state	General State	enum(Initial, Operational, Fault)				
specific_state	Specific State	enum(Idle, Unload, Load, DynamicLev, StaticLev, Traction)				
load_state	Load Demo State	enum(Idle, Pushing, CloseContactors, Crawling, Braking, OpenContactors)				
unload_state	Unload Demo State	enum(Idle, CloseContactors, Returning, Crawling, Braking, OpenContactors)				
traction_state	Traction Demo State	enum(Idle, ReceivingData, CloseContactors, PointTravel, OpenContactors)				
dynamic_state	Dynamic Demo State	enum(Idle, ReceivingData, CloseContactors, TakeOff, PointTravel, Landing, OpenContactos)				
static_lev_state	Static Lev Demo State	enum(Idle, CloseContactors, LevOff, LevOn, OpenContactors)				

Figura 4.6: Fragmento de la tabla de medidas de la VCU
Fuente: Hyperloop UPV H8

- **"Structure":** Esta tabla define los valores que transporta cada paquete, estableciendo una relación de uno a muchos: un paquete transporta múltiples valores, pero un valor solo puede ser transportado por un paquete. La Figura 4.7 muestra un fragmento de la tabla Structures de la VCU.

[TABLE] Structures

Padding	Padding	Padding	Padding	Padding	Padding	Padding	Padding
vcu_bottle_temperature_packet	vcu_environment	unbrake	disable_emerger	enable_emergen	brake	vcu_pressure_packet	healthcheck_and_load
bottle_temp_1	environment_pressure					high_pressure	
bottle_temp_2	environment_temperature					low_pressure_1	
						low_pressure_2	

Figura 4.7: Fragmento de la tabla de estructura de la VCU
Fuente: Hyperloop UPV H8

Este módulo está diseñado para manejar y procesar los datos almacenados en este archivo, para formar las estructuras con los diferentes tipos de paquetes y poder personalizar la información que recibe el back-end sin necesidad de modificar nada del repositorio de Go.

En este módulo se crea una estructura de datos "ADE" que representa la información y las tablas de la hoja de cálculo. La función principal de este módulo es "CreateADE" que crea la estructura a partir del fichero de Google, extrae la información general y los datos específicos de las diferentes tablas que se encuentran en las hojas del archivo. A continuación, se observa la función principal del módulo y cómo hace referencia a numerosas funciones de estructuras internas para dejar un código fácilmente interpretable:

```
func CreateADE(file *excelize.File) (ADE, error) {
    doc, err := document.CreateDocument(file)
```

```
    if err != nil {
        return ADE{}, err
    }
    adeErrors := common.NewErrorList()

    info, err := getInfo(doc)

    if err != nil {
        adeErrors.Add(err)
    }

    boardSheets := FilterMap(doc.Sheets, func(name string, _ document.Sheet)
    bool {
        return strings.HasPrefix(name, BoardPrefix)
    })

    boards, err := getBoards(boardSheets)

    if err != nil {
        adeErrors.Add(err)
    }

    if len(adeErrors) > 0 {
        return ADE{}, adeErrors
    }

    if err != nil {
        adeErrors.Add(err)
    }

    if len(adeErrors) > 0 {
        return ADE{}, adeErrors
    }

    return ADE{
        Info:   info,
        Boards: boards,
    }, nil
}
```

Para formar esta estructura se hace uso de objetos menores, cada uno de ellos cuenta con una función "get" específica, y a partir de ellas se forma la estructura global:

- **"Document":** Consiste en un array de "Sheets", cada una de estas se corresponde con una hoja del fichero.
- **"Sheet":** Se corresponde con una matriz de strings que representa las celdas de las tablas encontradas en la hoja. Convierte estas matrices en matrices rectangulares rellenando las celdas faltantes con celdas vacías.
- **"Board":** Representa la información de una tabla específica y todos sus componentes.
- **"Packet":** Representa cada uno de los paquetes que se pueden recibir, con el tipo de datos incluido en su interior para interpretar la información.

- **"Measurement":** Representa cada una de las medidas de un paquete.
- **"Structure":** Representa los valores que contiene cada paquete. Dado que estas tablas no tienen tamaño fijo, se realiza previamente un análisis del fichero definiendo el tamaño de cada tabla y los valores que contiene, de esta forma se facilita la creación del el objeto "Board".

El campo "Info" del objeto global incluye la información de las direcciones, las unidades, el puerto, los identificadores de cada tabla y de cada mensaje, y claves del back-end. En el campo "Boards" se incluye la información de cada tabla. Se implementan numerosas funciones para encontrar, leer y convertir tablas del fichero en estructuras útiles.

Adicionalmente, una de las funciones principales del módulo es la descarga del fichero de Google Drive, para ello se hace uso de las librerías "github.com/xuri/excelize/v2" y "google.golang.org/api/drive/v3". Este módulo también cuenta con un archivo de útiles para realizar conversiones en la transformación de la hoja de cálculo.

4.4 Testing y validación

Se han realizado numerosas pruebas unitarias para validar el código de todos los módulos. A continuación, se describen los más significativos para cada uno de ellos:

- **Transport Controller:**
  - Creación de numerosos paquetes con diferentes direcciones origen y destino. Se ha comprobado que solo pasan por el sniffer al back-end aquellos que pertenecen a una comunicación entre dos placas del vehículo, ignorando el resto.
  - Pruebas de desconexiones TCP y su correcto funcionamiento y gestión de la reconexión.
  - Simulación de cambios en los estados de las conexiones TCP y comprobación de su correcta notificación.
- **Packet Parser:**
  - Simulación de la codificación y decodificación de paquetes en Little Endian para comprobar una correcta comunicación.
  - Conversión de los paquetes a las estructuras apropiadas en el módulo. Cada valor y atributo es decodificado al tipo de dato y tamaño de memoria correspondiente (uint8, float32, etc.).
  - Transformación de estructuras en un array de bytes correcto comprensible por el vehículo.
- **Unit Converter:**
  - Envío de valores numéricos y no numéricos y comprobación de la conversión correcta de aquellos que lo requieren, además se comprueba la omisión de los no numéricos para este módulo.
- **Logger:**
  - Cada logger registra adecuadamente su traza en un fichero independiente, utilizando el formato correcto, junto con la fecha y hora y la etiqueta.

- **Transfer modules:**
  - Simulación de la transferencia de datos a los clientes para comprobar una correcta tasa de rendimiento. Es necesario verificar que no se pierde ningún paquete durante la transmisión.
  - Comprobación de los diferentes cambios de estado que puede tener cada conexión y correcta notificación y actualización de todos ellos.
  - Correcto funcionamiento del envío de advertencias y errores a través del Message Transfer.
  - Pruebas de envío de órdenes desde la Control Station y correcta recepción y actuación.
- **Servidor HTTP:**
  - Pruebas de sobrecarga y conexión de multiples clientes.
  - Conexión y desconexión de las solicitudes WebSockets.
  - Correcta recepción de las solicitudes de clientes y envío sin pérdidas de los datos a los clientes correctos.
- **Document Adapter:**
  - Conversión de diferentes ficheros a estructuras a través del módulo y comprobación del correcto filtrado y almacenamiento.

Golang facilita una utilidad para realizar tests automatizados sin necesidad de recurrir a una librería externa, facilitando los paquetes y el binario correspondientes. Esta aproximación para la definición de tests permite probar diferentes casuísticas sin necesidad de tener que elaborar varios tests con un código prácticamente igual. A modo de ejemplo se detalla un código de testing automatizado a partir de la utilidad de Golang en el que se inicia con un paquete de valores del vehículo, se transforma en un array de bytes con las características y etiquetas en las que está preparado el vehículo para trabajar, y posteriormente, se vuelve a transformar ese array en una estructura de datos. De esta forma se comprueba a través de múltiples casos que la codificación y decodificación es correcta. Cabe destacar que esta prueba abarca dos funcionalidades diferentes que se han probado de forma separada antes de incluirlas en este test, por lo que el paso intermedio de la prueba ya ha sido comprobado previamente y funciona correctamente.

```
func TestConversionToVehicleStateArray(t *testing.T) {
    t.Run("struct conversion", func(t *testing.T) {
        cases := []ConversionCaseBytes{
            {buf: []models.VehicleState{
                {XDistance: 2.45, YDistance: 2.45, ZDistance: 2.45,
                Current: 4.3, Duty: 1, Temperature: 10.2, XRotation: 2.45,
                YRotation: 2.45, ZRotation: 2.45},
                {XDistance: 3.21, YDistance: 8.41, ZDistance: 2,
                Current: 7.44, Duty: 81, Temperature: 58.07, XRotation: 0,
                YRotation: 0.3, ZRotation: 1},
                {XDistance: 20.45, YDistance: 3.45, ZDistance: 0.45,
                Current: 4.3, Duty: 1, Temperature: 10.2, XRotation: 1.11,
                YRotation: 2, ZRotation: 0.5}},
                result: []models.VehicleState{
                    {XDistance: 2.45, YDistance: 2.45, ZDistance: 2.45,
```

```
                    Current: 4.3, Duty: 1, Temperature: 10.2,
                    XRotation: 2.45, YRotation: 2.45, ZRotation: 2.45},
                    {XDistance: 3.21, YDistance: 8.41, ZDistance: 2,
                    Current: 7.44, Duty: 81, Temperature: 58.07,
                    XRotation: 0, YRotation: 0.3, ZRotation: 1},
                    {XDistance: 20.45, YDistance: 3.45, ZDistance: 0.45,
                    Current: 4.3, Duty: 1, Temperature: 10.2,
                    XRotation: 1.11, YRotation: 2, ZRotation: 0.5}}},
        }

        for _, testCase := range cases {
            bytes := GetAllBytesFromVehiclesState(testCase.buf)
            got, _ := GetAllVehicleStates(bytes)

            if !reflect.DeepEqual(got, testCase.result) {
                t.Fatalf("Wanted %f, got %f", testCase.result[1].Current,
                got[1].Current)
            }
        }

    })
}
```

CAPÍTULO 5
Front-end

Esta sección se enfoca en las diversas interfaces gráficas de usuario creadas por el subsistema Software para cumplir con los objetivos del equipo. En primer lugar, se establecen los requisitos impuestos tanto por el equipo como por la competición. Luego, se justifica el diseño elegido, incluyendo las herramientas utilizadas en el proceso. A continuación, se detallan los distintos repositorios e interfaces de usuario desarrollados. Por último, se proporciona una breve descripción de los métodos de validación empleados para garantizar el correcto funcionamiento de cada una de las interfaces.

5.1 Introducción

La interfaz gráfica de usuario (GUI) es la encargada de mostrar todos los datos proporcionados por el back-end y de recibir todas las entradas realizadas por el usuario a través del teclado, el ratón, controles táctiles, etc. De esta forma se permite a los usuarios que interactúen con programas muy complejos sin necesidad de conocer el funcionamiento interno del programa, facilitando la interacción persona-computador.

Las interfaces gráficas de usuario suelen estar divididas en dos categorías principalmente [3], por un lado se encuentran las interfaces de escritorio, y por el otro las páginas web. Para elegir nuestro front-end, atendemos a las ventajas que proporcionan cada una de estas categorías.

Las interfaces de usuario nativas se construyen sobre los elementos gráficos del sistema operativo, suelen formar parte de aplicaciones de alto rendimiento debido a que utilizan APIs expuestas por el sistema operativo para favorecer esta característica. Están diseñadas para una plataforma o tipo de dispositivo específico, y el usuario debe instalar la versión de software adecuada para su modelo. Permiten actuar con todas las capacidades del dispositivo (GPS, cámara, agenda, etc.). El desarrollo de estas aplicaciones suele ser más costoso. Además, si se desea cubrir varias plataformas, se deberá generar una aplicación para cada una, por ello, lleva mayores costos de actualización y distribución de nuevas versiones. Por otra parte, si quieren distribuirse, la opción más habitual es la de transmitirlas a App stores, que llevan un proceso de auditoría y aprobación.

Las aplicaciones web, en cambio, son diseñadas para ejecutarse en un navegador, se desarrollan utilizando la misma tecnología que la empleada para crear páginas web: HTML, CSS y Javascript. HTML (HyperText Markup Language) es un lenguaje diseñado para definir la estructura de una página web. CSS (Cascading Style Sheets) se utiliza para aplicar estilos y formatos a los documentos HTML. Javascript es un lenguaje de programación que permite que la página web sea interactiva, de esta manera de pueden procesar datos, obtener información de servidores y modificar la página en tiempo de

ejecución. No necesitan instalación previa, ni la aprobación de fabricantes para que sea distribuidas. Todas las actualizaciones son transmitidas en el momento porque se realizan sobre el servidor, no sobre los dispositivos. Su ejecución es más lenta que las nativas y la conectividad provoca malos rendimientos, tampoco pueden utilizar todos los elementos con los que cuenta el dispositivo. Sin embargo, sí que permiten la independencia de plataforma y no necesitan adecuarse a ningún entorno operativo, simplemente necesitan el uso del navegador y su compatibilidad con este.

Una última opción es la de una aplicación híbrida que obtenga beneficios de ambos tipos: permite la utilización de tecnologías multiplataforma como Javascript, HTML y CSS, pero con acceso a funcionalidades del dispositivo. Son desarrolladas con tecnología web y son ejecutadas en un contenedor web.

Las tecnologías web han ido incrementando su popularidad gracias a su flexibilidad y accesibilidad, y su uso se ha ampliado más allá de las páginas web. Existen frameworks, como es el caso de Electron, que permiten construir aplicaciones de escritorio utilizando estas tecnologías. Es decir, se puede crear una aplicación nativa completa a partir de HTML, CSS y Javascript. Los requerimientos descritos a continuación establecen cuál de las herramientas es la más apropiada para las aplicaciones del equipo.

5.2 Requerimientos de diseño

El front-end de la Control Station debe ser eficaz a fin de solucionar cualquier problema que presente el vehículo o actuar ante cualquier emergencia. Los requerimientos más desatacados son los siguientes:

- Se han de tomar las decisiones de manera inmediata y recibir la información en tiempo real. La European Hyperloop Week establece una latencia máxima de 100 milisegundos y, además, la tasa de refresco de la aplicación debe ser como mínimo de 2 Hz. Si alguna de estas acciones lleva un mínimo retraso, el equipo puede no tener tiempo para reaccionar.
- El front-end debe estar capacitado para enviar órdenes al vehículo con una latencia máxima de 100 milisegundos. Aunque es un requisito necesario que todas las medidas de emergencia sean gestionadas por el vehículo y la infraestructura de manera autónoma e independiente, existen situaciones que requieren de la actuación de los usuarios, como por ejemplo el inicio de una demostración.
- Toda la información del vehículo y la infraestructura debe ser visible durante todo el tiempo. Por esta razón se debe evitar disponer de numerosas pestañas con información separada o donde sea necesario desplazarse continuamente sobre la interfaz. Cada pestaña debe tener toda la información necesaria para las acciones que puedan ser necesarias en cada estado del vehículo. Al fin y al cabo, toda la información necesaria que no está visible y necesita de un "scroll" o de un cambio de pestaña, se traduce en un tiempo mayor de reacción en caso de emergencia. Concretamente, es obligatorio que aparezca la velocidad, posición, corriente y voltaje de las baterías, air gaps, y una serie de datos básicos de cada placa del vehículo.
- Debe estar incluida una sección de órdenes donde el freno de emergencia esté visible en todo momento.
- Es obligatorio implementar un frenado auxiliar accesible desde la interfaz.
- El vehículo no puede encontrarse en ningún momento en un estado en el que se frene y se active la propulsión al mismo tiempo.

- Es necesario elaborar rangos de seguridad; si no se cumplen, el vehículo debe entrar en estado de emergencia.
- Debe ser posible la visualización de gráficas y exportación de datos para realizar el análisis de la evolución durante la demostración.
- Debe estar adaptado al funcionamiento del back-end descrito en las secciones anteriores.
- El diseño debe seguir las pautas establecidas para interfaces persona-computador, todos los errores y componentes importantes deben estar destacados por color y posición.
- Si durante el funcionamiento del vehículo se produce algún error, este entra en estado "fault", deteniendo su funcionamiento, y enviando el detalle de estos errores. El front-end es el encargado de mostrar estos mensajes de manera visual y concisa.
- Por último, el front-end necesita adaptarse al tamaño y resolución de los distintos dispositivos donde se utiliza, para que pueda ser visto por el equipo en la Control Station, o por el público en los móviles.

Adicionalmente, es necesario para la etapa de testing y validación del vehículo desarrollar un front-end centrado en la lectura detallada de todos los datos del vehículo, con un historial de cada demostración, envío de órdenes personalizadas y de pruebas con las infraestructuras de testing del equipo. Por ello se requiere hacer un front-end adicional para la etapa de pruebas. Por otra parte, se plantea la necesidad de crear otra aplicación que sirva para aplicar perturbaciones al vehículo y ver en tiempo real cómo actúa y se estabiliza. Estas funcionalidades exceden las necesidades del programa principal y pueden desarrollarse de manera independiente.

5.3 Justificación del diseño

Una vez definidos los requerimientos se ha decidido utilizar el desarrollo de aplicaciones web. El desarrollo web cumple con las necesidades de estas aplicaciones y no hace falta utilizar ninguna capcidad en las que las aplicaciones nativas sean más efectivas. Además, las ventajas que ofrece el desarrollo web en cuanto a la accesibilidad, la capacidad de descarga para el público y su creciente popularidad, hacen que sean las más interesantes para el proyecto.

Los requerimientos descritos anteriormente hacen referencia a tres aplicaciones diferenciables. En primer lugar, una aplicación de testing centrada en la exposición de toda la información del vehículo y la ejecución de órdenes sobre los sistemas para asegurar una correcta actuación. Esta aplicación es privada para el equipo y no necesita de una infraestructura de red para ser mostrada a la audiencia. Esta aplicación se llama "Ethernet-view".

En segundo lugar, una aplicación que muestre un renderizado del vehículo y represente en tiempo real cómo actúa Kenos al recibir perturbaciones en el campo, con el objetivo de perfeccionar cómo interactúa el control para estabilizarse. Esta aplicación se centra en la posición en cada uno de los ejes, el airgap que tiene el vehículo con los raíles y la rotación en cada una de las dimensiones, donde se observa si su posición es adecuada o no está estabilizado. Esta aplicación tiene el nombre de "HIL-GUI".

Por último, la aplicación principal que se ejecuta en la Control Station, que se llama "Control-front", es más completa que las anteriores y se centra en el control del vehículo para las demostraciones y la transmisión de los datos y cámaras al público. Cuenta

con un apartado para la monitorización del vehículo y todos los sistemas y otro para la retransmisión de las cámaras durante las demostraciones. Esta aplicación ha de ser compartida con todo el público, por lo tanto, se desarrolla un front-end replicado en el que se omite la parte de control del vehículo e infraestructura, quedando a modo de expositor para que nadie externo pueda comunicarse con el vehículo. Concretamente, se eliminan las órdenes, los controles de emergencia y avisos. Este segundo front-end está adaptado para dispositivos móviles y toma el nombre de "Mobile-front".

Dado el número de interfaces de usuario que se implementan, se ha creado un repositorio llamado "common" que almacena los componentes comunes para poder reutilizar código.

5.4 Herramientas utilizadas

Con el objetivo de cumplir los requerimientos establecidos, los diferentes front-end se han diseñado buscando la simplicidad y eficacia. Por ello se ha elegido React (véase la Figura 5.1), una librería de JavaScript para elaborar interfaces de usuario interactivas y reutilizables. React se caracteriza por ser realmente rápida y contar con un ecosistema muy amplio de librerías para aumentar sus capacidades. Está centrado en la construcción de una interfaz de forma declarativa, es decir, se define cómo debe ser la interfaz con diferentes estados y React se encarga de actualizar automáticamente la interfaz cuando los datos cambian [6]. Las principales ventajas por las que se ha elegido esta librería son:

- Se basa en componentes, por lo que la reutilización de código se hace mucho más intuitiva y facilita el mantenimiento.
- Utiliza un virtual DOM que mejora el rendimiento.
- Puede ser fácilmente combinada con otras bibliotecas y herramientas según las necesidades del programa.
- Tiene una curva de aprendizaje suave, por lo que los nuevos miembros pueden incorporarse en un tiempo breve al desarrollo de las diferentes aplicaciones del subsistema.
- Facilita la sintaxis JSX que permite combinar la lógica y la presentación en un mismo sitio, de esta forma, el código puede ser más legible.

Figura 5.1: React
Fuente: es.react.dev

Esta librería ha sido elegida en vez de Angular debido a que este último es un framework que tiene una curva de aprendizaje algo más lenta, y pese a ser muy completo y proporcionar soluciones integradas, no es necesario optar por la estructura sólida que

ofrece. Este framework fue utilizado durante la edición anterior; sin embargo, la incorporación de diferentes miembros a lo largo de la edición era lenta y no se aprovechaban especialmente las ventajas que ofrece, por esa razón, se decidió migrar a React.

Además, se ha decidido utilizar TypeScript (Véase la Figura 5.2) en vez de JavaScript. TypeScript es una variación de JavaScript que incluye tipado estático y algunas mejoras de sintaxis del lenguaje [1]. Las principales ventajas que ofrece este lenguaje y por lo que ha sido elegido son las siguientes:

- **Tipado estático:** Se puede definir tipos de datos para variables, parámetros y propiedades de objetos. Esto permite detectar errores en tiempo de compilación y mejorar la robustez del código. En JavaScript estos errores se suelen identificar en tiempo de ejecución.
- **Mantenimiento y refactorización más sencillo:** El tipado estático hace que el código sea más fácil de entender y refactorizar. Los editores de código pueden ofrecer sugerencias basados en los tipos, acelerando el desarrollo de código.
- **Colaboración en equipos más sencilla:** Es más claro entender los tipos de datos, lo que facilita la colaboración en equipos y la comprensión del código.
- **Seguridad en el código:** Los errores y advertencias de tipo permiten evitar problemas comunes y errores en el código.
- **Buena integración con frameworks y bibliotecas:** React tiene soporte directo con TypeScript, por lo que facilita la integración y desarrollo de esta combinación.

Figura 5.2: TypeScript
Fuente: typescriptlang.org

Como sistema de gestión de paquetes se ha utilizado Node Package Manager (npm, véase la FIgura 5.3). Permite compartir, descargar, instalar y administrar librerías y paquetes. Gestiona además las dependencias. Estos paquetes están ubicados en el Registro de Paquetes de npm, una base de datos pública donde los desarrolladores pueden compartir sus contribuciones con la comunidad.

Figura 5.3: Node Package Manager
Fuente: npmjs.com

A pesar de que "Create React App" es la herramienta más habitual que proporciona un entorno de desarrollo preconfigurado y optimizado para aplicaciones React, se ha decidido hacer uso de Vite (véase la Figura 5.4). Vite es un marco de desarrollo web popular centrado en la velocidad, ofrece un tiempo de inicio corto y una recarga rápida, aprovecha módulos nativos de ECMAScript, por lo que se pueden importar módulos en el navegador sin necesidad de transpilación ni empaquetado[1]. Además, cuenta con una

Figura 5.4: Vite
Fuente: vitejs.dev

arquitectura eficiente de plugins y admite TypeScript. Es más ligero que otros empaquetadores, que resulta en proyectos más limpios y con menos sobrecargas.

Para aplicar estilos se utilizan módulos SCSS, una convención utilizada en muchos frameworks. Se refiere a archivos de hojas de estilo CSS utilizadas en módulos, permiten la utilización de estilos locales para cada componente individual. La principal ventaja frente a archivos CSS normales es el aislamiento y la modularidad de los estilos. Cuando se utilizan estilos en módulos, los estilos definidos sobre un componente no afectan a otros componentes, evitando problemas de colisión de nombres de clase, además, se generan nombres de clase únicos para cada componente, permitiendo reutilizar nombres de clase sin necesidad de preocuparse por los conflictos. De esta forma se evitan los estilos globales y se consigue una mejor legibilidad y mantenibilidad.

Es necesario seguir un estándar para gestionar la configuración y las variables de entorno de una aplicación. Inicialmente se hizo uso de ".env", que es un archivo de configuración que se utiliza para definir estas variables de entorno y configurar el comportamiento de una aplicación. En ella se incluye información sensible que no deba estar incluida en el código fuente, por ejemplo, contraseñas, direcciones IP, etc. Permite el funcionamiento de la aplicación en diferentes entornos, cambiando las variables de este fichero. Sin embargo, a mitad de año se decidió cambiar este archivo por "config.toml" que permitía dar una estructura a estas variables y utilizarlas como propiedades de un objeto. De esta manera, el código se hace más legible e intuitivo y está más organizado.

Una de las cuestiones a definir a la hora de crear un proyecto de React es la gestión del estado global de la aplicación, es decir, datos, variables, etc. que son compartidos por varios componentes en diferentes partes de la aplicación. De esta forma no es necesario guardar el estado en los componentes e ir pasando los datos entre ellos. Para ello, se ha utilizado Redux, desarrollada originalmente para React, aunque se puede utilizar en otros frameworks. Está basada en el patrón Flux y se centra en manejar el estado de la aplicación de una manera predecible y centralizada [9]. El "Store" es un objeto que contiene el estado global de la aplicación, es inmutable, no se modifica directamente, si no que se crea un nuevo estado cada vez que se realiza una acción. Las "acciones" son objetos que describen un cambio en el estado. Por último, los "reducers" son funciones que toman el estado actual y una acción, y devuelven un nuevo estado.

En cuanto a la representación del vehículo, se ha utilizado la biblioteca de JavaScript llamada Three.js que permite crear y renderizar gráficos 3D en navegadores web. Esta biblioteca simplifica en gran medida la creación y manipulación de los objetos tridimensio-

[1] Vite (s.f.). Recuperado el 8 de julio de 2023, de https://es.vitejs.dev/guide/

nales y su representación. Para la animación de estos objetos se hace uso de la biblioteca "react-spring".

El editor de código empleado ha sido Visual Studio Code, desarrollado por Microsoft (véase la Figura 5.5). Es una herramienta muy utilizada por los desarrolladores para escribir y editar código en una gran cantidad de lenguajes de programación. VSCode cuenta con gran variedad de extensiones para agregar funcionalidades adicionales, como formateo de código, depuración, soporte a lenguajes específicos, etc. Proporciona resaltado de sintaxis y autocompletado inteligente para agilizar el desarrollo de código. Además, incluye un sistema de depuración, un sistema de control de versiones y una terminal integrada. Ofrece un rendimiento muy elevado y es altamente personalizable. Su facilidad de manejo lo convierte en uno de los editores de código más utilizados.

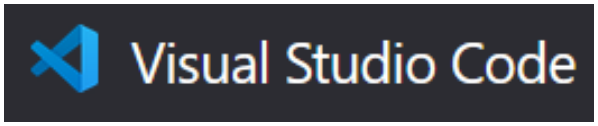

Figura 5.5: Visual Studio Code
Fuente: code.visualstudio.com

Para elaborar el diseño de las diferentes interfaces de usuario que realiza el subsistema Software se ha hecho uso de Figma (véase la Figura 5.6). Figma es una herramienta de diseño de interfaces de usuario que permite crear prototipos de aplicaciones web y móviles, entre muchas otras cosas [19]. Es popular en la industria de diseño y es ampliamente utilizado por los diseñadores. Esta herramienta permite que varias personas puedan colaborar en un mismo archivo de manera simultánea, ofreciendo una amplia gama de herramientas de diseño, dibujo, vectores, texto, estilos, etc. Además, facilita la realización de un prototipado interactivo y la simulación del funcionamiento de la interfaz. Cuenta con un gran número de bibliotecas y plugins y está basado en la nube, por lo que no necesita la instalación de ninguna aplicación. Adicionalmente, la curva de aprendizaje que tiene esta herramienta es suave y permite un dominio sumamente rápido.

Figura 5.6: Figma
Fuente: figma.com

5.5 Repositorio "common-front"

El repositorio "common-front"[2] incluye el código que es utilizado en varias de las interfaces de usuario que ha desarrollado Software, de esta manera, se puede reutilizar el código y tenerlo localizado en un sitio común, agilizando el proceso de desarrollo de las interfaces. En ella se incluyen desde estilos y fuentes de texto, hasta componentes completos, hooks, adaptadores o slices.

El método de funcionamiento es muy sencillo: cada directorio del repositorio cuenta con un fichero index.ts que exporta los diferentes ficheros del directorio y cada fichero

[2]El enlace del repositorio de Github "common-front" es https://github.com/HyperloopUPV-H8/common-front

exporta a su vez las funciones de interés. En el path global del repositorio se cuenta con otro archivo index.ts que exporta todos los subdirectorios. A modo de ejemplo se incluye el contenido de este fichero:

```
export * from "./adapters";
export * from "./broker";
export * from "./wsHandler";
export * from "./models";
export * from "./slices";
export * from "./BackendTypes";
export * from "./components";
export * from "./hooks";
export * from "./services";
export * from "./config/config";
export * from "./styles";
export * from "./form";
export * from "./config";
export * from "./Suspense";
export * from "./math";
export * from "./selectors";
```

Se realiza un ”build” del repositorio y se importa directamente en los proyectos en los que sea necesario, indicando en el package.json la ubicación de este ”build”. Una vez realizado este paso, se puede acceder a lo que se necesite del código.

Con el objetivo de que quede más detallada la utilización de este repositorio se añade el código del componente “ToggleInput” incluido en el repositorio “common-front”. En él se implementa un objeto que tiene un cuadro de texto y un botón, que cuando el botón está activo, permite realizar una escritura en el cuadro de texto. Para ello recibe como propiedades un identificador, un parámetro que indica si inicialmente está habilitado, una función que tiene como parámetro un boolean para actualizar la posición del botón, y otra que recibe un número para actualizar el contenido de la caja de texto. La primera función es llamada cuando se cambia el estado del interruptor, la segunda función cuando cambia la entrada de texto.

```
import { InputTag } from "../InputTag/InputTag";
import { ToggleSwitch } from "../ToggleSwitch/ToggleSwitch";
import { useToggle } from "../../hooks/useToggle";
import { DetailedHTMLProps, InputHTMLAttributes, useEffect } from "react";
import style from "./ToggleInput.module.scss";

type Props = {
    id: string;
    onChange: (state: number) => void;
    onToggle: (state: boolean) => void;
    disabled: boolean;
} & Omit<
    DetailedHTMLProps<InputHTMLAttributes<HTMLInputElement>, HTMLInputElement>,
    "onChange" | "disabled"
>;

export function ToggleInput({
    onToggle,
    onChange,
    disabled,
```

```
    ...inputProps
}: Props) {
    const [isOn, flip] = useToggle(!disabled);

    useEffect(() => {
        onToggle(isOn);
    }, [isOn]);

    return (
        <div className={style.toggleInputWrapper}>
            <InputTag disabled={!isOn} onChange={onChange} {...inputProps} />
            <ToggleSwitch onToggle={onToggle} isOn={isOn} flip={flip} />
        </div>
    );
}
```

El resultado es el siguiente componente, la Figura 5.7 muestra el componente deshabilitado, la Figura 5.8 muestra el componente habilitado tras pulsar el botón.

Figura 5.7: Componente "ToggleInput" deshabilitado
Fuente: Hyperloop UPV H8

Figura 5.8: Componente "ToggleInput" habilitado
Fuente: Hyperloop UPV H8

Respecto al estilo del componente, cobra especial interés el estilo del botón "ToggleSwitch", que realiza una transición al cambiar de estado simulando un interruptor de palanca, muy habitual para este tipo de funcionalidades. Para conseguirlo, se ha realizado el siguiente archivo CSS:

```
@use "../../styles/colors.scss";

$switch-height: 1rem;
$switch-aspect-ratio: 2;
$switch-padding: 0.2rem;

.toggleSwitchWrapper {
    display: flex;
    flex-direction: row;
    align-items: center;
    padding: $switch-padding;
    flex: 0 0 auto;

    height: $switch-height;
    width: calc($switch-height * $switch-aspect-ratio);

    border-radius: calc($switch-height / 2);
```

```
    transition: padding-left 100ms ease-in, background-color 100ms ease-in;

    div {
        transition: background-color 100ms ease-in;

        border-radius: 50%;
        height: 100%;
        aspect-ratio: 1/1;
    }
}

.on {
    padding-left: calc(
        $switch-height * ($switch-aspect-ratio - 1) + $switch-padding
    );

    background-color: var(--tertiary-90);

    div {
        background-color: var(--tertiary-90);
    }
}

.off {
    padding-left: $switch-padding;

    background-color: var(--neutral-90);

    div {
        background-color: var(--neutral-70);
    }
}
```

Este componente es común para varias funcionalidades diferentes de las distintas interfaces. Por ejemplo, es utilizado en "ControlInput" creando una especie de formulario con tantas entradas como elementos tenga el array formData. Para utilizar este ToggleInput simplemente ha habido que importarlo de "common". El resultado se observa en la Figura 5.9.

```
import style from "./ControlInputs.module.scss";
import { ChangeEnable, ChangeValue, Form } from "../TestAttributes";
import { ToggleInput } from "common";

type Props = {
    form: Form;
    changeEnable: ChangeEnable;
    changeValue: ChangeValue;
};

export const ControlInputs = ({ form, changeEnable, changeValue }: Props) => {
    return (
        <form className={style.inputWrapper}>
            {Object.entries(form.formData).map(([name, attributes]) => {
                return (
```

```
                    <ToggleInput
                        key={name}
                        id={attributes.id}
                        type={attributes.type}
                        disabled={!attributes.enabled}
                        value={attributes.value ? attributes.value : ""}
                        onToggle={(state) => {
                            changeEnable(attributes.id, state);
                        }}
                        onChange={(state) => {
                            changeValue(attributes.id, state);
                        }}
                    />
                );
            })}
        </form>
    );
};
```

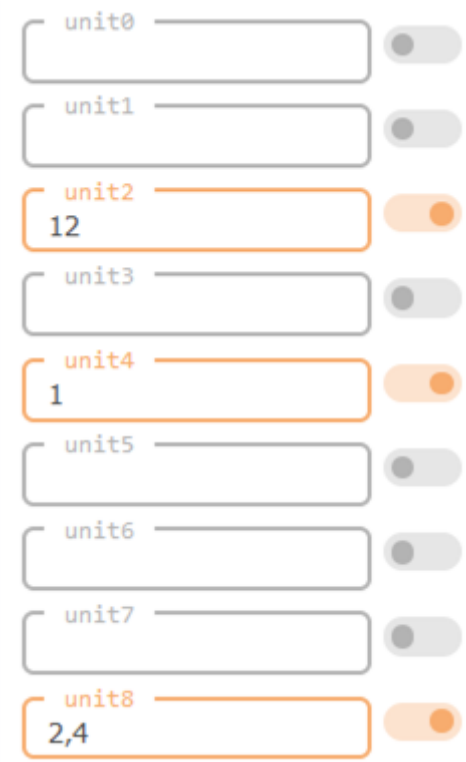

Figura 5.9: Componente "ControlInput"
Fuente: Hyperloop UPV H8

Por otra parte, para agregar los estilos globales incluidos en "common" al resto de repositorios tan solo se debe incluir la siguiente importación en el main.tsx:

```
import "common/dist/style.css";
```

Estos estilos incluyen principalmente una serie de variables sobre el tipo de fuente y sobre la paleta de colores elegida para poder manipularlo de manera general en todos los repositorios a la vez.

5.6 Interfaz de usuario "Ethernet-View"

Esta aplicación cubre la funcionalidad necesaria para la etapa de testing, permitiendo acceder a todos los datos y paquetes que envía el vehículo, estudiar su evolución a partir de gráficas, posibilitar la manipulación de los paquetes, enviar órdenes, recibir mensajes y errores del vehículo, mostrar el estado de las conexiones con las placas, o incluso crear ficheros que registren toda la información para su posterior análisis. Esta aplicación incluye toda la información en una misma pestaña, procurando que aparezca la mayor parte de la información de un único vistazo y obligando a que aparezca en pantalla en todo momento los componentes más importantes de la aplicación (véase la Figura 5.10).

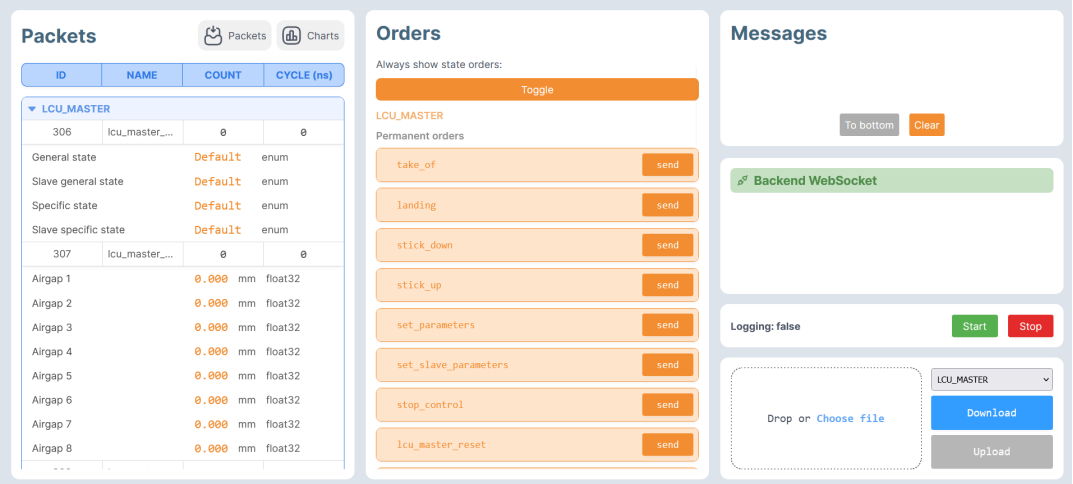

Figura 5.10: "Ethernet-View"
Fuente: Hyperloop UPV

5.6.1. Información general del repositorio "ethernet-front"

El repositorio cuenta con 4 contribuidores formados por el subsistema Software del equipo y un colaborador de la edición anterior que apoyó en alguna duda de la estructura al inicio del año. El repositorio cuenta con 412 commits y 75 ramas. El lenguaje predominante es TypeScript con un 81,5 %, los estilos a través de SCSS ocupan un 18 % y un 0,5 % está clasificado como "otros". Este repositorio fue creado en noviembre de 2022, la versión descrita en este documento puede estar sujeta a algunos cambios. Como todos los repositorios que ha realizado el subsistema en la edición, se mantiene open source [3]. El grueso del trabajo ha sido realizado entre noviembre y diciembre, y en abril, al comenzar la etapa de testing. Sin embargo, ha tenido actualizaciones constantes desde su creación hasta la semana de la competición.

5.6.2. Descripción de las secciones

Esta aplicación está dividida en columnas que se gestionan de manera independiente para distribuirse el espacio y facilitar la manipulación. El siguiente código muestra cómo se realiza la división, en este caso de la tercera columna que incluye los mensajes ("TabLayout"), las conexiones ("Connections"), el logger ("Logger") y el bootloader ("BootloaderContainer"). Contiene componentes adicionales para darle el formato de manera que

[3] El enlace del repositorio de Github "ethernet-front" es https://github.com/HyperloopUPV-H8/ethernet-front

quede homogéneo en todas las columnas, permitiendo cambiar la orientación y envolviendo los componentes. Esta división es aplicable a las tres columnas.

```
import styles from "pages/HomePage/MessagesColumn/MessagesColumn.module.scss";
import { SplitLayout } from "layouts/SplitLayout/SplitLayout";
import { Orientation } from "hooks/useSplit/Orientation";
import { TabLayout } from "layouts/TabLayout/TabLayout";
import { BiLineChart } from "react-icons/bi";
import { nanoid } from "nanoid";
import { MessagesContainer } from "components/MessagesContainer/
    MessagesContainer";
import { Logger } from "components/Logger/Logger";
import { useRef } from "react";
import { Connections } from "common";
import { BootloaderContainer } from "components/BootloaderContainer/
    BootloaderContainer";
import { Island } from "components/Island/Island";

export const MessagesColumn = () => {
    const messagesTabItems = useRef([
        {
            id: nanoid(),
            name: "Messages",
            icon: <BiLineChart />,

            component: <MessagesContainer />,
        },
    ]);

    return (
        <div className={styles.messageColumnWrapper}>
            <SplitLayout
                components={[
                    <TabLayout items={messagesTabItems.current}></TabLayout>,
                    <Island>
                        <Connections />
                    </Island>,
                ]}
                orientation={Orientation.VERTICAL}
            ></SplitLayout>
            <Logger />
            <BootloaderContainer />
        </div>
    );
};
```

Sección "Packets"

Esta sección muestra la información que envían y reciben las placas, con todas las variables de las que disponen, actualizando su valor en tiempo real. Cada placa viene separada y puede ser abierta con un desplegable, de esta manera el diseño es mucho más compacto, además, no es necesario acceder a la información de todas de manera

simultánea. En la Figura 5.11 se observa información que está obteniendo de la PCU. De esta manera se puede identificar cualquier valor anómalo.

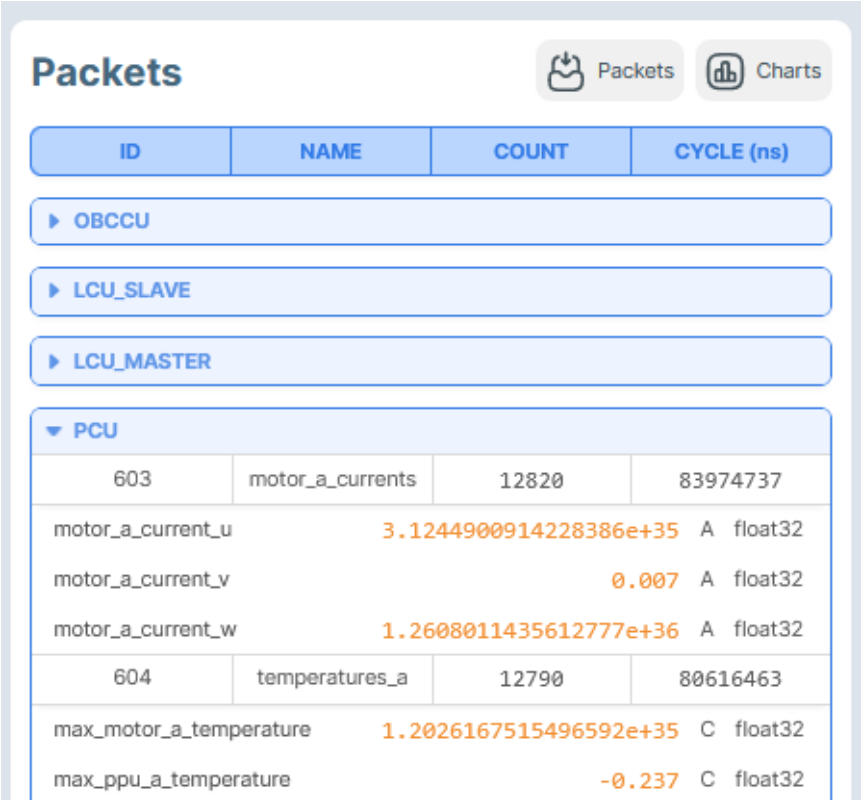

Figura 5.11: Sección "Packets"
Fuente: Hyperloop UPV

Además, con el objetivo de poder ver la evolución de estos datos, la sección cuenta arriba a la derecha con una pestaña para mostrar las gráficas que sean de interés. A través de un cómodo "drag and drop" se puede arrastrar aquellas gráficas que quieran ser estudiadas, como se observa en la Figura 5.12. Además, se permite incluir en un mismo gráfico varios atributos con el objetivo de realizar comparaciones y aprovechar el espacio (véase la Figura 5.13).

Sección "Orders"

La sección de órdenes cuenta con las órdenes permanentes que tienen cada una de las placas. Estas órdenes solo se habilitan cuando tiene sentido que puedan ser utilizadas, además, en el momento que ocurre cualquier error, se activan automáticamente las órdenes que bloquean y apagan el vehículo. Esta sección está pensada para que el usuario pueda manejar de manera manual cualquier instrucción que se necesite enviar al vehículo, pero el vehículo puede trabajar de manera autónoma cumpliendo el recorrido que se le ha indicado previamente sin necesidad de que el usuario esté enviando las instrucciones en el momento de la demostración. A continuación, se muestra en la Figura 5.14 esta sección.

Además, depende del estado en el que se encuentre el vehículo, puede crearse alguna orden temporal para ese estado. Estas órdenes son las "state orders" y se puede forzar su aparición; sin embargo, solo pueden ser utilizadas en momentos específicos. En la Figura 5.15 se muestran órdenes que aparecen según el estado en el que se encuentran.

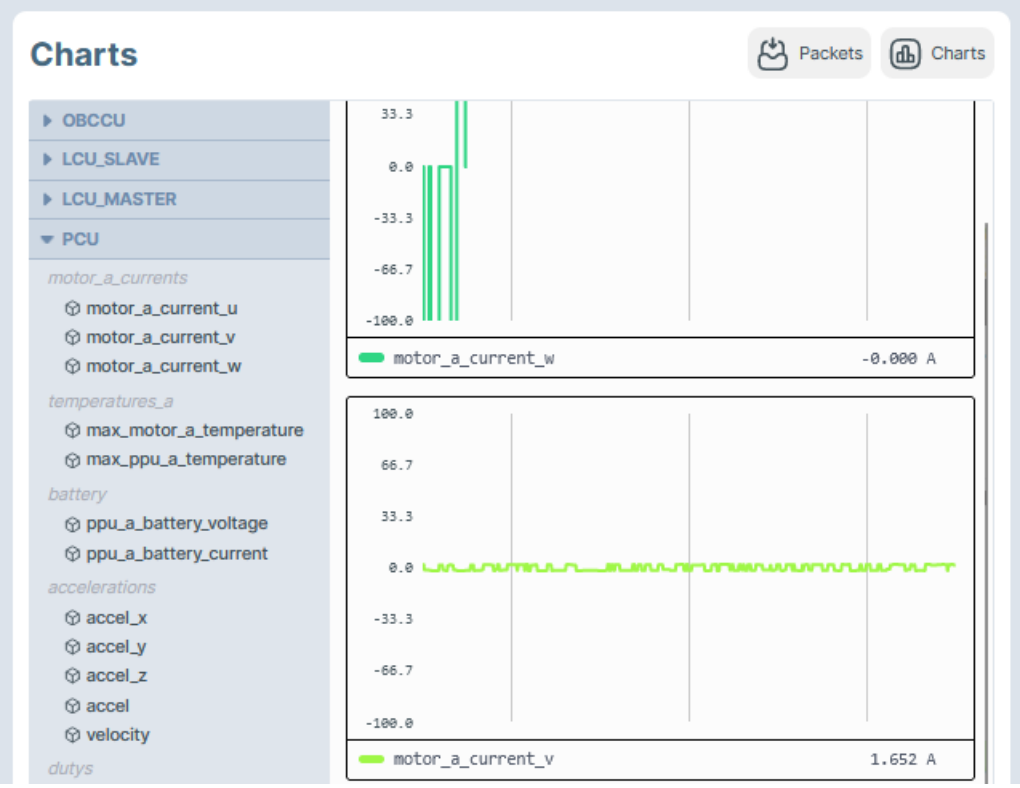

Figura 5.12: Gráficos de la sección "Packets"
Fuente: Hyperloop UPV

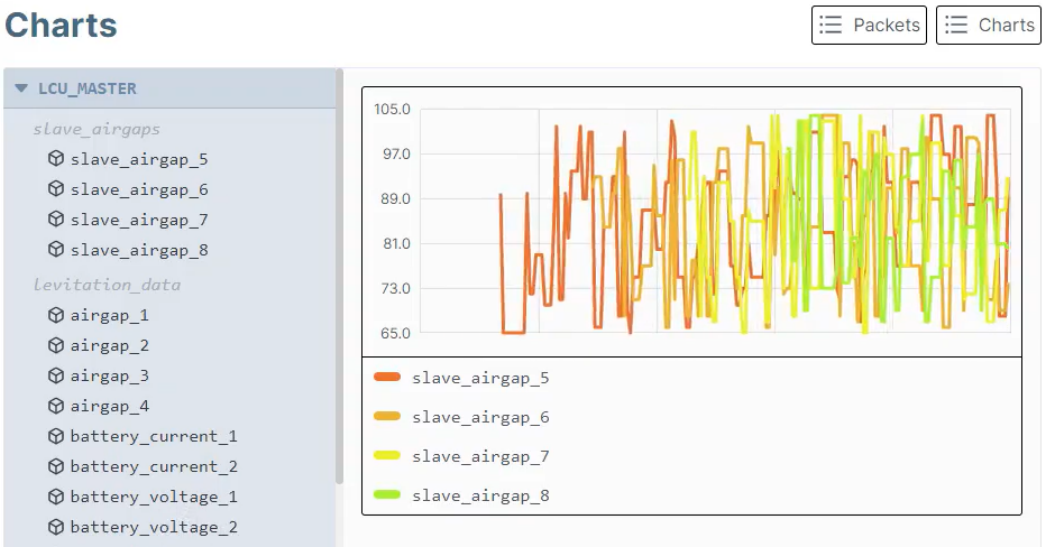

Figura 5.13: Varios atributos en un mismo gráfico
Fuente: Hyperloop UPV

Las listas de órdenes son enviadas por el vehículo y a partir de ellas se genera el formulario con todas las órdenes. La aplicación tiene en cuenta en qué estado se encuentra el vehículo y, por lo tanto, cuáles son las órdenes que se deben mostrar, junto con los diferentes desplegables y las listas de cada una de las placas. Además, esta implementación de las órdenes gestiona su envío y se asegura de que son recibidas.

Para almacenar esta lista de órdenes, se utilizó el ''store" de Redux. A través del método ''dispatch" se envían las acciones (objetos que describen un cambio en el estado) a los ''reducers" (funciones que toman el estado actual y devuelven un nuevo estado).

Figura 5.14: Sección ”Orders”
Fuente: Hyperloop UPV

Figura 5.15: Órdenes temporales según el estado del vehículo
Fuente: Hyperloop UPV

Para obtener los datos del estado debe haberse configurado de manera inicial el ‘’store” y todos los ‘’reducers” que van a gestionar el estado. A continuación, se observa un fragmento donde se exportan actions y reducers para las órdenes.

```
import { orderSlice } from "common";

export const { setOrders, updateStateOrders } = orderSlice.actions;

export default orderSlice.reducer;
```

A través del hook ‘’useSelector” se pueden acceder a los datos almacenados en el store desde un componente funcional. Para ello, se toma una función como argumento que especifica qué parte del estado global se necesita. Cuando los datos seleccionados cambian en el store, el componente se vuelve a renderizar automáticamente.

Con el hook ''useDispatch" se hace referencia a la función"dispatch" del store, permite realizar cambios en el estado global. A continuación, se representa un pequeño fragmento donde se hace uso de ambos hooks:

```
function useOrders$1() {
  const dispatch = useDispatch();
  useSubscribe("order/stateOrders", (msg) => {
    dispatch(orderSlice.actions.updateStateOrders(msg));
  });
  return useSelector((state2) => state2.orders.boards);
}
```

Para la aplicación se han desarrollado dos tipos de órdenes: por un lado, las que solo necesitan ser activadas, por el otro, aquellas que necesitan un parámetro de entrada. En la Figura 5.16 se pueden observar ambos tipos de órdenes: la primera está habilitada y resetea todas las LCUs, mientras que la segunda no se puede enviar, en este caso hasta que se rellenen los parámetros de entrada, porque si no, carece de sentido la orden. Dentro de aquellas órdenes que necesitan de un parámetro de entrada también hay diferencias, porque unas pueden necesitar un número, mientras que otras pueden utilizar un boolean.

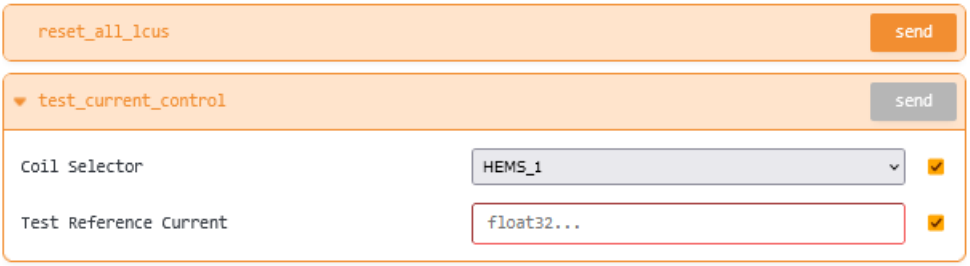

Figura 5.16: Tipos de órdenes
Fuente: Hyperloop UPV

El código de la sección es muy amplio, dada la funcionalidad descrita que se requiere, pero la parte esencial es la creación de cada orden, que se puede observar en este código:

```
return (
        <div className={styles.orderFormWrapper}>
            <Header
                name={description.name}
                info={headerInfo}
                disabled={!form.isValid}
                onTargetClick={listen}
                onButtonClick={trySendOrder}
                springs={springs}
            />
            {isOpen && (
                <Fields
                    fields={form.fields}
                    updateField={updateField}
                    changeEnable={changeEnable}
                />
            )}
        </div>
    );
};
```

Cada orden está formada por dos componentes:

- "Header" se utiliza para crear la barra del encabezado de la orden con contenido interactivo. Puede ser un encabezado desplegable o fijo. Las propiedades que tiene el componente incluye el nombre, un booleano que indica si está deshabilitado, información del tipo de encabezado para implementar o no el desplegable, un objeto "spring" con propiedades de las animaciones que se utilizará en "animated.div", una función "OnTargetClick" para cuando se hace click sobre ella, y otra, "OnButtonClick" para cuando se hace sobre su botón. Se utiliza un icono "Caret" con clases condicionales para el indicador de despliegue, se muestra el nombre del encabezado, un elemento "Target" para cuando se pulsa sobre la orden y un botón para enviar la orden si está habilitado.

```
export type HeaderInfo = ToggableHeader | FixedHeader;

type ToggableHeader = {
    type: "toggable";
    isOpen: boolean;
    toggleDropdown: () => void;
};

type FixedHeader = {
    type: "fixed";
};

type Props = {
    name: string;
    disabled: boolean;
    info: HeaderInfo;
    springs: Record<string, SpringValue>;
    onTargetClick: (state: boolean) => void;
    onButtonClick: () => void;
};

export const Header = ({
    name,
    disabled,
    info,
    springs,
    onTargetClick,
    onButtonClick,
}: Props) => {
    const [targetOn, setTargetOn] = useState(false);

    useEffect(() => {
        onTargetClick(targetOn);
    }, [targetOn]);

    return (
        <animated.div
            className={styles.headerWrapper}
```

```
            onClick={info.type == "toggable" ? info.toggleDropdown : ()
                => {}}
            style={{
                ...springs,
                cursor: info.type == "toggable" ? "pointer" : "auto",
            }}
        >
            <Caret
                isOpen={info.type == "toggable" ? info.isOpen : false}
                className={`${styles.caret} ${
                    info.type == "toggable" ? styles.visible : styles.
                        hidden
                }`}
            />
            <div className={styles.name}>{name}</div>
            <Target
                className={`${styles.target} ${
                    targetOn ? styles.targetVisible : ""
                }`}
                onClick={(ev) => {
                    ev.stopPropagation();
                    setTargetOn((prev) => {
                        return !prev;
                    });
                }}
            />
            <div className={styles.sendBtn}>
                <Button
                    label="send"
                    onClick={(ev) => {
                        onButtonClick();
                        ev.stopPropagation();
                    }}
                    disabled={disabled}
                />
            </div>
        </animated.div>
    );
};
```

- "Field" define un componente que se utiliza para renderizar diferentes tipos de campos de formulario. Cuenta con propiedades que definen el nombre, el campo del formulario, la función "onChange" que se llama cuando cambia el valor del campo, y la función "changeEnabled" llamada cuando cambia el estado de habilitación del campo. La función "handleTextInputChange" maneja los cambios numéricos, verifica que el número es válido y llama a "onChange". Dependiendo del tipo de campo ("field.kind") se crea un componente numérico, booleano o de selección. Además, cuenta con un checkBox que habilita o deshabilita el campo. Se cambia el estilo completo del campo si está habilitado o no.

```
type Props = {
    name: string;
    field: FormField;
```

```
    onChange: (newValue: boolean | string | number, isValid: boolean) =>
        void;
    changeEnabled: (isEnabled: boolean) => void;
};

export const Field = ({ name, field, onChange, changeEnabled }: Props) =>
    {
    function handleTextInputChange(
        value: string,
        type: NumericType,
        range: [number | null, number | null]
    ) {
        const isValid = isNumberValid(value, type, range);
        onChange(Number.parseFloat(value), isValid);
    }

    return (
        <div
            className={`${styles.fieldWrapper} ${
                !field.isEnabled ? styles.disabled : ""
            }`}
        >
            <div className={styles.name}>{name}</div>
            {field.kind == "numeric" ? (
                <NumericInput
                    required={field.isEnabled}
                    disabled={!field.isEnabled}
                    isValid={field.isValid}
                    placeholder={`${field.type}...`}
                    defaultValue={!field.isValid ? "" : (field.value as
                        number)}
                    onChange={(value) =>
                        handleTextInputChange(
                            value,
                            field.type,
                            field.safeRange
                        )
                    }
                />
            ) : field.kind == "boolean" ? (
                <CheckBox
                    isRequired={field.isEnabled}
                    disabled={!field.isEnabled}
                    onChange={(value: boolean) => {
                        onChange(value, true);
                    }}
                />
            ) : (
                <Dropdown
                    value={field.value as string}
                    options={field.options}
                    onChange={(newValue) => {
                        onChange(newValue, true);
                    }}
```

```
                    />
                )}

                <CheckBox
                    color="orange"
                    isRequired={true}
                    onChange={changeEnabled}
                    initialValue={field.isEnabled}
                />
            </div>
        );
};
```

Sección "Messages, connections, logger y bootloader"

Esta última sección se corresponde con la última columna de la interfaz y contiene múltiples funcionalidades diferentes:

- "Messages": Este componente está designado para mostrar advertencias y errores que son generados por el vehículo, como se observa en la Figura 5.17. Su funcionamiento es similar a la consola de un navegador web. Por defecto, esta especie de "consola" siempre se ubica en la parte inferior de la lista de mensajes, es decir, el mensaje más reciente se muestra primero. Sin embargo, al interactuar con un mensaje, como al hacer clic en él para desplegarlo o al desplazar hacia arriba en esta sección, la función de desplazamiento automático hacia abajo se detiene para facilitar la lectura. Al volver a desplazar hacia abajo hasta el final de la lista o al pulsar el botón "To bottom", se reactiva la funcionalidad anterior. Además, se proporciona un botón adicional para borrar por completo la lista de mensajes con el objetivo de no saturar la interfaz.

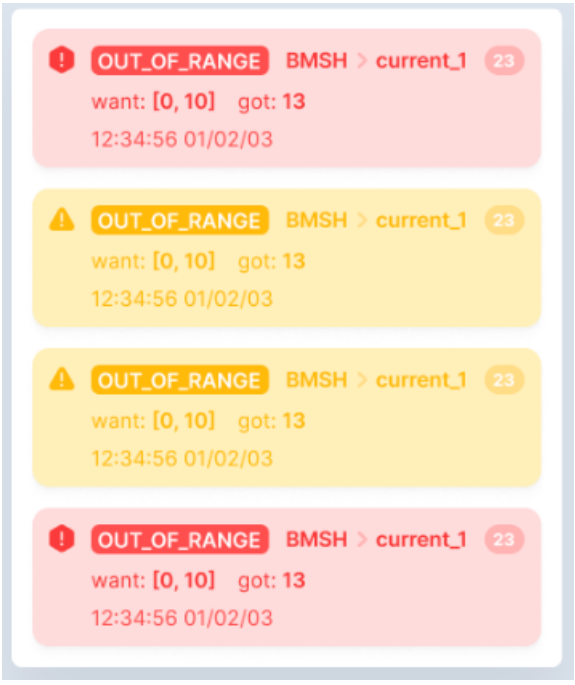

Figura 5.17: Componente de errores y advertencias
Fuente: Hyperloop UPV

- ”Connections”: Esta sección muestra todas las conexiones posibles con las que puede estar conectada la aplicación, tanto de las placas del vehículo como del back-end (véase la Figura 5.18). Identifica en verde aquellas que están activas y muestra en gris aquellas que no están disponibles. Este componente tan solo muestra información y no permite ningún tipo de interacción por parte del usuario. Permite identificar qué le ha ocurrido al vehículo en caso de que algo no haya funcionado como se esperaba.

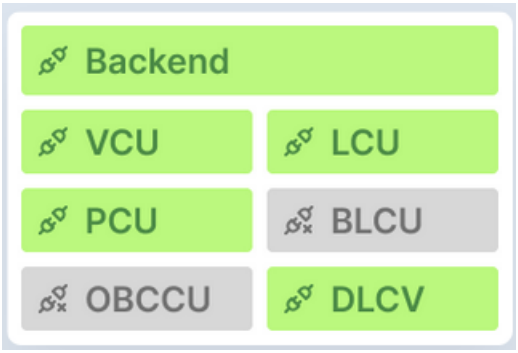

Figura 5.18: Componente de conexiones
Fuente: Hyperloop UPV

- ”Logger”: La interfaz cuenta con una opción de grabación, lo cual permite grabar en un fichero el desarrollo del test para su posterior evaluación. Tan solo es necesario hacer uso del botón ”Start” para que comience a grabar en la dirección predeterminada programada, y con el botón ”Stop” se termina. El componente cuenta con un texto que indica si la acción de escritura está activa (en true) o no. En la Figura 5.19 se observa el componente.

Figura 5.19: Componente para registrar las demostraciones
Fuente: Hyperloop UPV

- ”Bootloader”: Por último se habilita un componente que permite descargar y subir código a cada una de las placas conectadas (véase la Figura 5.20), de esta forma se puede actualizar cada una de ellas sin necesidad de manipularlas físicamente. Esta labor es imprescindible para la seguridad de los miembros de Avionics, puesto que evita que tengan que acercarse al vehículo y tocar cuando está encendido y todas las baterías están conectadas.

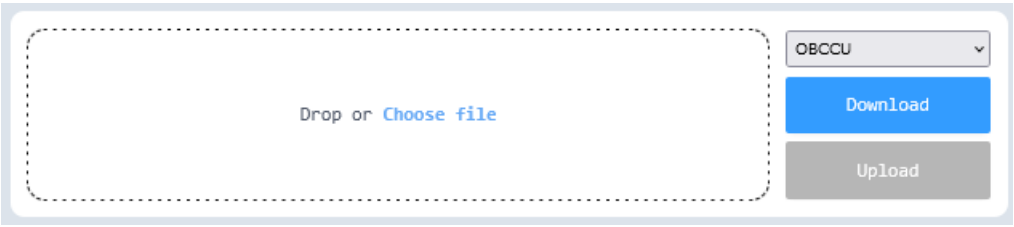

Figura 5.20: Componente para utilizar el ”Bootloader”
Fuente: Hyperloop UPV

5.7 Interfaz de usuario "HIL-GUI"

La placa "Software and Hardware Unit Test Platform" está diseñada para verificar el correcto funcionamiento de los diferentes sistemas desarrollados por Hardware y Firmware a partir de pruebas automatizadas de caja negra. Esta placa soporta todas las entradas y salidas de las placas de control y la totalidad del código que se ejecuta en la placa de desarrollo, de esta forma se puede simular una placa para probar que el firmware desarrollado es correcto, o aplicar un código funcional simulado sobre una placa diseñada por el equipo, para comprobar el funcionamiento del hardware. Para interactuar con esta placa y observar las pruebas de validación, se ha creado esta aplicación. Además, permite realizar las pruebas de perturbaciones sobre el vehículo y comprobar su funcionamiento. Incorpora además modelos 3D para permitir un mejor análisis y comprensión de la información recibida.

5.7.1. Información general del repositorio "hil-gui"

El repositorio cuenta con 2 contribuidores, el autor del Trabajo de Fin de Grado, y el Project Manager del subsistema. El repositorio cuenta con 35 commits y dos ramas. El lenguaje predominante es TypeScript con un 46,3 %, el back-end desarrollado en Golang ocupa un 37,9 %, los estilos a través de SCSS ocupan un 13,1 % y CSS un 1,6 %, un 1,1 % está clasificado como "otros". Este repositorio fue creado en junio de 2023, aunque agrupa los tres repositorios anteriores que se habían creado en mayo, la versión descrita en este documento puede estar sujeta a algunos cambios. Como todos los repositorios que ha realizado el subsistema en la edición, se mantiene open source[4]. El trabajo ha sido realizado entre junio y julio.

Las herramientas utilizadas son las mismas que para el resto de aplicaciones, el HIL-mock y el back-end utilizan golang, mientras que el front-end está realizado en React. La representación 3D está desarrollada con la librería de Javascript Three.js, que permite representar modelos en la web. La animación fluida se utiliza con otra librería de react, llamada "react-spring".

Esta aplicación necesita tres módulos diferentes para su desarrollo, los cuales se detallan a continuación.

5.7.2. HIL-Backend

Este back-end realizado en Golang permite recibir paquetes de bits con la información del vehículo, procesarlo y enviarlo al front-end para que se represente su estado. En sentido contrario, recibe órdenes del front-end, las procesa y las transforma en paquetes de bits para que sean enviados a las placas de testing.

Para ello se utiliza un objeto "Server" que tiene como atributo un manejador de las conexiones que le llegan. Depende de la dirección que recibe, identifica que es el front-end o es el HIL y se le asigna al manejador del hil "hilHandler" correspondiente, cuando ambos están conectados se comienza el método startIDLE().

```
server.SetConnHandler(func(conn *websocket.Conn) {
        remoteHost, _, errSplit := net.SplitHostPort(conn.RemoteAddr().String()
            )
        if errSplit != nil {
            trace.Error().Err(errSplit).Msg("Error spliting IP")
```

[4]El enlace del repositorio de Github "hil-gui" es https://github.com/HyperloopUPV-H8/hil-gui

```
            return
        }

        switch remoteHost {
        case config.Addresses.Frontend:
            hilHandler.SetFrontConn(conn)
            frontMsg := fmt.Sprintf("Frontened connected: %v", conn.RemoteAddr
                ())
            trace.Info().Msg(frontMsg)
        case config.Addresses.Hil:
            hilHandler.SetHilConn(conn)
            hilMsg := fmt.Sprintf("HIL connected: %v", conn.RemoteAddr())
            trace.Info().Msg(hilMsg)
        default:
            trace.Warn().Str("host", conn.RemoteAddr().String()).Msg("
                unrecognized host")
        }

        if hilHandler.AreConnectionsReady() {
            errReady := hilHandler.frontConn.WriteMessage(websocket.TextMessage
                , []byte("Back-end is ready!"))
            if errReady != nil {
                trace.Error().Err(errReady).Msg("Error sending ready message")
                return
            }
            hilHandler.StartIDLE()
            trace.Warn().Msg("Exit IDLE, waiting for connections")
        }
    })
```

El método startIDLE() es un bucle de espera que recibe mensajes, indica que está listo para empezar a trabajar y con un switch identifica si alguno de los mensajes que le llega está solicitando una función del manejador. Por el momento solo con el mensaje de inicio de simulación se activa el inicio del testing, no hay más funcionalidades, pero está preparado para añadir diferentes mensajes que permitan realizar otros métodos.

```
func (hilHandler *HilHandler) StartIDLE() {
    trace.Info().Msg("IDLE")
    ctx, cancel := context.WithCancel(context.Background())
    defer cancel()
    for {
        _, buf, err := hilHandler.frontConn.ReadMessage()
        if err != nil {
            trace.Error().Err(err).Msg("error receiving message in IDLE")
            cancel()
            continue
        }

        msg, err := hilHandler.parseFrontMessage(buf)

        if err != nil {
            trace.Error().Err(err).Msg("parsing message")
            continue
        }
```

```
        switch msg.(type) {
        case StartSimulationFront:
            err := hilHandler.notifyHilStartSimulation()
            if err != nil {
                trace.Error().Err(err).Msg("notifiying HIL to start simulation"
                    )
                cancel()
                break
            }

            err = hilHandler.startSimulationState(ctx)
            if err == nil {
                trace.Info().Msg("IDLE")

            } else if err != nil {
                return
            }
        default:
            trace.Warn().Type("unrecognized msg type", msg)
        }

    }
}
```

El método startSimulationState() crea un conjunto de canales para el intercambio de información entre los diferentes métodos en paralelo, uno de errores, otro para transmitir datos, otro que transmite las órdenes, y por último, uno para parar todos los métodos. Se define un contexto que permite controlar y cancelar las rutinas goroutines que se inicializan seguidamente, estas se encargan de toda la lógica de la transmisión de información con el front-end y con el HIL. A continuación se describe brevemente la funcionalidad de estas goroutines sin entrar en detalles de la implementación dada la envergadura del proyecto:

- **startListeningData():** Escucha datos del HIL sobre el estado del vehículo, los procesa y los sitúa en el canal de datos.
- **startSendingData:** Envía los datos que se han depositado en el canal de datos en formato JSON hacia el front-end.
- **startListeningOrders:** Escucha órdenes del front-end y los deposita en el canal de órdenes.
- **stratSendingOrders:** Envía las órdenes del canal de órdenes hacia el HIL, codificándolo en array de bytes con el formato que entiende.

```
func (hilHandler *HilHandler) startSimulationState(ctx context.Context) error {
    errChan := make(chan error)
    dataChan := make(chan models.VehicleState)
    orderChan := make(chan models.Order)
    stopChan := make(chan struct{})
    trace.Info().Msg("Simulation state")

    simCtx, cancel := context.WithCancel(ctx)
```

```
    go hilHandler.startListeningData(dataChan, errChan, simCtx)
    go hilHandler.startSendingData(dataChan, errChan, simCtx)
    go hilHandler.startListeningOrders(orderChan, errChan, stopChan, simCtx)
    go hilHandler.startSendingOrders(orderChan, errChan, simCtx)

    for {
        select {
        case err := <-errChan:
            cancel()
            return err
        case <-stopChan:
            cancel()
            return nil
        default:
            time.Sleep(time.Millisecond * 100) // To avoid spinning loop
        }
    }
}
```

La información de la ejecución de la aplicación está documentada en el terminal, como se ve en la Figura 5.21.

```
10:09AM INF main.go:70 > Listening in 127.0.0.1:8010/backend
10:10AM INF main.go:52 > HIL connected: 127.0.0.2:51351
10:10AM INF main.go:48 > Frontened connected: 127.0.0.1:51366
10:10AM INF hilHandler.go:66 > IDLE
10:10AM INF main.go:48 > Frontened connected: 127.0.0.1:51367
10:10AM INF hilHandler.go:66 > IDLE
10:10AM ERR hilHandler.go:80 > parsing message error="unrecognized front message"
```

Figura 5.21: Terminal con información sobre el back-end de la HIL-GUI
Fuente: Hyperloop UPV

5.7.3. HIL-Frontend

Es el encargado de representar toda la información para el usuario y ofrece una interfaz que permite enviar órdenes y observar cómo interactua el vehículo (véase la Figura 5.22).

En la esquina superior izquierda aparecen dos botones para iniciar y terminar la simulación. Debajo, se encuentra un panel de control que muestra las tres órdenes principales: Levitar, acelerar y frenar. Además, se facilitan varios botones personalizados para que desde el HIL puedan añadir las perturbaciones oportunas al obtener el identificador de esa orden, como se observa en la Figura 5.23.

Como foco central de la página se encuentra el renderizado 3D de Kenos. Este renderizado cambia su posición y sus rotaciones según las valores de estado que recibe, simulando cómo se vería realmente el vehículo. Permite ampliar, alejar y desplazarse para observarlo desde cualquier ángulo (véase la Figura 5.24). Además, cuenta con la librería "react-spring" para darle realismo y suavidad a las animaciones observadas tras los cambios de valores del vehículo. Para ello se hace uso de la librería "react-spring" que facilita componentes para la representación 3D. El componente recibe la información del estado del vehículo con los atributos que se han de representar. Cuenta con varios componentes:

- Dentro del componente "Canvas" se crea un lienzo donde se renderiza el objeto 3D.

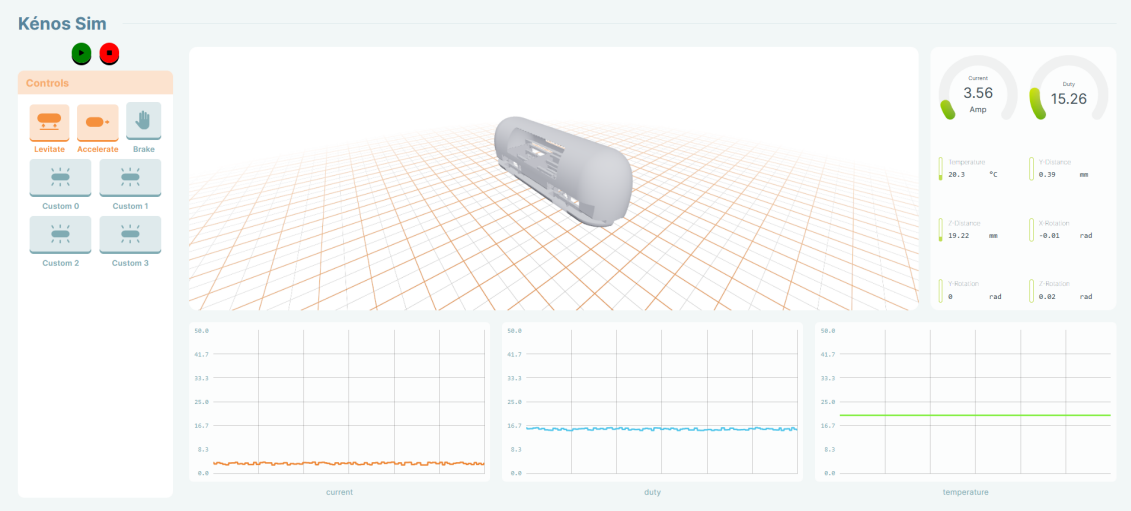

Figura 5.22: Interfaz de usuario "HIL-GUI"
Fuente: Hyperloop UPV

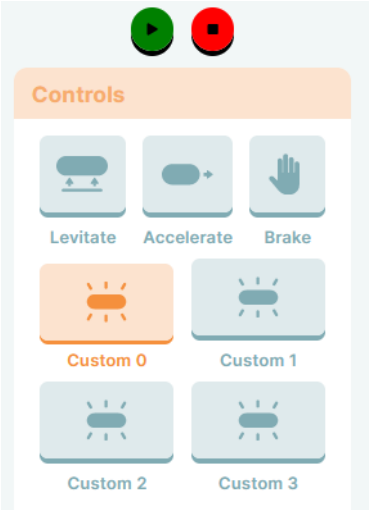

Figura 5.23: Panel de control de la "HIL-GUI"
Fuente: Hyperloop UPV

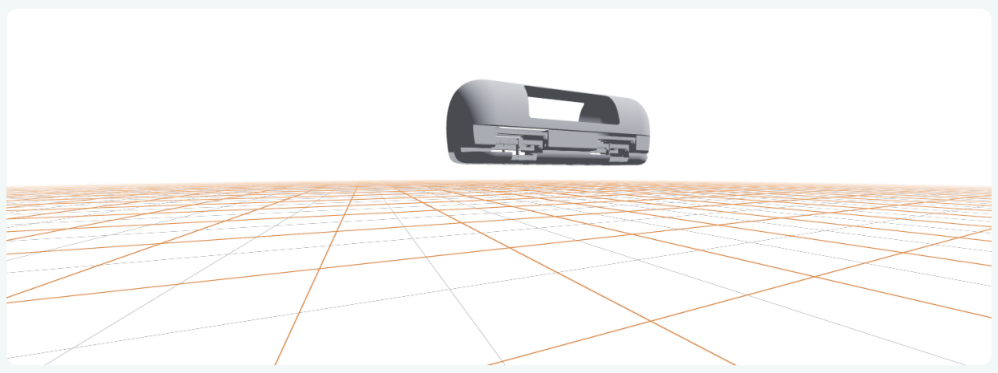

Figura 5.24: Representación 3D de Kenos
Fuente: Hyperloop UPV

- El componente "PerspectiveCamera" define una cámara en perspectiva para visualizar la escena.

- "OrbitControl" proporciona interacción de control orbital para mover y rotar la cámara.
- Componentes de luz para iluminar la escena y agregar sombras.
- "Grid" crea una cuadrícula en el suelo.
- El componente "VehicleRepresentation" representa el renderizado del vehículo y sus animaciones.

```
import { Grid, OrbitControls, PerspectiveCamera } from "@react-three/
    drei";
import { Canvas } from "@react-three/fiber";
import { VehicleRepresentation } from "./VehicleRepresentation/
    VehicleRepresentation";
import { VehicleState } from "home/VehicleSimulation/
    VehicleSimulation";

type Props = {
    info: VehicleState;
};

export function ThreeJsVehicle({ info }: Props) {
    return (
        <Canvas>
            <PerspectiveCamera
                makeDefault
                position={[7, 5, 6]}
                fov={60}
            />
            <OrbitControls />
            <ambientLight intensity={0.1} />
            <directionalLight
                color="white"
                position={[15, 20, 10]}
                intensity={0.8}
            />
            <Grid
                args={[30, 30]}
                sectionColor="#ee7623"
                fadeDistance={20}
                infiniteGrid
            />
            <VehicleRepresentation info={info} />
        </Canvas>
    );
}
```

- "VehicleRepresentation" tiene el renderizado desarrollado por el subsistema Outreach del equipo. A este objeto se le adaptan los atributos que recibe del estado para su correcta representación en la escena, como es el caso de la función "calculatePositionDistance()". Se hace uso de useSpring para las animcaiones del objeto, pasándole una configuración con proporciones físicas del objeto para que reaccione como interesa. Cada vez que se modifica la información, se cambian los valores del useSpring y se aplican al objeto animado "animated.mesh".

```
    import { animated, useSpring } from "@react-spring/three";
import { useGLTF } from "@react-three/drei";
import { useFrame } from "@react-three/fiber";
import { VehicleState } from "models/vehicle";
import { useEffect, useRef } from "react";
import { Mesh } from "three";

const MAX_YDISTANCE = 25;
const MIN_YDISTANCE = 15;
const MAX_CANVAS_X = 3;

type Props = {
    info: VehicleState;
};

function calculatePositionDistance(distance: number) {
    if (distance >= 10) {
        return (
            ((distance - MIN_YDISTANCE) * MAX_CANVAS_X) /
            (MAX_YDISTANCE - MIN_YDISTANCE)
        );
    } else {
        return 0;
    }
}

function calculatePositionYDistance(yDistance: number) {
    return ((yDistance - 0 + 1) * MAX_CANVAS_X) / (5 - 0);
}

export function VehicleRepresentation({ info }: Props) {
    const meshRef = useRef<Mesh>(null!);
    const valueRef = useRef(info);

    const model = useGLTF("./pod_simplified.glb");

    const [springProps, setSpringProps] = useSpring(() => ({
        positionX: calculatePositionDistance(info.xDistance),
        positionY: calculatePositionYDistance(info.yDistance),
        positionZ: calculatePositionDistance(info.zDistance),
        rotationX: info.xRotation,
        rotationY: info.yRotation,
        rotationZ: info.zRotation,
        config: {
            mass: 1,
            tension: 15,
            friction: 10,
            precision: 0.0001,
        },
    }));

    useEffect(() => {
        valueRef.current = info;
```

```
        valueRef.current.xDistance = calculatePositionDistance(info.
            xDistance);
        valueRef.current.yDistance = calculatePositionYDistance(info.
            yDistance);
        valueRef.current.zDistance = calculatePositionDistance(info.
            zDistance);

        setSpringProps({
            positionX: valueRef.current.xDistance,
            positionY: valueRef.current.yDistance,
            positionZ: valueRef.current.zDistance,
            rotationX: valueRef.current.xRotation,
            rotationY: valueRef.current.yRotation,
            rotationZ: valueRef.current.zRotation,
        });
    }, [info]);

    return (
        <animated.mesh
            scale={3}
            ref={meshRef}
            position-x={springProps.positionX}
            position-y={springProps.positionY}
            position-z={springProps.positionZ}
            rotation-x={springProps.rotationX}
            rotation-y={springProps.rotationY}
            rotation-z={springProps.rotationZ}
        >
            <primitive object={model.scene} />
        </animated.mesh>
    );
}
```

A la derecha se encuentra una tabla con los diferentes datos del vehículo. Concretamente, la corriente y el duty vienen representados junto a una curva para obtener a simple vista si se encuentra en el rango de seguridad o se acerca a valores peligrosos. En la Figura 5.25 se observa esta tabla.

Por último, abajo se encuentran tres gráficas que representan la evolución de los correspondientes atributos, con el objetivo de poder analizar algún pico que pueda ser anómalo o peligroso (véase la Figura 5.26).

HIL-mock

Con el objetivo de realizar pruebas simuladas y comprobar el correcto funcionamiento del back-end y el front-end, se desarrolla un simulador de las placas de validación y el vehículo, de forma que se envían estados del vehículo y se reciben paquetes de bits como órdenes para comprobar la correcta transmisión.

Figura 5.25: Tabla de información de la "HIL-GUI"
Fuente: Hyperloop UPV

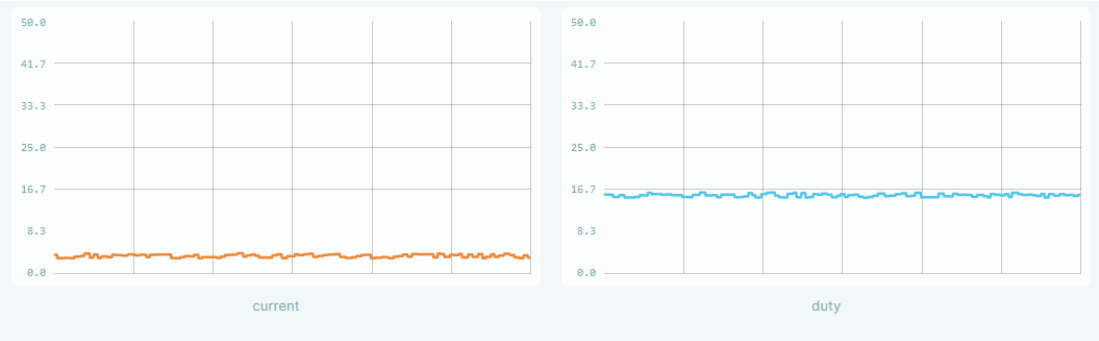

Figura 5.26: Gráficos de la "HIL-GUI"
Fuente: Hyperloop UPV

5.8 Interfaz de usuario "Control-front"

Esta interfaz gráfica de usuario es la aplicación principal y más completa de Hyperloop UPV. Está centrada en la representación de todos los datos de relevancia del vehículo durante sus demostraciones, en el envío de órdenes y la comunicación con el vehículo, y además ofrece una sección donde se retransmite con cámaras situadas en la infraestructura y en el propio vehículo (Figura 5.27). De esta forma tanto los miembros como cualquier persona del público pueda observar lo que sucede en el interior del tubo, mejorando la experiencia de usuario. Dado el tamaño de la aplicación, en esta aplicación sí es necesario desarrollar varias pestañas: monitorización del vehículo y transmisión de datos básicos y cámaras al público.

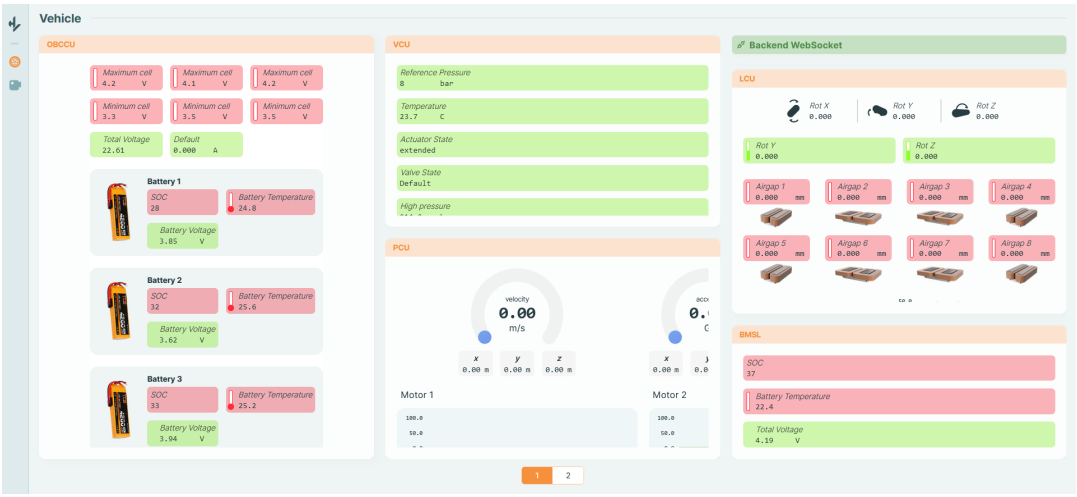

Figura 5.27: Interfaz "Common-front"
Fuente: Hyperloop UPV

5.8.1. Información general del repositorio "control-front"

El repositorio cuenta con 3 contribuidores formados por el subsistema de software del equipo[5]. El repositorio cuenta con 238 commits y 31 ramas. El lenguaje predominante es TypeScript con un 92,4 %, los estilos a través de SCSS ocupan un 2,9 % y de CSS un 4,6 %, un 0,1 % está escrito en HTML. Este repositorio fue creado en enero de 2023, la versión descrita en este documento puede estar sujeta a algunos cambios. Como todos los repositorios que ha realizado el subsistema en la edición, se mantiene open source. El grueso del trabajo ha sido realizado entre marzo y abril. Sin embargo, ha tenido actualizaciones constantes desde su creación hasta la semana de la competición.

5.8.2. Navegación entre pestañas

La navegación entre las diferentes pestañas se controla utilizando React Router. Se define una estructura de enrutamiento con rutas y elementos que determinan qué componentes se debe renderizar al acceder a determinada URL.

Dentro de App se tiene la barra lateral con los diferentes iconos y los path correspondientes:

```
{([handler]) => (
                    <WsHandlerProvider handler={handler}>
                        <Sidebar
                            items={[
                                { path: "/vehicle", icon: <Wheel /> },
                                 { path: "/tube", icon: <Tube /> },
                                { path: "/cameras", icon: <Cameras /> },
                            ]}
                        />
                        <Outlet />
                    </WsHandlerProvider>
                )}
```

Este ''Sidebar'' indica en qué URL se encuentra la aplicación:

[5]El enlace del repositorio de Github "control-front" es https://github.com/HyperloopUPV-H8/control-front

```
export const Sidebar = ({ items }: Props) => {
    const location = useLocation();
    return (
        <nav className={styles.sidebarWrapper}>
            <Link to={"/"}>
                <TeamLogo className={styles.logo} />
            </Link>
            <div className={styles.separator} />
            <div className={styles.items}>
                {items.map((item) => {
                    return (
                        <SidebarItem
                            key={item.path}
                            item={item}
                            isActive={isInSubpath(item.path, location.pathname)
                                }
                        />
                    );
                })}
            </div>
        </nav>
    );
};

function isInSubpath(itemPath: string, currentPath: string): boolean {
    return "/" + currentPath.split("/")[1] == itemPath;
}
```

Cada icono de la barra lateral tiene un NavLink que al ser pulsado produce el cambio al path que le ha llegado en las propiedades:

```
export const SidebarItem = ({ item, isActive }: Props) => {
    return (
        <NavLink
            to={item.path}
            className={`${styles.link} ${isActive ? styles.active : ""}`}
        >
            <div className={styles.iconWrapper}> {item.icon}</div>
        </NavLink>
    );
};
```

Para cambiar de página se usa ''<Navigate />" como se observa en los diferentes fragmentos de código. Cuando la URL coincida con una específica se renderizará ese componente. En este caso, también existen unas subrutas secundarias que están definidas en "vehicleRoute", que concretamente puede tener dos subpestañas, según la URL seleccionada, se renderizará un componente u otro. Para cambiar de pestaña solo se necesita navegar a la URL deseada utilizando ''Navigate". A continuación, se crea un router que navega inicialmente a "vehicle" pero incluye al resto de alternativas:

```
const router = createBrowserRouter([
    {
        path: "/",
        element: <App />,
```

```
        children: [
            { path: "", element: <Navigate to={"vehicle"} /> },
            vehicleRoute,
            camerasRoute,
            tubeRoute,
        ],
    },
]);
```

Dentro de “vehicleRoute” se navega inicialmente a “first”, añadiendo ese path a la URL:

```
export const vehicleRoute = {
    path: "/vehicle",
    element: <VehiclePage />,
    loader: loadPodData,
    children: [
        { path: "", element: <Navigate to={"first"} /> },
        { path: "first", element: <BoardsPage /> },
        { path: "second", element: <ControlPage /> },
    ],
};
```

Se debe tener en cuenta que cada página utiliza el componente ‘’PageWrapper” para darle un formato común y homogéneo a la aplicación:

```
export const VehiclePage = () => {
    useOrders();

    return (
        <PageWrapper title="Vehicle">
            <div className={styles.contentWrapper}>
                <Outlet />
                <Pagination routes={["first", "second"]} />
            </div>
        </PageWrapper>
    );
};
```

```
type Props = {
    title: string;
    children?: React.ReactNode;
};

export const PageWrapper = ({ title, children }: Props) => {
    return (
        <main className={styles.pageWrapper}>
            <header className={styles.header}>
                <h1>{title}</h1>
            </header>
            <div className={styles.content}>{children}</div>
        </main>
    );
};
```

A continuación, se expone cada una de las pestañas de la aplicación.

5.8.3. Monitorización del vehículo

Esta es la página principal de la aplicación, muestra toda la información importante relacionada con el estado del vehículo y los mensajes de advertencias y errores que envía. Además, proporciona el control necesario para iniciar las demostraciones y parar el vehículo manualmente en caso de emergencia. Dado que es la parte más completa de la aplicación, requiere de dos subpestañas para poder mostrar toda la información. Para cambiar entre ellas tan solo es necesario pulsar un botón que se encuentra en la parte inferior. A continuación, se expone la información de cada una de ellas.

Primera pestaña

Existe una sección de control para cada una de las principales placas del vehículo. En cada una de ellas se representa de manera visual la información más relevante que aportan cada uno de los circuitos impresos y sus sistemas con códigos de colores. El objetivo es obtener de un rápido vistazo todos los valores de interés y resaltar aquellos que no se encuentren en su respectivo rango de seguridad. En la Figura 5.28 se observa esta pestaña cuando todavía no ha sido inicializado con el vehículo.

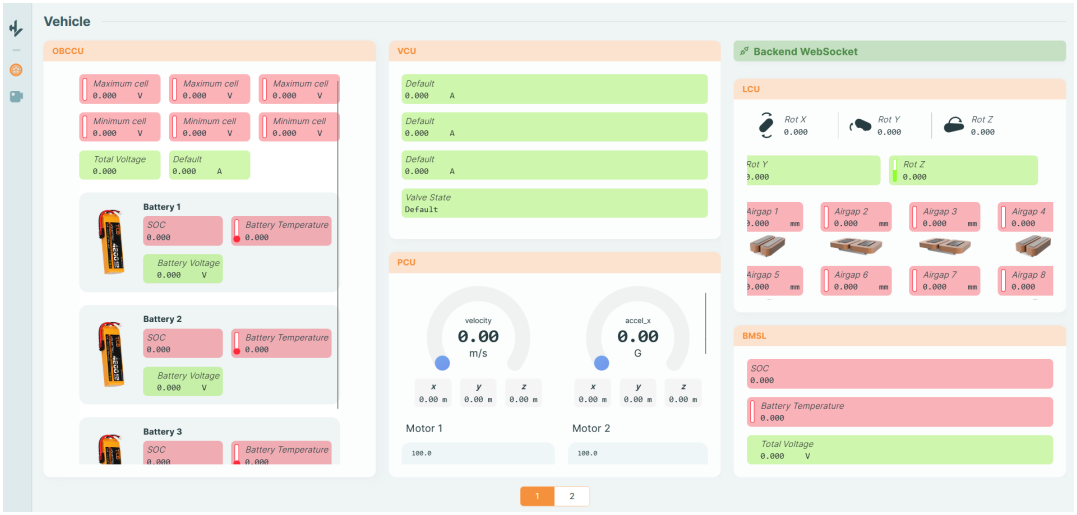

Figura 5.28: Primera pestaña de la monitorización del vehículo "control-front" sin inicializar
Fuente: Hyperloop UPV

El primer componente contiene la información sobre la OBCCU (Figura 5.29), que es la placa que controla los elementos de protección del circuito y recibe el estado de las baterías de alto voltaje. Esta información viene complementada con la información de la placa BMSH (Battery Management System High-Voltage) que no tiene una sección independiente para sus valores. En ella se mantiene información de los voltajes de las celdas, el voltaje total, y el voltaje y temperatura de cada una de las baterías.

El componente de arriba de la parte central de la pestaña se corresponde con la placa VCU, encargada de centralizar la comunicación del vehículo y de gestionar la navegación. Esta sección permite guardar la información de variables de interés de diferentes sistemas del vehículo, como por ejemplo el sistema de frenado, como es el caso de la Figura 5.30, donde se observa el estado de las válvulas, la presión de referencia, etc.

El componente central inferior se corresponde con la PCU (Figura 5.31), que es la placa que controla la propulsión. Muestra los principales atributos de velocidad y acele-

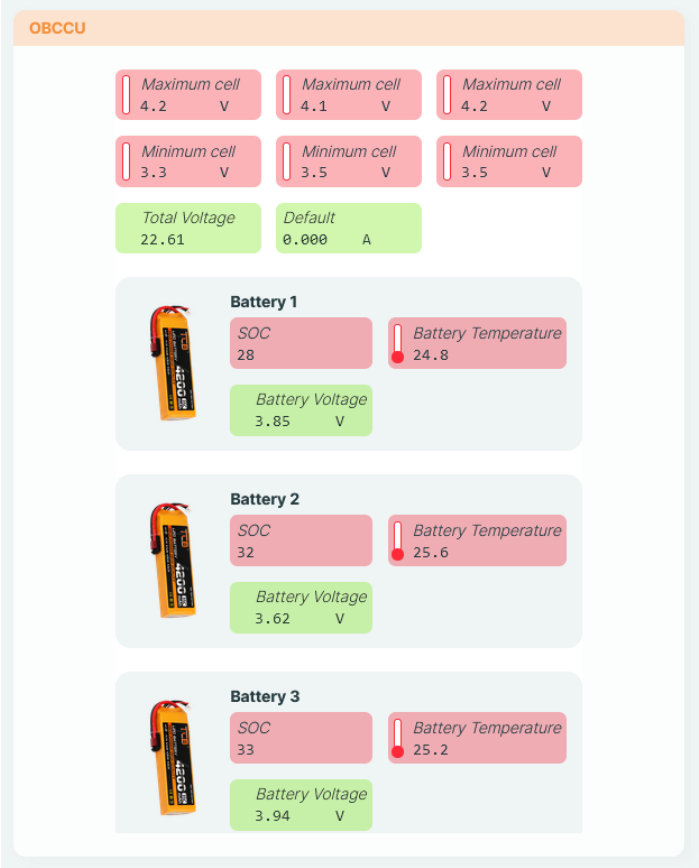

Figura 5.29: Información sobre la OBCCU
Fuente: Hyperloop UPV

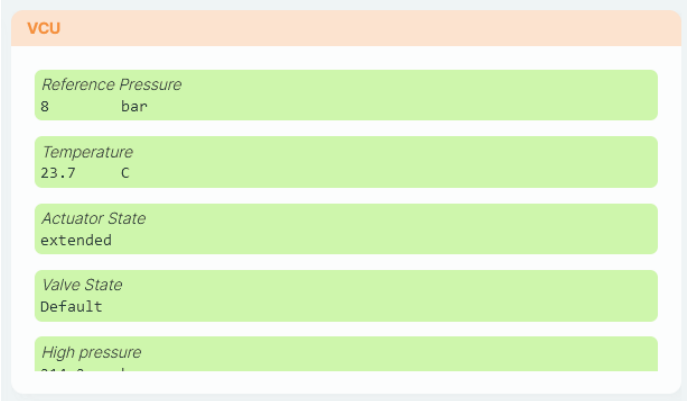

Figura 5.30: Información sobre la VCU
Fuente: Hyperloop UPV

ración, junto con gráficas de la corriente en el motor de inducción lineal, y la temperatura tanto del motor como de la placa. A modo de ejemplo se visualiza el código de esta sección, el cual es similar en todas. Como propiedades se recibe una estructura incluida en el repositorio "common". A partir de ahí se crea el contenedor y se van incluyendo todos los atributos de interés con diferentes componentes creados para mostrar de manera visual cada dato, como por ejemplo, ''VectorGauge", ''MotorInfo", etc.

```
 import styles from "./PCU.module.scss";
import { Window } from "components/Window/Window";
import { VectorGauge } from "./VectorGauge/VectorGauge";
```

Figura 5.31: Información sobre la PCU
Fuente: Hyperloop UPV

```
import { MotorInfo } from "./MotorInfo/MotorInfo";
import { PcuMeasurements } from "common";
import { TempTag } from "./TempTag/TempTag";
import motorUrl from "assets/images/motor.png";
import pcbUrl from "assets/images/pcb.png";
export const PCU = (props: PcuMeasurements) => {
    return (
        <Window title="PCU">
            <section className={styles.pcuSectionWrapper}>
                <VectorGauge
                    x={props.velocity_x}
                    y={props.velocity_y}
                    z={props.velocity_z}
                />
                <VectorGauge
                    x={props.accel_x}
                    y={props.accel_y}
                    z={props.accel_z}
                />
                <MotorInfo
                    title="Motor 1"
                    motorCurrentU={props.motor_a_current_u}
                    motorCurrentV={props.motor_a_current_v}
                    motorCurrentW={props.motor_a_current_w}
                />
                <MotorInfo
                    title="Motor 2"
                    motorCurrentU={props.motor_b_current_u}
                    motorCurrentV={props.motor_b_current_v}
                    motorCurrentW={props.motor_b_current_w}
                />
                <TempTag
                    meas={props.max_motor_a_temperature}
                    icon={
                        <img
                            src={motorUrl}
```

```
                                style={{
                                    objectFit: "contain",
                                    width: "9rem",
                                    height: "4rem",
                                }}
                            />
                        }
                    />
                    <TempTag
                        meas={props.max_ppu_a_temperature}
                        icon={
                            <img
                                src={pcbUrl}
                                style={{
                                    objectFit: "contain",
                                    width: "4rem",
                                    height: "4rem",
                                }}
                            />
                        }
                    />
                </section>
            </Window>
        );
};
```

En la esquina superior derecha se representa la sección de control de la levitación, monitoriza la placa LCU. En ella se representa el airgap (distancia entre el vehículo y el raíl) para cada uno de los sensores del vehículo, que analizan todos los ejes. En las representaciones visuales se representa su posición en el vehículo, de manera que los miembros del equipo pueden entender de forma rápida a qué medida se refiere cada airgap, además, están numerados según la posición que tienen en el vehículo (véase la Figura 5.32). Adicionalmente, también se analiza la rotación de cada uno de los ejes para observar su posición. Con estos datos, el sistema de control diseñado por Electromagnetics aplica una corriente eléctrica determinada para corregir la posición en cada zona del vehículo. Estas corrientes se observan en las gráficas situadas en esta misma sección.

Por último se encuentra la sección del BMSL, es muy similar a la de la OBCCU pero esta muestra la información de las baterías de bajo voltaje que alimentan los circuitos (Figura 5.33).

Esta pestaña cuenta además con un componente que muestra si la conexión con el back-end está activa, en la Figura 5.34 se puede observar activada.

Segunda pestaña

La segunda pestaña de la sección de control se asemeja en las diferentes secciones a la interfaz “Ethernet View”, también está dividido en tres columnas: la sección de órdenes, la sección de mensajes y el bootloader, y la sección de conexiones, escritura sobre fichero y las principales instrucciones del vehículo (véase la Figura 5.35). Como en la interfaz “Ethernet View” ya se ha explicado la utilidad de estos componentes, en este apartado se detalla el modo de funcionamiento de las órdenes de estado, de los mensajes y, sobre todo, de las órdenes de máxima prioridad.

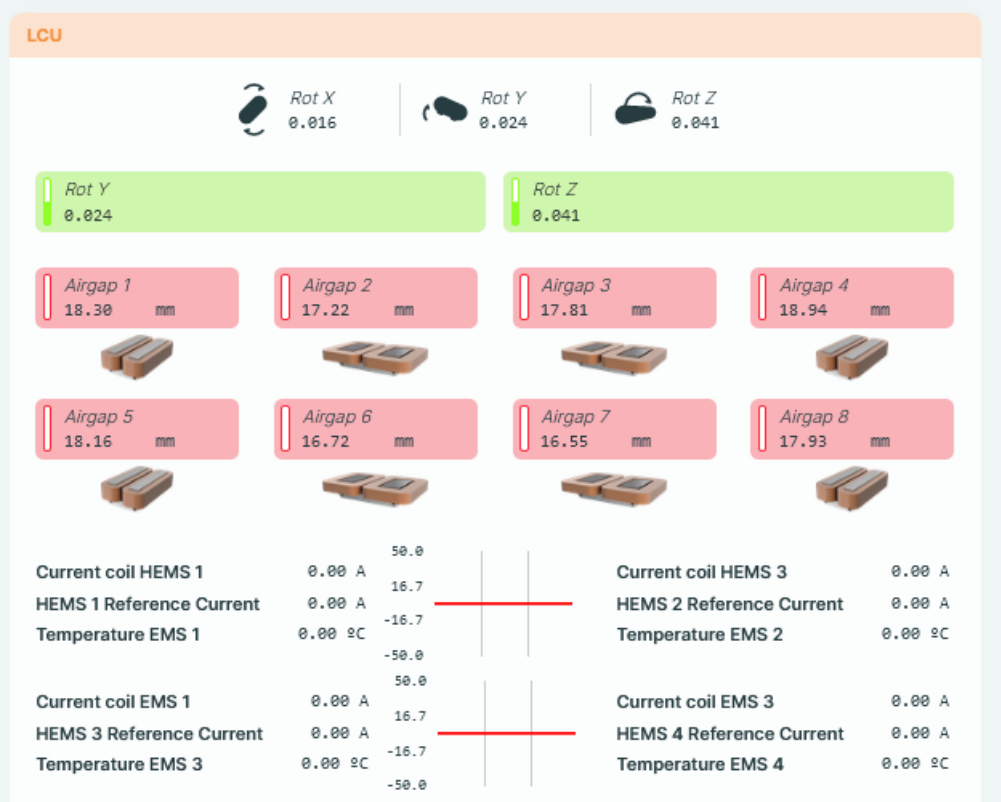

Figura 5.32: Componente LCU
Fuente: Hyperloop UPV

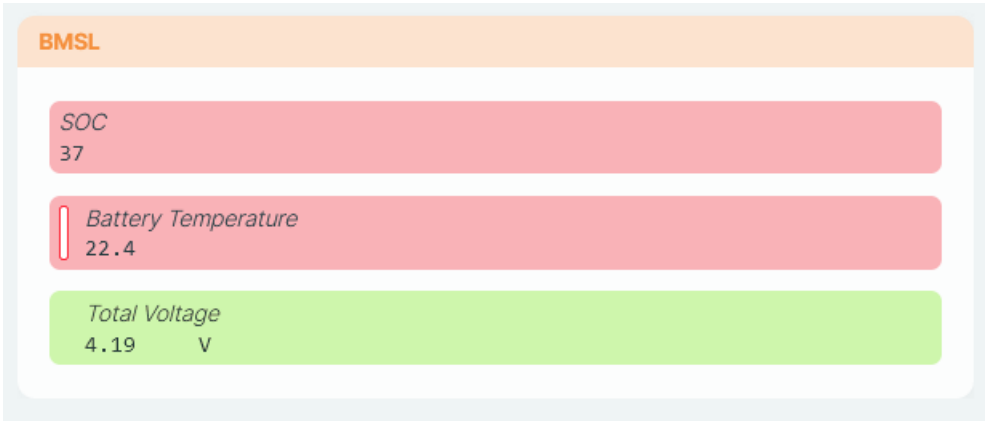

Figura 5.33: Componente BMSL
Fuente: Hyperloop UPV

Figura 5.34: Indicador de conexión con el back-end
Fuente: Hyperloop UPV

Órdenes de estado

El control directo sobre el vehículo es necesario en diversos escenarios, por ejemplo, los frenos están por defecto extendidos y frenando el vehículo, por lo tanto, se debe enviar la orden de "desfrenar" para poder introducirlo en la infraestructura Atlas. Sin embargo, no todas las órdenes pueden enviarse de manera manual en cualquier estado, por ejem-

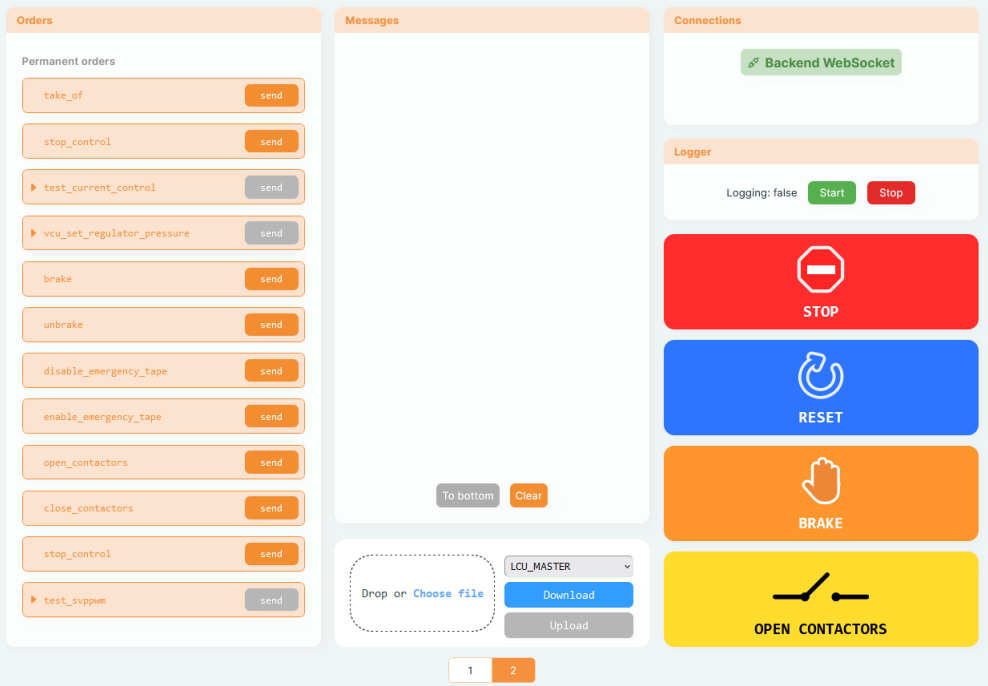

Figura 5.35: Segunda pestaña de monitorización del vehículo
Fuente: Hyperloop UPV

plo la propulsión y la levitación pueden ser un peligro si son activadas manualmente sin hacer los previos chequeos. Se necesita un componente que actualice el estado en el que se encuentra el vehículo para que actualice el listado de órdenes que pueden ser utilizados. Esta información es transmitida desde el vehículo a través de la placa VCU (Vehicle Control Unit), en cuya API hay una opción de envío de las órdenes disponibles en cada paso de la demostración. De esta manera, cuando el estado de la VCU cambia, notifica a la Control Station con una nueva lista de órdenes permitida. Esta sección utiliza una lista de botones para visualizar cada una de las órdenes disponibles junto con el estado actual de la VCU, garantizando que la totalidad de las órdenes utilizables en cada momento es seguro y adecuado, a modo de ejemplo, la Figura 5.36 muestra las órdenes que aparecen en el estado ”HealthCheck”.

Consola de errores y advertencias

El vehículo envía un mensaje a la Control Station cuando entra en estado “fault” indicando qué protección ha sido violada; esta información llega a la estación de control y el equipo puede identificar a que se ha debido el bloqueo del vehículo. Los mensajes de error contienen los siguientes campos:

- Un número de 16 bits donde se identifica el paquete como un paquete de error.
- La placa que ha detectado el error y ha creado el mensaje.
- El nombre del valor que ha causado el fallo.
- El tipo de protección que se ha violado.
- El valor que ha causado el estado “fault”.

Los tipos de protecciones incluidas para los valores numéricos son OUT_OF_RANGE, BELOW, ABOVE, EQUALS, Y NOT_EQUALS. En el caso de OUT_OF_RANGE, esta pro-

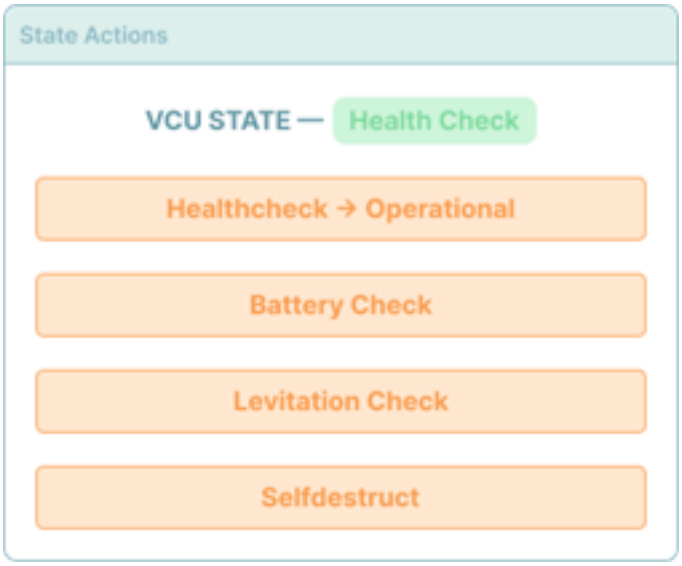

Figura 5.36: Diagrama de órdenes en el estado "HealthCheck"
Fuente: Hyperloop UPV

tección es violada si el valor recibido se encuentra fuera del rango asignado como correcto. Cada tipo de protección establece el resto de información asignada a los campos del mensaje. Por ejemplo, para esta protección, aparecerá la información de cuáles son los límites superiores e inferiores establecidos para el valor que ha generado el error. En la Figura 5.37 se puede observar la apariencia de un mensaje de error.

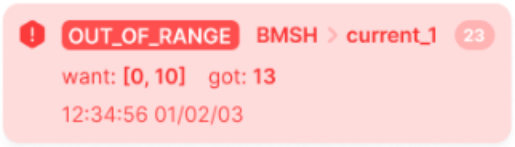

Figura 5.37: Mensaje de error
Fuente: Hyperloop UPV

Por otro lado, el vehículo puede enviar mensajes de advertencia por si algún dato no es incorrecto pero se está acercando a valores límite. En el caso de OUT_OF_RANGE se cuenta con un intervalo óptimo, si se excede se entra en el intervalo de advertencia, que marca los límites en los que se considera error y se debe bloquear el vehículo.

Órdenes de máxima prioridad

Para garantizar el control sobre cualquier escenario y emergencia, existen dos órdenes que están siempre disponibles en el front-end, sin importar el estado en el que se encuentre el vehículo o la infraestructura. Este recurso está destinado a utilizarse cuando fallan todas las demás opciones. Las dos órdenes son "parada de emergencia" y "reinicio del sistema", tanto para el vehículo como para la infraestructura. Cuando ocurre cualquier error se presiona el botón de parada de emergencia, una vez se ha resuelto la incidencia, se pulsa el restablecimiento del sistema para devolver el vehículo o la infraestructura a su estado inicial. Estas órdenes no se encuentran solo en el front-end, sino que también existen esos botones físicos en el vehículo e infraestructura. Estas dos órdenes vienen acompañadas de otras dos órdenes de emergencia de suma importancia, el botón "Brake" y el botón de abrir contactores "Open contactors", para que puedan ser llamados en cualquier momento (véase la Figura 5.38).

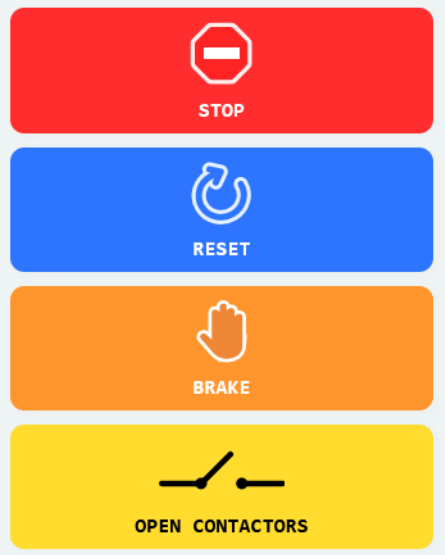

Figura 5.38: Órdenes de máxima prioridad
Fuente: Hyperloop UPV

5.8.4. Página de cámaras

El Control-front cuenta con una página centrada en la visualización del vehículo durante la demostración. Como Kenos se encuentra dentro del tubo, solo es visible desde las ventanas situadas en las puertas de la infraestructura. Con el objetivo de que todo el público pueda ver las demostraciones, tres cámaras están situadas a lo largo de la infraestructura (inicialmente se diseñó para cuatro cámaras, al igual que la infraestrctura de red). Como se ha comentado en la sección "Infraestructura de red", la transmisión del vídeo se hace vía WebRTC, un protocolo diseñado para comunicación de vídeo en tiempo real. Esta transmisión está disponible en la página de cámaras del Control-front, en la Figura 5.39 se puede observar su apariencia.

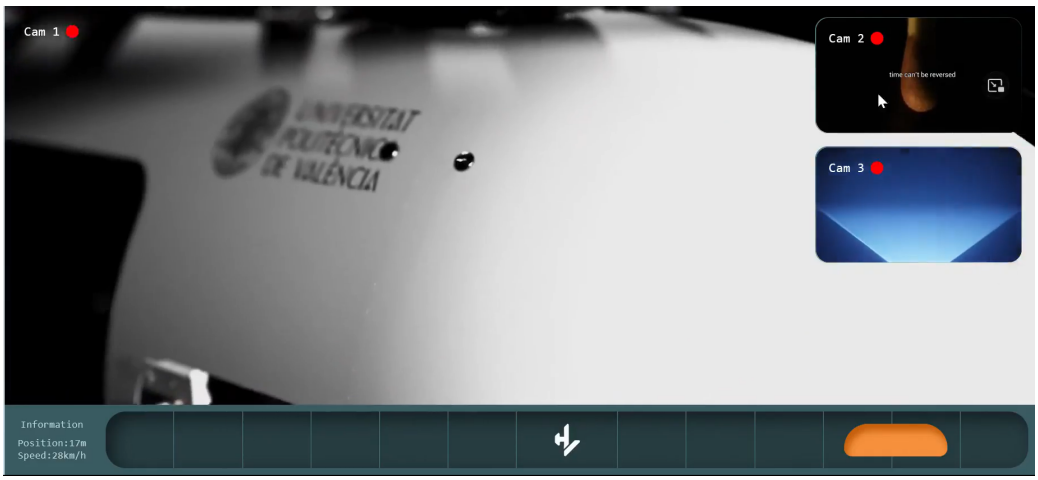

Figura 5.39: Página de retransmisión de las cámaras
Fuente: Hyperloop UPV

5.8.5. Interfaz de usuario "Mobile-front"

Este repositorio[6] tan solo es una adaptación del control-front para versión de móviles, limitando funcionalidades, por lo que es un proyecto menor y ha sido desarrollado en los meses de junio y julio (véase la Figura 5.40).

Figura 5.40: Interfaz de usuario "Mobile-front"
Fuente: Hyperloop UPV

5.9 Testing y software de prueba

Esta sección tan solo cuenta con una breve descripción de la metodología utilizada por el subsistema, no se detallan los procesos de verificación y validación del front-end.

Inicialmente, React ofrece un modo llamado "Strict Mode", que monta los componentes dos veces en la primera carga de la página. Este método ayuda a identificar efectos secundarios inesperados, mejorando la depuración de los componentes y la conexión entre el front-end y el back-end. Por ejemplo, en estas aplicaciones hay componentes que establecen un WebSocket con el back-end cuando se crean, como React renderiza dos ve-

[6]El enlace del repositorio de Github "mobile-front" es https://github.com/HyperloopUPV-H8/mobile-front

ces al inicio, la primera conexión se descarta. Si el back-end no es capaz de gestionar la pérdida de conexiones, empiezan a aparecer errores.

Otra herramienta utilizada para verificar el correcto funcionamiento del front-end es React Developer Tools. Es una extensión del navegador que permite inspeccionar una web de React de forma similar a las Herramientas de Desarrollador del navegador, pero desde la perspectiva de React. Se representan los componentes de la web en forma de árbol, muestra gráficamente cuándo se rerenderiza cada componente permitiendo identificar ineficiencias, indica valores de estados y hooks, y permite comprobar el funcionamiento de cada uno de los componentes y eventos.

Además, se han desarrollado múltiples repositorios privados de simulación de envío de paquetes, de conexiones, de mensajes, etc. con el objetivo de simular información del vehículo que recibe el back-end, para comprobar el funcionamiento de cada uno de los repositorios. Detallando diferentes casos de uso, comprobando la robustez de cada uno de los proyectos y, depurando y arreglando los diferentes bugs que se fuesen encontrando con estas pruebas.

CAPÍTULO 6
Resultados, propuestas de mejora y conclusiones

Este capítulo se enfoca en el análisis de los resultados obtenidos y se proponen una serie de mejoras factibles para el sistema. Posteriormente, se exponen las conclusiones fundamentales de este trabajo.

6.1 Resultados y propuestas de mejora

El proyecto ha tenido un buen desempeño en las actividades principales del subsistema Software, descritos en este Trabajo de Fin de Grado. Los requerimientos identificados han sido adecuados, y los diseños desarrollados han cumplido las necesidades detectadas.

La arquitectura de red diseñada ha permitido cumplir con todos los requisitos planteados para comunicar el vehículo y la infraestructura, con la estación de control. La implementación ha resultado ser sólida y funcional. En ningún caso ha resultado ser un cuello de botella ninguna conexión y, por lo tanto, no ha habido ralentizaciones ni pérdidas por parte del diseño de la red.

Por otra parte, esta arquitectura tenía otra función adicional que consistía en retransmitir las demostraciones a través de cámaras para la audiencia que estuviese presente. Sin duda, está es la parte más exigente de la red y la que mayores problemas podía ocasionar. Inicialmente se realizó un diseño con tres puntos de acceso que pertenecían al equipo. Sin embargo, este diseño tan solo permitía la conexión de aproximadamente 60 personas, y se estimó que durante la demostración de la competición podíamos llegar a un pico de 200 personas. El punto de acceso alternativo que decidió comprarse permitía alcanzar el servicio para esas 200 personas teóricas, aunque su límite no era mucho mayor. Finalmente, durante el día de la competición no se conectaron todas las cámaras y la transmisión de información por cada conexión disminuyó, permitiendo un mayor número de espectadores. No hubo ningún tipo de limitación en cuanto al rendimiento por parte de la red.

Por lo tanto, la implementación del nuevo punto de acceso ha resultado ser efectiva, aunque habría sido aconsejable disponer de un poco más de ancho de banda para tener un rango de seguridad. Dada la limitación económica que tiene el proyecto, no era posible obtener productos más potentes. Por lo tanto, la propuesta de mejora en esta actividad consiste en utilizar este nuevo punto de acceso de forma simultánea junto a los tres puntos de acceso anteriores, intentando unificarlos en una misma SSID (nombre público

de una red de área local inalámbrica), de esta forma se alcanzaría un ancho de banda que soporta 260 conexiones simultáneas teóricas.

Por otra parte, esta red no está conectada a internet para evitar cualquier vulnerabilidad y un posible ataque externo. Como propuesta de mejora se propone su incorporación a internet, implementando cortafuegos y cualquier medida de seguridad necesaria, y prestando especial atención a la autentificación de usuarios, por ejemplo a través de Auth0, para fortalecer la protección de la red y los datos. Habría que valorar hasta qué punto la parte de control del vehículo sería segura conectada a internet, y si se detectase cualquier vulnerabilidad separarlo dejándolo de forma local. Para la conexión del público esta situación es menos crítica y se puede facilitar su conexión a través de internet simplemente, aportando comodidad a los usuarios.

Por último, también es aconsejable buscar una alternativa para la recepción de paquetes en el back-end. El funcionamiento actual se basa en la inspección de los paquetes enviados entre placas, duplicado y envío al exterior. Se debe estudiar una forma óptima de realizarlo.

En segundo lugar, el back-end actual demuestra ser altamente modular y fácilmente refactorizable. Su diseño contribuye a la eficiencia del programa y la mantenibilidad del código. Además, está desarrollado de tal forma que permite añadir funcionalidad al proyecto de manera sencilla. La independencia de los módulos permite la realización de cualquier cambio en cada uno de ellos sin necesidad de modificar módulos adyacentes. Por ejemplo, si se modificase la estructura de los datos trasmitidos a través de los paquetes, tan solo habría que modificar la hoja de cálculo que especifica este orden. Y si, por ejemplo, se cambiasen las conexiones con el vehículo, esto solo afectaría al módulo correspondiente. El resto trabajaría de igual forma ya que los outputs de ese módulo no varían. Sin embargo, se han identificado algunas áreas donde se podrían realizar mejoras significativas:

- No se ha implementado un mecanismo de gestión de los fallos de la descarga de la hoja de cálculo con la información de los paquetes. Una gestión robusta de ellos permitiría una mejor confiabilidad del sistema. Algunas propuestas podrían ser el reintento automático de descarga, una notificación de errores más detallada, etc.
- Por otra parte, la manipulación de la hoja de cálculo es manual por parte de los miembros de Avionics del equipo y por lo tanto, están sujetos a errores humanos. Es interesante incorporar controles adicionales al manejo de este archivo para prevenir este tipo de errores, incluyendo validaciones o verificaciones adicionales de forma previa al procesamiento de sus datos. Este desarrollo facilitaría el proceso de depuración y agilizaría la etapa de testing del equipo.
- La conexión del back-end con el vehículo es limitada: actualmente, solo una única conexión WebSocket permite iniciar la sesión y trabajar con el vehículo. Si se permitiese la conexión de múltiples conexiones se podrían añadir varios usuarios conectados simultáneamente, permitiendo trabajar desde diferentes ordenadores. Sería necesario diseñar la gestión de las órdenes cuando hay varios usuarios conectados, quizá a partir de un superusuario que controla, y el resto que simplemente visualiza datos.

Con el desarrollo de estas propuestas de mejora el back-end permitiría mucha mayor agilidad durante las etapas de pruebas del vehículo y de demostraciones, permitiendo analizar en paralelo y minimizar los errores humanos.

Por último, con el objetivo de cumplir con los requerimientos del front-end, ha sido necesario desarrollar diversas interfaces de usuario. La decisión de separar la aplicación

de testing, la aplicación que permite la manipulación de la "Software and Hardware Unit Test Platform" y la aplicación de control principal ha sido un acierto. Aunque ha significado una carga de trabajo superior, la implementación del repositorio "common-front" para alojar el código y los componentes comunes ha permitido ahorrar desarrollo de código.

Se ha logrado realizar un diseño limpio que alberga toda la información sin necesidad de desplazarse a través de numerosas pestañas en las aplicaciones de testing. Mientras, en la aplicación principal, a pesar de contener mucha más información, se ha agrupado de manera efectiva por sistemas, de manera que se puede observar la información necesaria de los sistemas relacionados de un simple vistazo.

La interfaz de usuario "Ethernet-View" fue la primera en desarrollarse para las primeras pruebas de validación del equipo. Pese a que inicialmente estaba diseñada con ciertas limitaciones, las solicitudes que han ido realizando los diferentes sistemas para el desarrollo de las pruebas ha permitido una interfaz mucho más completa, a partir de un back-end modular que también ha crecido en cuanto a funcionalidades.

Esta primera interfaz no permitía verificar todas las pruebas del vehículo, por lo que el desarrollo de la HIL-GUI cobró especial interés para comprobar cómo actúa el vehículo ante las perturbaciones. Las dificultades del subsistema Firmware para terminar la plataforma de testing de esta interfaz ha provocado que ciertas funcionalidades de este front-end no hayan llegado a ser utilizadas en la etapa de pruebas del vehículo. Sin embargo, sí ha sido validada por el subsistema a través de simulaciones.

Por último, la interfaz principal ha supuesto el grueso del trabajo del equipo durante los últimos meses. Ha sufrido diversos cambios provocados por el resto de los subsistemas. Por ejemplo, se tuvo que prescindir de la pestaña de monitorización de la infraestructura porque no hubo tiempo de implementarla por parte del resto de subsistemas de Avionics. Sin embargo, se ha conseguido implementar una interfaz sencilla, intuitiva, funcional y eficiente. Además, se ha podido desarrollar la versión "mobile-front" para los móviles, que fue utilizada por el público durante la competición. Por otra parte, la gestión del envío de las órdenes es sencilla y los botones de órdenes de máxima prioridad son visibles. En cuanto a la transmisión de las cámaras en la aplicación, tiene una navegación sencilla que facilita la visualización de las demostraciones, aportando datos de posición del vehículo e información genérica de la prueba.

Aún así, se han detectado numerosas posibilidades de mejora, las más destacadas se describen a continuación:

En primer lugar, es necesario para la próxima edición desarrollar una pestaña que permita la monitorización de la infraestructura. Durante esta edición se ha comenzado a desarrollar, pero por dificultades con los plazos por parte de algunos subsistemas, no ha dado tiempo a terminarla, por lo que el diseño para la interfaz también se ha paralizado. Con esta propuesta, se permitiría monitorizar toda la información de presión y temperatura del tubo y detectar posibles problemas o pequeñas fugas, así como presentar esta información a los usuarios y complementar la experiencia de usuario no solo con información del vehículo y con las cámaras, sino también con la información de la infraestructura.

En segundo lugar, es muy recomendable desarrollar un repositorio oficial de simulación de paquetes para cada uno de los sistemas del vehículo y de la infraestructura con el objetivo de facilitar los procedimientos de verificación y de validación que tienen los proyectos de software. A pesar de que se han desarrollado de manera independiente, el objetivo de ellos ha sido aportar pruebas concretas y específicas. Sería interesante contar con el desarrollo de un repositorio que simulase de forma completa el vehículo, con el envío de paquetes, simulación de las conexiones, envío de mensajes, etc.

En tercer lugar, las interfaces de usuario de pruebas y de control solo podían ser ejecutadas en un ordenador concreto, pero las aplicaciones en sí no contaban con ningún tipo de inicio de sesión ni de confirmación para que solo pudiese ser manipulado por las personas correctas ya que cualquier persona que utilizase ese ordenador podía manipularlo. De cara a siguientes ediciones es interesante desarrollar un inicio de sesión de forma que no sea ejecutable solo desde un ordenador concreto controlado, sino también por una persona preparada para utilizarlo.

Otra propuesta de mejora consiste en permitir la ejecución simultánea de varios usuarios en la aplicación de control del vehículo. Esta propuesta comentada para el back-end tiene que ser diseñada para el front-end de forma que se diferencie qué persona está manipulando realmente el vehículo, facilitando al mismo tiempo el análisis y la visualización del resto de usuarios.

La aplicación de la HIL-GUI, por el momento, define botones personalizados para que se definan perturbaciones concretas. Una propuesta de mejora sería definir un formulario que permitiese añadir los valores que interesan a cada atributo para generar las perturbaciones directamente, sin tener que definirlas de forma previa. Manteniendo estas dos funcionalidades se consiguiría la agilidad de las perturbaciones predefinidas, y la personalización del formulario.

Por último, de cara a la experiencia del público, es de sumo interés permitir una interfaz interactiva que permita añadir y quitar los atributos que quiere observar cada usuario, con un “drag and drop” o algún formato similar, por lo que el diseño nuevo habría que estudiarlo y definirlo.

En definitiva, el resultado ha sido muy satisfactorio en las tres principales tareas que describe el Trabajo de Fin de Grado, al tiempo que se han detectado mejoras para continuar la curva creciente que sigue el equipo en cuanto al desarrollo en todos los subsistemas.

6.2 Conclusiones

Aunque la comunicación entre las placas en el interior del vehículo y el funcionamiento de su electrónica, en general, depende fundamentalmente de firmware y hardware, es imprescindible el desarrollo de un software que permita procesar todas esas comunicaciones y enviárselas a la estación de control para que sea interpretada y manipulada por los usuarios.

Por otro lado, la utilización de un tubo como infraestructura donde se desplaza el vehículo ha imposibilitado el acceso a los botones físicos de los que dispone y en general, cualquier tipo de manipulación del vehículo. Por esta razón el subsistema Software ha obtenido un aumento de protagonismo, con la gestión remota, que ya se implementaba en las ediciones anteriores. Además, se han añadido funcionalidades respecto a otros años con el desarrollo de una aplicación de testing para la comunicación de las placas Ethernet con ”Ethernet-View”. Hasta el momento, como se había utilizado el protocolo CAN, se utilizaba un software comercial. La aplicación de generación de perturbaciones también es completamente novedoso. A la aplicación principal se le ha añadido el bootloader para cargar código en las placas y la retransmisión de las demostraciones a través de cámaras para mejorar la experiencia del usuario, entre otras cosas.

Esta evolución del subsistema ha generado un interés adicional en el desarrollo de un Trabajo de Fin de Grado sobre este tema.

La infraestructura de red ha sido más compleja que las ediciones anteriores, ya que había que contar con la retransmisión de las cámaras y el soporte para 200 conexiones

simultáneas. Las pruebas de validación han comprobado la efectividad del diseño, primando la sencillez y la eficiencia. Ha establecido una base sobre la que iterar para futuras ediciones, y se han propuesto nuevos desarrollos para continuar desarrollando una red más sólida.

El desarrollo del back-end ha permitido trabajar cómodamente con los datos y desarrollar las diferentes interfaces de usuario. Con un desarrollo simple y modular, se ha permitido ofrecer mucha información y desarrollar nuevas funcionalidades sin necesidad de modificar en gran medida el proyecto. Las conexiones UDP cuentan con una tasa de pérdidas muy reducida y las órdenes gestionadas por TCP no generan ningún problema.

En cuanto el trabajo desarrollado para el front-end, se ha conseguido un grado de profesionalización no alcanzado hasta el momento, y se ha pasado de la elaboración de una única interfaz de usuario dedicado al control de las demostraciones, a tres interfaces independientes y una versión de la principal adaptada para móviles. El diseño ha sido elaborado siguiendo las pautas establecidas, y la funcionalidad tiene como objetivo la efectividad y sencillez.

En definitiva, el cumplimiento de los requerimientos de las diferentes actividades ha sido acertada, se han realizado diseños ajustados, simples y eficientes. Además, la gestión del trabajo del subsistema ha sido buena, y el seguimiento de las metodologías ágiles ha resultado muy apropiada para las características del trabajo.

La elección de herramientas y lenguajes ha sido acertada. No se ha tenido que sacrificar ninguna funcionalidad debido a limitaciones en las herramientas; de hecho, estas han agilizado el proceso de desarrollo en comparación con ediciones anteriores.

Aunque generalmente no es común basarse en desarrollos previos debido a que la mayoría del equipo se renueva por completo y se introducen nuevos requisitos para el vehículo, estos repositorios pueden servir de una base muy sólida para la próxima edición. Esta base permite centrarse en las adaptaciones al nuevo vehículo, sus placas y nuevos protocolos de comunicaciones, pero ahorrando esfuerzos para implementar las propuestas de mejora planteadas, sin necesidad de desarrollar de nuevo la base central del subsistema.

En cuanto al desarrollo del vehículo Kenos y la infraestructura Atlas, pese a la ambición del proyecto y las limitaciones temporales y económicas, se han desarrollado de manera exitosa la gran mayoría de los objetivos tecnológicos propuestos para la edición. Kenos es el primer vehículo presentado a la European Hyperloop Week que cumple las características de un vehículo hyperloop real: dispone de un sistema de levitación, es propulsado a través de un motor de inducción lineal evitando el rozamiento con cualquier superficie para impulsarse y, además, puede funcionar en condiciones de "vacío" en una infraestructura desarrollada por el equipo que permite este contexto. Todas las demostraciones del equipo fueron probadas y documentadas en la Universidad Politécnica de Valencia de manera previa a la competición, aunque durante la European Hyperloop Week, en Edimburgo, un inconveniente que no pudo ser resuelto por la falta de medios en esa ubicación, evitaron que se pudieran realizar todas las demostraciones durante el evento.

Por todo ello, considero que el desarrollo de las actividades descritas en este Trabajo de Fin de Grado ha significado una solución acertada a las necesidades del equipo, permitiendo respaldar la tendencia ascendente que tiene el equipo en el ámbito tecnológico y establecer unas bases para las futuras ediciones del subsistema Software que permitirán un progreso más avanzado. Las propuestas descritas en este documento serán valoradas por los miembros de la siguiente edición con el objetivo de realizar su implementación y apoyar la mejora continua del proyecto.

A nivel personal, las tareas han supuesto una carga de trabajo elevada, pero el valor que aporta en cuanto a las competencias adquiridas y el complemento que supone a la titulación han supuesto un aspecto diferencial para mi desarrollo. Mi participación en el proyecto universitario Hyperloop UPV ha sido una experiencia enriquecedora, no solo en el ámbito profesional y académico, sino también en el personal. Mi participación en la arquitectura de red y en el desarrollo de software me ha permitido adquirir una experiencia adicional muy valiosa para mi formación, complementando la formación académica que he recibido de la doble titulación y aproximando estos aspectos teóricos a la práctica del mundo laboral.

Este trabajo me ha permitido adquirir una comprensión global de la forma de trabajo de un proyecto de software y significa un inicio de gran valor para mi desarrollo profesional.

Bibliografía

[1] Bierman, G., Abadi, M., & Torgersen, M. (2014). Understanding typescript. En ECOOP 2014–Object-Oriented Programming: 28th European Conference, Uppsala, Suecia, 28 de julio a 1 de agosto, 2014. Proceedings 28 (pp. 257-281). Springer Berlin Heidelberg.

[2] Canós, J. H., Letelier, P., & Penadés, M. C. (2003). Metodologías ágiles en el desarrollo de software. *Universidad Politécnica de Valencia, Valencia*, 1-8.

[3] Delía, L. N., Galdamez, N., Thomas, P. J., & Pesado, P. M. (2013). Un análisis experimental de tipo de aplicaciones para dispositivos móviles. En XVIII Congreso Argentino de Ciencias de la Computación.

[4] EHW (noviembre de 2022). EHW 2023 Rules & Regulations Recuperado el 18 de abril de 2023, de https://hyperloopweek.com/event/

[5] Escalona, M. J., & Koch, N. (2002). Ingeniería de Requisitos en Aplicaciones para la Web–Un estudio comparativo. Universidad de Sevilla.

[6] Fedosejev, A. (2015). React. js essentials. Packt Publishing Ltd.

[7] Fernández Gago, J.A., Collado Pérez-Seoane, F. (2021). QUANTIFICATION OF TRANSPORT OFFER LINKED TO A EUROPEAN HYPERLOOP NETWORK. Recuperado el 12 de marzo de 2023, DOI: 10.36443/10259/6986

[8] Fette, I., & Melnikov, A. (2011). The websocket protocol (No. rfc6455).

[9] Garreau, M. (2018). Redux in action. Simon and Schuster.

[10] Gitflow Workflow. Atlassian Git Tutorial. Recuperado el 16 de junio de 2023. Consultado en https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow.

[11] Hansen, I. A. (2020). Hyperloop transport technology assessment and system analysis, Transportation Planning and Technology. 43:8, 803-820, DOI: 10.1080/03081060.2020.1828935

[12] Hirde, A., Khardenavis, A., Banerjee, R. et al. (2022). Energy and emissions analysis of the hyperloop transportation system. Environ Dev Sustain. DOI: 10.1007/s10668-022-02393-5

[13] Hyung-Woo, L., Ki-Chan, K. y Ju L. Review of maglev train technologies (2006). *IEEE Transactions on Magnetics*. vol. 42, no. 7, pp. 1917-1925. DOI: 10.1109/TMAG.2006.875842.

[14] Lluesma, F., Arguedas, A., Hoyas, S., A. Sánchez, y Vicén, J. (2021). Evacuated-Tube, High-Speed, Autonomous Maglev (Hyperloop) Transport System for Long-Distance Travel: An overview. *IEEE Electrification Magazine*. vol. 9, no. 4, pp. 67-73 DOI: 10.1109/MELE.2021.3115543

[15] Macola, I. G. (25 de marzo de 2021). Timeline: tracing the evolution of hyperloop rail technology. *Railway Technology*. Recuperado el 11 de marzo de 2023, de https://www.railway-technology.com/features/timeline-tracing-evolution-hyperloop-rail-technology/

[16] Mitropoulos, L., Kortsari, A., Koliatos, A., Ayfantopoulou, G. (2021). The Hyperloop System and Stakeholders: A Review and Future Directions. Sustainability. 13(15), 8430. DOI: 10.3390/su13158430

[17] Musk, E. (2013). Hyperloop Alpha. SpaceX/Tesla Motors. Recuperado el 11 de marzo de 2023, de https://www.tesla.com/sites/default/files/blog_images/hyperloop-alpha.pdf

[18] Spinellis, D. (2012). Git. *IEE software*, 29(3), 100-101. Consultado de http://hdl.handle.net/10609/17885

[19] Staiano, F. (2022). Designing and Prototyping Interfaces with Figma: Learn essential UX/UI design principles by creating interactive prototypes for mobile, tablet, and desktop. Packt Publishing Ltd.

[20] Trigás Gallego, M. (junio de 2012). Metodología Scrum *Universitat Oberta de Catalunya*.

APÉNDICE A
Objetivos de Desarrollo Sostenible

El concepto hyperloop propone el desarrollo de un medio de transporte sostenible que pueda sustituir a medios de transporte tradicionales que generan unas emisiones muy elevadas. Hyperloop UPV no solo se enfoca en la creación de este medio de transporte revolucionario, sino que también trabaja activamente para abordar los desafíos globales establecidos por los Objetivos de Desarrollo Sostenible de las Naciones Unidas. Hyperloop UPV tiene una estrategia medioambiental que va más allá del desarrollo de esta tecnología: también realiza un análisis y un esfuerzo por la minimización de los posibles impactos medioambientales del proyecto, relacionados con la generación de basura, transporte de materiales, etc. El grado de relación del trabajo con los Objetivos de Desarrollo sostenible se observan en la Figura A.1.

Reflexión sobre la relación del TFG con los ODS y con los ODS más relacionados:

Algunos de los ODS en los que más influencia tiene el equipo se indican a continuación.

ODS 7. Energía asequible y no contaminante

Hyperloop se plantea como una alternativa sostenible a los medios de transporte convencionales. Puede ser utilizado a partir de fuentes de energía completamente renovables, y además se pueden implementar estas mismas fuentes de energía renovable en las propias infraestructuras para hacerlos autosuficientes, por ejemplo, a través de paneles solares a lo largo del tubo.

La implantación de este medio de transporte conlleva una eficiencia energética muy alta al evitar la fricción con cualquier superficie y disminuir prácticamente en su totalidad las pérdidas por fricción con el aire. De esta manera, no solo es sostenible, sino más eficiente y se puede plantear incluso su autosuficiencia.

ODS 9. Industria, innovación e infraestructuras

Hyperloop es uno de los conceptos más de moda en cuanto a la innovación tecnológica. Las áreas en las que su innovación es más notoria son la propulsión electromagnética, la levitación y el vacío, aunque incluye muchas otras. Estas áreas tinen influencia más allá del transporte, de forma que contribuyen al desarrollo de tecnologías de vanguardia generales. Hyperloop también contribuye con la creación de infraestructuras avanzadas y la promoción de la industrialización sostenible. Además, el desarrollo de software que propone este documento supone un primer paso para elaborar una solución innovadora para la navegación y comunicación en el transporte de pequeños vehículos.

ODS 11. Ciudades y comunidades sostenibles

Hyperloop facilita la conexión entre las ciudades y las diferentes áreas urbanas, lo que resulta en una disminución del tráfico urbano y una importante reducción de las

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ANEXO

OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	Alto	Medio	Bajo	No Procede
ODS 1. **Fin de la pobreza.**				X
ODS 2. **Hambre cero.**				X
ODS 3. **Salud y bienestar.**			X	
ODS 4. **Educación de calidad.**				X
ODS 5. **Igualdad de género.**				X
ODS 6. **Agua limpia y saneamiento.**			X	
ODS 7. **Energía asequible y no contaminante.**	X			
ODS 8. **Trabajo decente y crecimiento económico.**		X		
ODS 9. **Industria, innovación e infraestructuras.**	X			
ODS 10. **Reducción de las desigualdades.**				X
ODS 11. **Ciudades y comunidades sostenibles.**	X			
ODS 12. **Producción y consumo responsables.**		X		
ODS 13. **Acción por el clima.**	X			
ODS 14. **Vida submarina.**			X	
ODS 15. **Vida de ecosistemas terrestres.**			X	
ODS 16. **Paz, justicia e instituciones sólidas.**				X
ODS 17. **Alianzas para lograr objetivos.**	X			

Figura A.1: Objetivos de Desarrollo Sostenible
Fuente: Universidad Politécnica de Valencia

emisiones. Esto no solo mejora la conectividad entre las ciudades, sino que lo hace de una manera más sostenible y eficiente.

ODS 13. Acción por el clima

Hyperloop, al presentarse como la mejor alternativa para reemplazar los vuelos de corta y media distancia, reduce las emisiones de dióxido de carbono, es energéticamente eficiente, promueve tecnologías limpias y disminuye la huella de carbono. Además, como ya se ha comentado, el equipo se esfuerza por minimizar los posibles impactos medioambientales del proyecto. Para ello, colabora con la UPV en la gestión ambiental, reubicando los residuos no reutilizables que genera y devolviendo los elementos a las compañías que sí pueden reutilizarlos.

ODS 17. Alianzas para lograr objetivos

La European Hyperloop Week es un evento en el que los principales equipos universitarios colaboran para desarrollar un proyecto común. Se comparten avances tecnológicos y se exponen sistemas desarrollados por los equipos, lo que acelera el desarrollo del medio de transporte de manera más eficiente.