UNIVERSIDAD DE GRANADA

TRABAJO FIN DE GRADO

INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Experimentación en 5G mediante UGVs

Autor
Ángel Marín Munuera

Directores
Jorge Navarro Ortiz
Juan J. Ramos Muñoz

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, junio de 2024

Experimentación en 5G mediante UGVs

Autor
Ángel Marín Munuera

Directores
Jorge Navarro Ortiz
Juan J. Ramos Muñoz

Experimentación en 5G mediante UGVs

Ángel Marín Munuera

Palabras clave: Redes 5G, Quinta Generación, medidas experimentales, vehículo automatizado, redes móviles, UGV, AWS DeepRacer EVO

Resumen

La Quinta Generación (5G) de redes móviles ha significado una oportunidad sin precedentes para el surgimiento e implementación de diversas tecnologías avanzadas, que requieren escenarios con conexiones de alta velocidad y fiabilidad. Este proyecto se centra en la medición experimental de las redes 5G empleando UGVs, con el objetivo de determinar la viabilidad y evaluar el rendimiento de la conectividad de estas redes en diferentes escenarios utilizando estos dispositivos.

En este documento se detalla la configuración de un sistema de mediciones experimentales 5G utilizando un UGV, concretamente, un AWS DeepRacer EVO. Se han realizado múltiples pruebas de campo en diferentes entornos, tanto de interior como de exterior.

Las estadísticas de la red se recogieron y monitorizaron en tiempo real, principalmente utilizando dos plataformas de software: *Prometheus* y *Grafana*. Además, todo este proceso se automatizó mediante el uso de scripts y archivos de rutas, de forma que los experimentos sean reproducibles. Se puso a prueba la capacidad de este UGV para ser automatizado, permitiendo la realización de estos experimentos de manera eficiente.

Los resultados obtenidos demuestran la capacidad de estos vehículos para realizar experimentos automatizados, medir redes 5G y comprobar su funcionamiento. Se incluye también una pequeña guía para la ejecucion de experimentos similares a los que se han realizado, sin necesidad de conocimientos avanzados.

Finalmente, este proyecto ofrece recomendaciones para la realización de estos experimentos y sugiere líneas futuras de investigación para mejorar aún más la integración de estas tecnologías.

Project Title: Project Subtitle

First name, Family name (student)

Keywords: 5G Networks, Fifth Generation, Experimental Measurements, Automated Vehicle, Mobile Networks, UGV, AWS DeepRacer EVO

Abstract

The Fifth Generation (5G) of mobile networks has presented an unprecedented opportunity for the emergence and implementation of various advanced technologies that require scenarios with high-speed and reliable connections. This project focuses on the experimental measurement of 5G networks using UGVs, aiming to determine the feasibility and evaluate the performance of network connectivity in different scenarios using these devices.

This document details the configuration of a 5G experimental measurement system using a UGV, specifically an AWS DeepRacer EVO. Multiple field tests have been conducted in various environments, both indoors and outdoors.

Network statistics were collected and monitored in real-time, primarily using two software platforms: *Prometheus* and *Grafana*. Additionally, this entire process was automated using scripts and route files to ensure the experiments are reproducible. The capability of this UGV to be automated was tested, allowing the efficient execution of these experiments.

The results obtained demonstrate the ability of these vehicles to perform automated experiments, measure 5G networks, and verify their functionality. A small guide is also included for conducting similar experiments to those performed, without the need for advanced knowledge.

Finally, this project offers recommendations for conducting these experiments and suggests future research lines to further improve the integration of these technologies.

Yo, **Ángel Marín Munuera**, alumno de la titulación Grado de Ingenierías en Tecnologías de Telecomunicación de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 76739780G, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Ángel Marín Munuera

Granada a día 20 de junio de 2024.

D. **Jorge Navarro Ortiz**, Profesor Titular de Universidad del Área de Ingeniería Telemática del Departamento Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada.

D. **Juan J. Ramos Muñoz**, Profesor Titular de Universidad del Área de Ingeniería Telemática del Departamento Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado ***Experimentación en 5G mediante UGVs***, ha sido realizado bajo su supervisión por **Ángel Marín Munuera**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada en junio de 2024.

El director:

Jorge Navarro Ortiz

Agradecimientos

Agradecimientos a mi familia, por haber estado apoyándome durante todos estos años de forma incondicional, anteponiendo mi bienestar al suyo.

Agradecimientos a mis amigos, por todos los buenos momentos durante esta etapa. Con especial mención a los Primos, Pornais, el Grupo del Kanketilla, y Moisés.

Agradecimientos a Wendy, por su gran importancia durante la última recta de la carrera.

Y agradecimientos a Jorge Navarro, por haber dedicado una gran cantidad de esfuerzo, tiempo y paciencia en que este proyecto fuera completado.

Índice general

Índice de figuras

Índice de cuadros

Abreviaciones y notaciones utilizadas

3D Tres Dimensiones (*Three Dimensional*)

4G Cuarta Generación de tecnología móvil (*Fourth Generation*)

5G Quinta Generación de tecnología móvil (*Fifth Generation*)

API Interfaz de Programación de Aplicaciones (*Application Programming Interface*)

AWS Servicios Web de Amazon (*Amazon Web Services*)

CPU Unidad Central de Procesamiento (*Central Processing Unit*)

DNN Red Neuronal Profunda (*Deep Neural Network*)

HDMI Interfaz Multimedia de Alta Definición (*High-Definition Multimedia Interface*)

HTTP Protocolo de Transferencia de Hipertexto (*HyperText Transfer Protocol*)

IEEE Instituto de Ingenieros Eléctricos y Electrónicos (*Institute of Electrical and Electronics Engineers*)

IP Protocolo de Internet (*Internet Protocol*)

IoT Internet de las Cosas (*Internet of Things*)

JPEG Grupo Conjunto de Expertos en Fotografía (*Joint Photographic Experts Group*)

JSON Notación de Objetos JavaScript (*JavaScript Object Notation*)

KPI Indicador Clave de Rendimiento (*Key Performance Indicator*)

LED Diodo Emisor de Luz (*Light Emitting Diode*)

LiDAR Detección y Rango de Imágenes por Láser (*Light Detection and Ranging*)

LTE Evolución a Largo Plazo (*Long-Term Evolution*)

MEC Computación en el Borde de la Red (*Multi-access Edge Computing*)

MIMO Múltiple Entrada Múltiple Salida (*Multiple Input Multiple Output*)

MJPEG JPEG en Movimiento (*Motion JPEG*)

ML Aprendizaje Automático (*Machine Learning*)

MLAN Red de Área Local de Medios (*Media Local Area Network*)

MTCP Protocolo de Control de Transmisión Múltiple (*Multipath TCP*)

OpenCV Biblioteca Abierta de Visión por Computador (*Open Source Computer Vision Library*)

PPPR *Pose, Position, and Path Representation*

PromQL Lenguaje de Consultas de Prometheus (*Prometheus Query Language*)

QoS Calidad de Servicio (*Quality of Service*)

RAM Memoria de Acceso Aleatorio (*Random Access Memory*)

RF Radiofrecuencia (*Radio Frequency*)

RL Aprendizaje por Refuerzo (*Reinforcement Learning*)

RSRP Potencia de Referencia de la Señal Recibida (*Reference Signal Received Power*)

SCM Gestión de Configuración del Software (*Software Configuration Management*)

SHA Algoritmo de Hash Seguro (*Secure Hash Algorithm*)

SH Shell Seguro (*Secure Shell*)

SLA Acuerdo de Nivel de Servicio (*Service Level Agreement*)

SQL Lenguaje de Consulta Estructurada (*Structured Query Language*)

SQlite Biblioteca de Software de Gestión de Bases de Datos SQL (*SQL database management library*)

ss Estadísticas de Socket (*Socket Stats*)

TCP Protocolo de Control de Transmisión (*Transmission Control Protocol*)

TFG Trabajo de Fin de Grado

UAV Vehículo Aéreo No Tripulado (*Unmanned Aerial Vehicle*)

URL Localizador Uniforme de Recursos (*Uniform Resource Locator*)

USA Estados Unidos de América (*United States of America*)

USB Bus Universal en Serie (*Universal Serial Bus*)

Wi-Fi Fidelidad Inalámbrica (*Wireless Fidelity*)

XML Lenguaje de Marcado Extensible (*eXtensible Markup Language*)

YAML Otro Lenguaje de Marcado (*YAML Ain't Markup Language*)

YML Abreviatura de YAML

Capítulo 1

Introducción

5G es la denominación de la quinta generación de redes móviles. Presenta mejoras sustanciales en la velocidad, latencia y eficiencia [1], pudiendo hacer posible la aparición de nuevas tecnologías y revolucionar los sectores industriales. Ejemplo de ello es el *Internet de las cosas*, también conocido como *IoT*, con un tamaño de mercado valorado en 595.73 billones de dólares en 2023 [2]. Otros ejemplos son el de la *realidad aumentada* y *realidad virtual*, con un tamaño de mercado de 32.5 billones de dólares en el anterior año [3], y el de los vehículos automatizados, con un tamaño de mercado de 50.63 billones de dólares [4].

Además, la transformación digital en sectores como la salud, transporte y educación están directamente relacionados con esta tecnología. Ejemplo de ello son, en primer lugar, la *telemedicina* y la *cirugía remota*, que necesitan comunicaciones fiables, rápidas y de calidad. Por otro lado, los vehículos automatizados hacen uso de grandes cantidades de datos para mejorar la seguridad y la eficiencia del tráfico en tiempo real, comunicándose con otros vehículos y con la propia infraestructura vial. Por último, las aulas virtuales deben contar con experiencias interactivas e inmersivas, consiguiendo mayor satisfacción para los estudiantes y posibilitar mayor participación. Todo esto significa que esta generación de redes móviles no es solo en un cambio radical en la tecnología, sino un proceso de transformación de la sociedad actual.

Estas mejoras sustanciales mencionadas son resultado de hacer uso de diferentes tecnologías como las frecuencias de radio más altas del espectro electromagnético, las ondas milimétricas. Esta frecuencia presenta como dificultad principal su cobertura limitada, debido a fenómenos como la absorción atmosférica, difracción, reflexión, dispersión y atenuación. Es por ello que se varían las estrategias de tamaño de celdas seguidas (*macro*, *small*, *micro* y *pico*), con las que se pretenden crear redes densas que den una señal continua en el tiempo. Además, se hace uso de tecnologías *MIMO* y

beamforming para mejorar la eficiencia y el alcance de la señal, contando también con la implementación de frecuencias combinadas para contar con las ventajas de cada rango.

Paralelamente, los vehículos automatizados terrestres y aéreos están revolucionando diferentes sectores del mercado de forma paralela. Dotados de capacidades avanzadas de autonomía, permiten mejorar la eficiencia, reducir costos y aumentar la seguridad en diferentes campos. Algunas aplicaciones son las siguientes:

- Transporte y logística: en el sector comercial, se hace posible la entrega y transporte sin necesidad de intervención operativa, consiguiendo un menor coste logístico y de recursos humanos, así como un atractivo recorte en el tiempo de entrega para el cliente. Empresas como *Amazon* y *Nuro R2* ya cuentan con programas de entrega a domicilio con estos dispositivos. En otros sectores, estos permiten el transporte de materiales y realización de tareas peligrosas, significando además de una reducción de tiempo y aumento de la eficiencia, mejora la seguridad de los trabajadores

- Vigilancia y monitorización: equipando este tipo de vehículos con diferentes sensores, se hace posible una vigilancia y monitorización ininterrumpidos durante las 24 horas de un día. La monitorización forestal para la prevención y detección de incendios, y la vigilancia en eventos masivos sociales son ejemplos de ello.

- Operaciones de emergencia: como puede ser el caso de operaciones de búsqueda y rescate, se puede contar con estos dispositivos para intentar localizar a las víctimas en situaciones peligrosas y que necesitan una rápida actuación.

Estas dos tecnologías descritas presentan una sinergia bidireccional. Por una parte, los vehículos automatizados necesitan una conectividad, que en caso de usar las redes 5G, se tienen además las ventajas comentadas anteriormente. Por otra parte, con el uso de estos vehículos automatizados se puede brindar, por ejemplo, un monitorización y mantenimiento de la infraestructura de estas redes, una optimización en el despliegue del mismo, posibilidad de nuevas aplicaciones y servicios, e incluso se pueden utilizar como plataformas de prueba y evaluación de estas redes móviles. Esto último será el tema a estudiar en este TFG.

La medición de redes 5G se ha realizado tradicionalmente mediante el uso de equipos manuales y vehículos tripulados. Implementar el uso de los vehículos automatizado en este campo, contando con equipos de monitorización para la medición y evaluación de estas redes, resulta en diversas

ventajas. Haciendo uso de estos sensores y dispositivos, se pueden recoger parámetros que caracterizan a la red, denominados *KPIs* [5], mientras se desplazan por las diferentes áreas geográficas donde se tiene que realizar la evaluación.

De esta forma, se puede permitir la repetibilidad de los experimentos, así como una menor influencia del error humano en las mediciones, y una mayor eficiencia económica y temporal. Se tiene así una experimentación con puntos de mediciones fijos y obtención de parámetros repetitiva. Es decir, se hace posible la reproducibilidad de los experimentos. Estos experimentos de medición son realmente útiles para, por ejemplo, identificación de áreas con baja cobertura, conocer la eficiencia de la infraestructura y optimizar el despliegue. Se puede conocer también la capacidad de la red y una planificación proactiva de la misma. Además, la detección de problemas y garantización de calidad aseguran el cumplimiento de los *SLA*. Los *SLAs* son contratos formales entre proveedores de servicios y clientes que definen de forma legal los servicios que el proveedor brinda a los clientes a cambio de una suscripción [6].

Desde el punto de vista de un negocio esto puede ser visto como una oportunidad, dado que se puede brindar transparencia y fiabilidad al cliente del cumplimiento de los servicios y niveles de los mismos ofrecidos. Es decir, el cliente cuenta con una garantía de calidad por parte del proveedor dado que los datos, al ser fruto de experimentos reproducibles, son verificables. Pueden contar con informes detallados de las mediciones realizadas, además de poder tener un soporte de rápida detección de problemas y optimización. Se posibilita un mantenimiento y planificación proactiva para mantener los niveles deseados de *QoS*.

Por último, cabe señalar que cada tipo de vehículo automatizado, es decir, terrestres y aéreos, presentan sus propias ventajas y desventajas. Por una parte, la cobertura y alcance de un *UAV* es considerablemente mejor que la de un vehículo automatizado terrestre, debido a que no dependen del terreno y pueden variar la altura.

Otro factor a tener en cuenta para la experimentación de cara al cliente, es que las redes se diseñan para usuarios terrestres, por lo que un *UGV* caracterizará de una forma más similar a la que un usuario móvil experienciará la propia red. Aunque esto sea posible volando drones a muy baja altura, pierde el sentido debido a la cantidad de obstáculos frente a la fragilidad de un dron, así como la dificultad de que un dron se mantenga fijo a alturas tan bajas especialmente con la existencia de corriente.

Una vez introducidas las dos tecnologías en las que se va a basar este

TFG, se expondrán los objetivos del mismo:

- **Mediciones experimentales de 5G**: objetivo principal, se pretende poder realizar medidas experimentales de rendimiento de redes 5G de forma automatizada con un vehículo terrestre en el que se puede configurar una ruta.

- Automatización: la automatización del experimento aportará medidas fiables y precisas, sin depender del error humano. Esto aportará escalabilidad en futuros proyectos, con un aumento de la eficiencia al no depender de trabajadores como en los métodos tradicionales, y menos costes al no pagar un salario al trabajador que se encargaría típicamente de ello

- Enfocado al usuario: las mediciones estarán enfocadas en la similitud de las experiencia *QoS* de un usuario, por lo que se usará un vehículo terrestre. Adicionalmente, esto resultará en menos problemas con las interferencias entre señales y menos vulnerabilidad climatológica. Esto provocará una mayor satisfacción para el usuario, y una posible diferenciación en la competición entre proveedores en el mercado.

- Reproducibilidad: la metodología del experimento debe ser reproducible para la obtención de medidas precisas y de calidad, y repetirse todas las veces necesarias ahorrando tiempo y recursos. Se expondrá un ejemplo de ejecución para que cualquier persona que cuente con el equipo necesario pueda hacer uso de la misma, teniendo rigor en los resultados obtenidos. Esto también resulta en una aportación de información irrefrutable, basándose en la evidencia.

- Tiempo real: para una metodología aun más dinámica se asegurará que las mediciones sean visualizables a tiempo real. Se tendrá una respuesta inmediata con la posibilidad de optimizar recursos, así como gestionar el tráfico, de forma rápida. Esto es crucial en determinadas aplicaciones que exigen una respuesta inmediata, como en situaciones de emergencia [7].

- Conexión en local: para evitar retardos innecesarios, se comprometerá a no tener que usar infraestructuras externas para reducir al máximo posible los tiempos de respuesta, y que se mida fielmente la caracterización de la red.

- Bajo costo: además de los propios beneficios empresariales, el bajo costo de esta experimentación permitirá un fomento de la misma con las iteraciones deseadas, sin limitaciones de número de prototipos de medición y una posible implementación gradual.

Expuestos estos objetivos, se introduce el vehículo automatizado que ejecutará la medición de redes 5G: *AWS DeepRacer EVO*. *AWS DeepRacer* es un coche de tamaño reducido que permite la introducción en el *ML*, concretamente en el aprendizaje mediante refuerzo *RL*, con el objetivo de crear competiciones de carreras de este tipo de vehículos mediante el entreno por *RL* en entornos simulados [8]. Sin entrar en detalle en esto último, pues no será el punto de interés en este TFG, se pasa directamente a qué es *AWS DeepRacer EVO*. *AWS DeepRacer EVO* no es más que una versión mejorada de *AWS DeepRacer* que viene equipada con dos cámaras estéreo y un sensor *LiDAR* para permitir una mejor actuación de estos vehículos gracias a la adición de información del entorno.

La plataforma resultante de este proyecto se podrá ejecutar siguiendo una metodología reproducible y realizable por cualquier persona sin necesidad de poseer conocimientos técnicos mediante el uso de diferentes herramientas y tecnologías que se detallarán en las siguientes secciones. De esta forma, se demostrará la capacidad de este tipo de vehículos para contribuir a las mediciones necesarias. Se podrán demostrar así la validez de los diferentes modelos de propagación *3D*, diseños de la antena, y parámetros relacionados con la calidad del servicio. Se podrá además contar con un despliegue dinámico y proactivo, que permite cambiar la visualización en tiempo real de la misma para configuración instantánea. Además, se detallará cómo no se necesita hacer uso de la infraestructura de *AWS*, lo que permite su funcionamiento en local. De esta forma, se cumplirán los objetivos propuestos.

Para dar por concluida esta introducción, se termina detallando la estructura que sigue este TFG.

Esta primera sección, Introducción, presenta las dos tecnologías principales de este proyecto (5G y vehículos automatizados), así como sus principales características y aplicaciones además de la sinergia que presentan entre ellas. Adicionalmente, se introduce el vehículo automatizado terrestre que se usará (*AWS DeepRacer EVO*). Por último, se presentan los objetivos del proyecto y su estructura.

Seguidamente, la Sección 2 explica la Planificación temporal y Costes del proyecto. En ella, se desarrollan de forma breve las diferentes tareas realizadas, así como su representación temporal en un Diagrama de Gantt. Esta planificación fue realizada antes de realizar el proyecto, por lo que se ha seguido de una forma aproximada, pero no exacta. También se incluye un coste estimado del proyecto, teniendo en cuenta los recursos *hardware*, *software* y humanos empleados.

Después, la Sección 3 incluye el Estado del Arte. En dicha sección se

presentan los diferentes artículos analizados que tienen relación con el proyecto. Algunos de ellos consisten precisamente en la medición de redes 5G mediante el uso de vehículos automatizados, mientras otros presentan posibles aplicaciones que utilizan la sinergia entre ambos conceptos e incluso la complejidad que presenta la misma. Otras referencias simplemente abarcan el uso de vehículos automatizados. En el caso de los primeros, se señala de forma clara qué tema no se aborda de forma completa. Estos temas serán tratados con énfasis en este proyecto.

En la Sección 4, se continúa hablando de las Tecnologías Implicadas. Esta sección hace un repaso a todas las tecnologías de las que se han hecho uso en el proyecto, intentando agruparlas de la manera más coherente posible. Se incluye el vehículo utilizado, las plataformas que han servido para recoger y analizar las métricas de las redes 5G, los lenguajes de programación utilizados con diferentes propósitos, así como la propia estación base utilizada para dar la conectividad 5G en uno de los experimentos. No se entra en el detalle de cada una, y se deja su aplicación concreta para más adelante, dando simplemente una breve definición y características principales que resultarán de utilidad en este trabajo.

Se sigue el documento con la Sección 5, que contiene el Diseño e Implementación del proyecto. Comenzando con el Diseño, se expone un esquema y una breve explicación del mismo para entender con alto nivel de abstracción la arquitectura elegida para relacionar las diferentes tecnologías y desempeñar los experimentos de manera satisfactoria. Se continúa explicando la Implementación, a un nivel de abstracción mucho menor, con el objetivo de explicar paso a paso, de forma consistente, cómo se ha llevado a cabo el experimento a realizar según el diseño propuesto. Se incluye por tanto cómo se ha realizado el control del coche, cómo se han configurado las plataformas de medición, cómo se realizan las mediciones, entre otros. Se incluye también un ejemplo de ejecución de la plataforma con el objetivo de permitir al lector reproducir el experimento, incluyendo los requisitos del sistema a preparar, cómo realizar el experimento y una pequeña guía de depuración de errores enfrentados durante el desarrollo del trabajo.

Se continúa con la Sección 6, Experimentación, que explica cómo se han llevado a efecto los experimentos deseados. Se han incluido detalles sobre los dos experimentos realizados, con sus respectivos resultados y demostraciones como *links* de vídeos subidos a *Google Drive*.

Por último, en la Sección 7, Conclusiones, se reflexiona sobre la validez de los resultados obtenidos, así como del cumplimiento de los objetivos presentados anteriormente. Además, se exponen diferentes mejoras que se podrían realizar para ejecutar estos experimentos de una forma más cómoda

y eficiente, así como un planteamiento de ideas para futuros proyectos.

Capítulo 2

Planificación y Costes

En esta sección se incluirá la planificación temporal de este proyecto, que se ha seguido de forma aproximada. Además, se incluirán los costes de los elementos *software* y *hardware* para realizar una estimación de los mismos. También se añadirá al cómputo total el coste de los recursos humanos involucrados en este trabajo.

2.1. Planificación

El proyecto ha tenido una duración total de seis meses, desde enero hasta junio. A lo largo de este intervalo, se han llevado a cabo diferentes tareas que se recogen en la siguiente lista. La planificación temporal se presenta como Diagrama de Gantt en la Figura 2.1.

- Estudio de AWS DeepRacer EVO: corresponde a la labor de investigación realizada para comprender los fundamentos y el funcionamiento del vehículo automatizado utilizado, AWS DeepRacer EVO. Para ello, se hizo uso de una gran cantidad de recursos multimedia.
- Estudio de AWS: de forma casi paralela a la anterior tarea, fue necesario comprender la estructura de los servicios en la nube de Amazon para entender claramente el funcionamiento del vehículo.
- Pruebas manuales con AWS DeepRacer EVO: con el objetivo de familiarizarse con el vehículo, se hicieron diversas pruebas desde la consola virtual que posteriormente se detallará. Esto tenía el objetivo de entender el control que permite el movimiento del vehículo según lo desea el usuario. Después de esto, se indagó en la estructura de directorios que contiene AWS DeepRacer por defecto, lo que permite la comunicación con la nube de Amazon y la ejecución de los diferentes servicios.
- Investigación sobre el control del vehículo en local: dado que AWS DeepRacer EVO no presenta de forma estándar una librería de fun-

ciones que permita al desarrollador determinar rutas estáticas para el movimiento del coche sin necesidad de conexión a Internet, fue necesario investigar a través de los diferentes recursos disponibles. Dado que se abría un gran abanico de posibilidades y ningún recurso era presentado de forma oficial por el propio soporte de AWS DeepRacer, se tuvo que invertir una gran cantidad de tiempo.

- Automatización del movimiento en local: haciendo uso de lo extraído de las anteriores investigaciones, se desarrollaron diferentes archivos, detallados en futuras secciones, que permiten la automatización del movimiento en local del vehículo.
- Visualización de la cámara: también se desarrollaron pruebas para la visualización del vídeo que obtiene la cámara del vehículo.
- Experimentación: se llevaron a cabo diferentes experimentos para determinar el éxito del proyecto, en entornos interiores y exteriores. Durante esta etapa se citaron diferentes tutorías.
- Redacción de la memoria: todo el proyecto se plasmó en la redacción de esta memoria, de la forma más coherente y cohesiva posible, intentando transmitir al lector un buen entendimiento del objetivo y desarrollo del mismo.

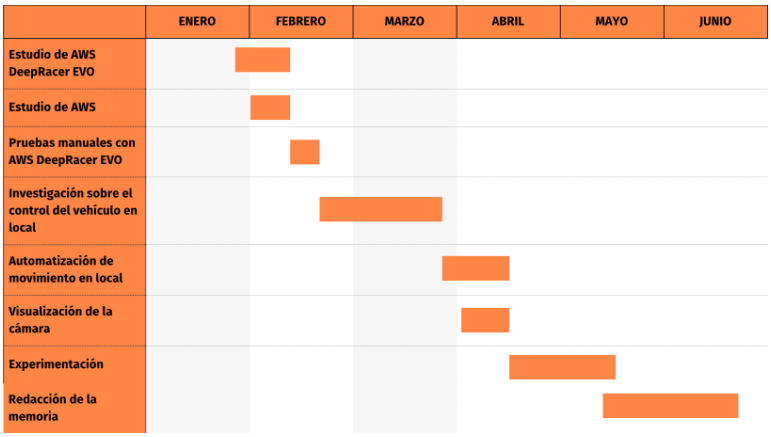

Figura 2.1: Diagrama de Gantt del Proyecto

2.2. Costes

En esta sección se detalla el coste de los diferentes recursos utilizados durante la elaboración del proyecto. Los costes se han dividido en dos categorías principales: costes de *hardware* y costes de *software*.

2.2.1. Costes *hardware*

A continuación se desglosan los costes asociados al *hardware* utilizado:

- AWS DeepRacer EVO: este dispositivo ha sido utilizado para realizar las pruebas y mediciones de las prestaciones de la red móvil 5G. El Amazon DeepRacer EVO cuesta aproximadamente 548 euros, considerando el precio de venta en dólares (598 USD) y aplicando una tasa de conversión aproximada de 1 USD = 0.92 EUR [9]. Estimando una vida útil de 3 años, como es común en la mayoría de equipos electrónicos, se imputa un costo amortizado.

$$\text{Costo amortizado 6 meses} = \frac{548\,\text{EUR}}{3\,\text{años}} \times \frac{6\,\text{meses}}{12\,\text{meses/año}} = 91.33\,\text{EUR}$$

- Teléfono móvil con 5G: utilizado para brindar conectividad a la red de datos móviles de quinta generación. En este caso, se ha usado un *Xiaomi 12*, con un precio aproximado de 799 EUR [10]. Estimando una vida útil de 3 años, el costo amortizado en 6 meses a imputar es el siguiente.

$$\text{Costo amortizado 6 meses} = \frac{799\,\text{EUR}}{3\,\text{años}} \times \frac{6\,\text{meses}}{12\,\text{meses/año}} = 133.17\,\text{EUR}$$

- PC: la estación de trabajo principal donde se han llevado a cabo las tareas de experimentación, desarrollo de *software* y visualización de resultados. Se ha hecho uso de un *ASUS ExpertCenter D7 SFF* [11], que tiene un costo de aproximadamente 747 euros.

- Monitor: se ha utilizado para visualizar los resultados de las pruebas. El monitor Xiaomi A27i tiene un precio aproximado de 249 euros [12]. Estimando una vida útil de 3 años, el costo amortizado en 6 meses a imputar es el siguiente.

$$\text{Costo amortizado 6 meses} = \frac{249\,\text{EUR}}{3\,\text{años}} \times \frac{6\,\text{meses}}{12\,\text{meses/año}} = 41.50\,\text{EUR}$$

2.2.2. Costes *software*

En esta sección se detallan los costes asociados al *software* y servicios utilizados:

- Suscripción a AWS: aunque la mayoría de los servicios de AWS no se hayan utilizado, es necesaria una suscripción a ellos para que no se bloquee la cuenta y garantizar el correcto funcionamiento de AWS DeepRacer, que incluye entrenamiento y almacenamiento de modelos. Estimando 30 horas de entrenamiento total, que incluye todo tipo de control de movimiento realizado, y 10 GB de almacenamiento, el costo es [9]:

$$\text{Entrenamiento} = 30\,\text{horas} \times 3.50\,\text{USD/hora} = 105\,\text{USD}$$
$$\text{Almacenamiento mensual} = 10\,\text{GB} \times 0.023\,\text{USD/GB-mes} = 0.23\,\text{USD}$$
$$\text{Total 6 meses} = 0.23\,\text{USD} \times 6 + 105\,\text{USD} = 106.38\,\text{USD}$$
$$\text{Total en EUR} = 106.38\,\text{USD} \times 0.92\,\text{EUR/USD} \approx 97.87\,\text{EUR}$$

- Contrato operador móvil: para garantizar la conectividad 5G del teléfono móvil, se ha necesitado un contrato con el proveedor *O2*. Una suscripción al operador *O2* con tarifa estándar en Granada, España, cuesta 30 euros mensuales [13].

- Herramientas de Ubuntu: las herramientas utilizadas en el entorno de desarrollo de Ubuntu que contiene AWS DeepRacer son de código abierto y no presentan costes adicionales.

2.2.3. Costes humanos

En este proyecto se ha involucrado a los siguientes profesionales, con los siguientes costes.

Ángel Marín Munuera, alumno de la UGR

$$\text{Tarifa} = 20\,\text{EUR/hora}$$
$$\text{Tiempo dedicado} = 12\,\text{créditos} \times 25\,\text{horas/crédito} = 300\,\text{horas}$$
$$\text{Costo total} = 300\,\text{horas} \times 20\,\text{EUR/hora} = 6000\,\text{EUR}$$

Juan José Ramos Muñoz, co-director del proyecto

$$\text{Tarifa} = 50\,\text{EUR/hora}$$
$$\text{Tiempo dedicado} = 10\,\text{horas}$$
$$\text{Costo total} = 10\,\text{horas} \times 50\,\text{EUR/hora} = 500\,\text{EUR}$$

Jorge Navarro Ortiz, director del proyecto

$$\text{Tarifa} = 50\,\text{EUR/hora}$$
$$\text{Tiempo dedicado} = 20\,\text{horas}$$
$$\text{Costo total} = 20\,\text{horas} \times 50\,\text{EUR/hora} = 1000\,\text{EUR}$$

Total de costes humanos:

$$\text{Total} = 6000\,\text{EUR} + 500\,\text{EUR} + 1000\,\text{EUR} = 7500\,\text{EUR}$$

2.2.4. Estimación del coste total

En esta sección se detalla la estimación del coste total del proyecto, incluyendo tanto los costes de *hardware* como los de *software* y los costes humanos. Como se observa en el Cuadro 2.1., el total ha sido de ocho mil seiscientos cuarenta euros con ochenta y siete céntimos.

Concepto	**Cantidad**	**Coste (EUR)**
Costes *hardware*		
AWS DeepRacer EVO	6 meses (amortización)	91.33
Teléfono móvil Xiaomi 12	6 meses (amortización)	133.17
ASUS ExpertCenter D7 SFF	6 meses (amortización)	747.00
Monitor Xiaomi A27i	6 meses (amortización)	41.50
Total *hardware*		1013.00
Costes *software*		
Suscripción a AWS	6 meses	97.87
Contrato operador móvil O2	1 mes	30.00
Herramientas de Ubuntu		0.00
Total *software*		127.87
Costes humanos		
Ángel Marín Munuera	300 horas	6000.00
Juan José Ramos Muñoz	10 horas	500.00
Jorge Navarro Ortiz	20 horas	1000.00
Total costes humanos		7500.00
	Coste total del proyecto	8640.87

Cuadro 2.1: Estimación del coste total del proyecto.

Capítulo 3

Estado del arte

Esta sección repasará los estudios actuales de medidas de redes de datos con vehículos automatizados y otra bibliografía que relaciona los conceptos de 5G y vehículos automatizados. De esta forma, se podrá tener una contextualización de la sinergia entre 5G y los vehículos automatizados, así como la identificación de vacíos, como la experimentación en la medición de estas señales con el uso de vehículos automatizados.

En el caso de los artículos que abordan de forma directa este tema, [14] presenta un método para evaluar la cobertura de redes 5G mediante el uso de *UAVs*. El análisis espacio-temporal de la señal 5G recibida sirve para verificar el modelo de las antenas, debiendo ser sistemático, automatizado y repetible. Aprovechando las ventajas de los drones, actúa como un usuario que baja recursos y sube información en diferentes puntos geográficos para verificar los comportamientos esperados o modificar la configuración y repetir estos experimentos. De esta forma, los autores muestran un diseño de sistema de medición que permite volar rutas predefinidas para realizar diferentes mediciones de los parámetros de la red, además de la fecha, hora y duración de las mismas. Se demuestra que los drones pueden realizar mediciones de forma precisa, pero existen grandes desafíos con la precisión de los sistemas de posicionamiento y la estabilidad de las mediciones debido a la naturaleza del vehículo. La experimentación se realiza, sin embargo, en 4G, concretamente en *LTE* de banda 800 MHz, con el objetivo de estudiar el efecto del *fading* en las señales mediante la medición del *RSRP* en una malla rectangular de un único *eNodeB*. Este proyecto ha contemplado las medidas de redes 5G.

En [15], se explica cómo 5G brinda la posibilidad de aumentar la seguridad pública mediante la exploración y demostración de las mejoras en comunicación y respuesta rápida en situaciones de emergencia. Con el uso de tecnologías 5G y *MEC* para procesar los datos en el lugar de la emer-

gencia, y la instalación de *IoT* para una rápida actuación, se realizan casos simulados y una experimentación utilizando *PPDRone*. Haciendo uso de este dron, una estación base para controlarlo y un *smartphone* como dispositivo de medida, se demuestra que estos artefactos, junto a las tecnologías mencionadas, están preparados para operaciones críticas. También se menciona el uso de vehículos terrestres en operaciones de gestión de desastres, búsqueda y rescate, y vigilancia. Todas estas aplicaciones necesitan una garantía de conectividad apropiada según las necesidades, que en estos ámbitos son altas por la criticidad de las mismas. He aquí una exposición, por tanto, de la importancia de las medidas de las redes 5G realizadas en este proyecto.

En [16], se exponen los obstáculos en el diseño de *UAVs* conectados a redes móviles y las innovaciones que 5G puede aportar para superarlos. Se describen los desafíos de construir un modelo tridimensional que tenga en cuenta la altura de la antena, altitud del *UAV*, diagrama de radiación y la coexistencia de estos *UAVs* con los usuarios terrestres, considerando que las redes actuales están diseñadas para usuarios terrestres. La gestión de la interferencia entre usuarios aéreos y terrestres es crucial, así como otros fenómenos como el *fading*, para los que no valen los modelos tradicionales como el de *Rayleigh*, optando por otros como *Nakagami-m*. Otros factores incluyen la necesidad de una conectividad continua y realizar *handover* de la forma más óptima posible, así como optimizar la ruta según la aplicación del *UAV* y su función de coste. Además, tener tantos sensores los expone más a diferentes ataques, aumentando el número de posibles entradas al sistema. Las redes 5G permiten hacer frente a estos desafíos. Por ejemplo, mediante una señal de *downlink* y aprendizaje por refuerzo, se miden los parámetros deseados y se dinamizan las decisiones de *handover* mediante el aprendizaje de los estados del entorno y las salidas deseadas. Se expone un caso práctico que incluye un manual de usuario para el desarrollo del experimento, cuyo objetivo es la verificación de los modelos de propagación en bandas *LTE*. Esto se realiza en entornos urbanos, suburbanos, rurales y mixtos, a diferentes alturas. Se concluye que existe un problema relacionado con la cantidad de *handovers* realizados de 5G a 4G, que empeora según la altura por las condiciones de propagación, necesitando mayores garantías de fiabilidad, latencia y velocidad; así como la necesidad de verificaciones de campo reales, no solo limitándose a medidas en casos simulados, remarcando la casi inexistencia de trabajos enfocados en campañas de mediciones. Este proyecto expone una metodología para superar estos obstáculos mencionados, así como la propia experimentación 5G que demanda. Además, el experimento se centra en la medición de redes 4G.

En [17], se expone el uso de vehículos automatizados terrestres y aéreos como una mejora significativa en la calidad y precisión de redes inalámbricas, necesarias para el despliegue de sistemas como 5G y el cumplimiento de los

SLAs. Se explica que los métodos tradicionales no son óptimos a gran escala, pues consumen demasiado tiempo y no son precisos. Se exponen dos casos prácticos, usando primero un vehículo automatizado terrestre y luego uno aéreo. Se concluye con la dificultad de un uso óptimo de dispositivos de medición sobre artefactos que no están destinados a ellos y la dificultad de volar drones a alturas bajas, así como las interferencias detectadas con señales de radio sobre las señales de control del dron. En ambos se goza de automatización; no obstante, las mediciones realizadas son en frecuencias *LTE*, además de no contar con mediciones en tiempo real. En este proyecto, las medidas realizadas serán de redes 5G, con monitorización en tiempo real. Además, los equipos con los que se han realizado las mediciones presentan un costo mucho más elevado que el necesario para este proyecto.

En el TFG [18], se expone el funcionamiento de *DeepRacer* orientado al aprendizaje por refuerzo. Tratando los diferentes tipos de vehículos automatizados y su tecnología, así como los diferentes tipos de *Machine Learning*, pasa a la explicación del funcionamiento de los servicios de la nube de *Amazon, AWS*, de cara al funcionamiento de este vehículo. Se consigue construir un algoritmo lo suficientemente bueno para que, utilizando el entorno generado de *Gazebo*, se obtengan resultados decentes en la puesta en marcha de una pista real. En otro Trabajo de Fin de Grado, [19], se tiene un objetivo similar al del anterior, pero en este caso usando un dron, que se guía por balizas visuales y un programa que determina el comportamiento del dispositivo. Estos *TFG* no profundizan en las posibles aplicaciones de estos vehículos para la medición automatizada de redes 5G. Por último, en [20], el objetivo es localizar un transmisor *RF* mediante aprendizaje por refuerzo en un dron, basándose en la aproximación de *Friis* para generar el modelo de propagación de señales. Sí se incluyen medidas experimentales, pero están enfocadas a esta aplicación concreta, no a una medición general de la quinta generación de redes móviles.

Capítulo 4

Tecnologías implicadas

Este capítulo describirá de la forma más breve posible las diferentes tecnologías utilizadas. Se entrará en detalle, con su aplicación específica en este proyecto, en el siguiente capítulo.

4.1. AWS DeepRacer Evo

Dado que es el vehículo automatizado terrestre con el que se realizan las mediciones del proyecto, se comentarán sus componentes *hardware* y *software*. Como ya se ha explicado, AWS DeepRacer es un vehículo destinado a la iniciación en el aprendizaje por refuerzo. La versión EVO presenta sensores adicionales para una mayor capacidad de navegación.

4.1.1. Hardware

Se describen a continuación los diferentes componentes de *hardware* con los que cuenta AWS DeepRacer EVO [8]:

- **Chasis**: el vehículo cuenta con un chasis propio de un coche *Monster Truck* a escala 1/18. Cuenta con una tracción a cuatro ruedas, construido con el objetivo de ser lo más ligero posible, algo fundamental en este tipo de vehículos debido a las limitaciones que presenta la batería.

- **Batería de conducción**: es la batería que permite el movimiento del coche, hecha de polímero de litio. Presenta un voltaje nominal de 7.4V y una capacidad de 1000mAh. Este tipo de baterías es común en vehículos teledirigidos.

- **Motor 390**: es un tipo de motor eléctrico en coches radioteledirigidos. Funciona con un voltaje nominal de 7.4 voltios, con un peso ligero y generalmente estable. Al funcionar con alta torsión y baja velocidad, presenta una vida útil larga con el menor nivel de ruido posible.

- **Memoria y almacenamiento**: cuenta con 4 GB de RAM y 32 GB de almacenamiento, lo cual ha presentado un factor muy limitante en el desarrollo del proyecto debido a la falta de espacio en disco y el lento funcionamiento del computador.

- **Procesador**: el procesador es un Intel Atom, de la familia de tensión ultra baja de Intel que presenta bajo gasto de energía y es fácilmente transportable, factores críticos para este tipo de vehículo.

- **Cámaras**: cuenta con dos cámaras que permiten una visión 3D para una percepción de la profundidad y así una mejor detección de obstáculos en las rutas. Cada una cuenta con una resolución de 4 megapíxeles, con formato de compresión MJPEG para una mejor codificación y decodificación de vídeo.

- **Sensores**: cuenta con diferentes sensores para una interacción con el entorno más precisa y eficaz que en el caso de su versión no mejorada. Primeramente, cuenta con un acelerómetro y un giroscopio que permiten determinar la orientación y el movimiento del vehículo en tres dimensiones mediante la medición de la aceleración y la velocidad angular.

- **LiDAR**: siglas de *Light Detection and Ranging*, es una característica de la versión mejorada que mediante pulsos de luz láser permite medir distancias con alta precisión, permitiendo detectar obstáculos de forma rápida y precisa.

- **Batería de cómputo**: es una batería de alta capacidad, 13600mAh, que permite jornadas de larga duración de forma continua. Se carga vía USB-C, estándar ampliamente adoptado.

- **Wi-Fi**: soporta 802.11ac.

- **Puertos**: presenta 4 puertos USB-A, para la inserción de cámaras, teclados, ratones, *pen-drives*, o lo que el usuario necesite. También cuenta con un USB tipo C para la conexión de la batería de cómputo o conexión al cargador de la misma, así como un puerto micro-USB útil para la primera configuración Wi-Fi del dispositivo. Por último, cuenta con un puerto HDMI para acceder a la interfaz gráfica del sistema operativo, que se continúa explicando seguidamente.

4.1.2. Software

- **Sistema operativo**: utiliza Linux Ubuntu 20.04, que es de código abierto y ha permitido la modificación de aspectos del sistema.

- **AWS**: AWS DeepRacer cuenta con Amazon Web Services. AWS es una plataforma de computación en la nube que permite utilizar diferentes servicios de almacenamiento, bases de datos, redes, movilidad, desarrollo, entre otros. Esto permite a sus clientes obtener un servicio bajo demanda de una serie de herramientas sin necesidad de invertir en el *hardware* [21]. Aunque DeepRacer hace uso de varios servicios, se expondrán solo los de interés en este proyecto, pues al limitar el uso de DeepRacer a la replicación del control manual mediante *scripts* sin recolectar la información de los sensores con los que cuenta, la mayoría no están implicados en este TFG. No obstante, sí que se hace uso de las cámaras, pero no están relacionadas con el comportamiento de la conducción del vehículo, sino para simplemente visualizar el *stream* de vídeo mientras el coche se desplaza.

  - **Amazon S3**: su nombre proviene de las siglas de *Simple Storage Service*. Como su nombre indica, permite el almacenamiento de diferentes datos, aunque en este caso serán las configuraciones del vehículo, como la calibración y el color de la luz LED que posee en la parte trasera. También permite almacenar registros de control manual en caso de fallo o excepciones.
  - **Amazon CloudWatch**: es un servicio de monitorización y gestión de *logs*, que permitirá la detección de problemas en tiempo real, como consumo de CPU, que permite asegurar un rendimiento adecuado del coche. En caso de existir fallos durante la ejecución de rutas, se puede obtener información detallada.
  - **AWS IoT Core**: permite la interacción entre la consola y el vehículo de una forma segura. Permite el envío de datos de telemetría, como el nivel de batería restante, y los comandos de control de movimiento del vehículo, con los parámetros especificados en los diferentes *scripts* de movimiento.
  - **Amazon Kinesis Video Streams**: este servicio permite la transmisión y visualización de *streams* de video en tiempo real desde la cámara del AWS DeepRacer EVO.

4.1.3. ROS

Robot Operating System [22] es un marco de trabajo enfocado a la creación de aplicaciones de robótica mediante la utilización de diferentes herramientas y bibliotecas. Cuenta con una arquitectura modular caracterizada por los nodos, que son los componentes que realizan tareas específicas; los *topics*, que son canales de comunicación entre nodos; y servicios, para el procesamiento y recepción de respuestas entre ellos. Aplicado al caso limitado en el que nos encontramos, ROS permite un sistema de mensajería

ROS *kinetic* entre los diferentes nodos de DeepRacer para intercambiar información. Es el responsable, por tanto, de traducir los comandos de control ejecutados por las funciones de los *scripts* de las rutas en señales para los actuadores. También permite la escalabilidad para la inserción de diferentes sensores. En conclusión, funciona como *software* de navegación y control.

4.1.4. Consolas y APIs

AWS presenta diferentes *APIs* que permiten el control del coche de la forma deseada, como la consola. Esta consola es una interfaz web, en el caso de interés de forma local, que permite tanto manejar el coche de forma autónoma o manual mediante un *joystick* virtual como la configuración de SSH para la conexión remota, o incluso el color del LED con el que cuenta en la parte trasera del chasis.

4.2. Plataformas destinadas a las métricas 5G

4.2.1. Prometheus

Prometheus [23] es una plataforma de monitorización y alerta de código abierto, especializada en las métricas en series temporales. Estas métricas pueden ser recopiladas de un amplio abanico de fuentes, incluyendo aplicaciones, sistemas operativos y servicios de la nube. Cuenta con su propio lenguaje de consulta, *PromQL*, para la recuperación y análisis de los datos almacenados. La herramienta *Node Exporter* permite monitorear métricas que almacena Prometheus, y será usada también en este proyecto.

4.2.2. Grafana

Grafana [24] es una plataforma de análisis y visualización de código abierto, mediante la creación de *dashboards* personalizados que contienen la información deseada. Cuenta con una gran flexibilidad para la integración con múltiples fuentes de datos, entre ellas, las generadas con *Prometheus*. Los paneles, *dashboards*, son interfaces visuales que muestran gráficos en paneles, que se refrescan según el periodo establecido. Es un elemento con gran atractivo visual por su simplicidad y limpieza, además de la versatilidad que presentan al ser personalizables, como se verá más adelante. Los *datasources*, por su parte, son las fuentes de datos a visualizar, que en este caso estarán enlazados con *Prometheus* [23]. Utiliza bases de datos tipo SQLite por defecto. Estas bases de datos contienen la información a visualizar, y este tipo permite una gran simplicidad y es ligera, algo importante debido a las limitaciones de cómputo que presenta el *software* de este vehículo. Además, no requiere de configuración adicional, ni un servidor dedicado a la misma.

Se almacena todo contenido en un solo archivo, facilitando la configuración, mantenimiento y exportación.

4.3. Lenguajes

4.3.1. Python

Python [25] es un lenguaje de programación de alto nivel con gran popularidad en la actualidad debido a su simplicidad y legibilidad. Soporta diferentes paradigmas de programación, aunque el más común es el orientado a objetos, pues permite una sintaxis clara y entendible. Además, con el objetivo de aumentar la claridad y legibilidad, su funcionamiento se basa en la indentación, sin necesidad de usar los puntos y comas presentes en otros lenguajes de programación populares. Además, cuenta con múltiples bibliotecas estándar que permiten al desarrollador trabajar de una forma más eficiente. Cabe destacar su interoperabilidad y capacidad multiplataforma. Puede ejecutarse en todos los principales sistemas operativos, y se puede utilizar junto a otros lenguajes de programación como C. En este trabajo, el uso de *Python* estará enfocado a la automatización y creación de *scripts*.

4.3.2. JSON

JSON [26] es un subconjunto del lenguaje de programación *JavaScript*, usado como alternativa al lenguaje de marcado XML, caracterizado por su simplicidad y eficiencia. Presenta una sintaxis clara y simple basada en pares clave-valor y listas ordenadas y estructuradas mediante signos de notación como llaves, corchetes, puntos dobles, comas y comillas. Permite crear diferentes datos como *arrays* y booleanos. Esto permite una gran legibilidad permitiendo una fácil comprensión, edición y depuración. En este trabajo, *JSON* servirá principalmente para la automatización de rutas.

4.3.3. Shell

Shell es un lenguaje interpretado propio de sistemas operativos Linux que se basan en archivos de texto con comandos ejecutables por el *Shell* del sistema, incluyendo *bash* y *sh*, entre otros. Su objetivo en este proyecto será la automatización de procesos y despliegue de *software*.

4.3.4. YAML

YAML es un lenguaje de serialización de datos legibles, que cuenta con una estructura también basada en la indentación, de forma que se tiene una sintaxis clara y legible como ocurría con el lenguaje de programación *Python*. Puede presentar diferentes tipos de datos y comentarios, y su uso

más extendido es para la creación de ficheros de configuración, como es el caso de este proyecto.

4.4. Otras

4.4.1. Git

Git [27] es un sistema de control de versiones distribuido creado originalmente para el desarrollo del kernel de Linux. Cada desarrollador tiene una copia completa del repositorio para proporcionar flexibilidad y seguridad. Facilita la integración y trabajo simultáneo gracias a la creación y gestión de ramas para tener un entorno aislado. Además, cada archivo se protege con un hash SHA-1 que da integridad al historial de cambios. En este proyecto, junto a la plataforma *GitHub* [28], donde se encuentran los repositorios utilizados, ha sido fundamental el uso de esta tecnología para aprovechar funciones ya existentes en el control manual del coche, así como del propio repositorio del tutor Jorge Navarro para la medición de la red 5G.

4.4.2. USB-tethering

Para realizar las mediciones experimentales de la red de datos 5G, se ha optado por hacer uso de *USB tethering*. El *USB tethering* consiste en compartir la conexión a Internet de, en este caso, un dispositivo móvil, a otro equipo. Para ello, se conectan ambos con un cable USB, en este caso de tipo C. De esta forma, el teléfono móvil funciona como un punto de acceso a Internet. De esta forma, el vehículo automatizado siempre dispondrá de una conexión a la red 5G estable, sin limitaciones de naturaleza inalámbrica, aun mientras se mueve.

Capítulo 5

Diseño e implementación

5.1. Diseño

Se comenzará la sección explicando con gran nivel de abstracción el diseño propuesto para la realización de las mediciones, de forma que se cumplan los objetivos propuestos en la Sección 1. Este se muestre en la Figura 4.1.

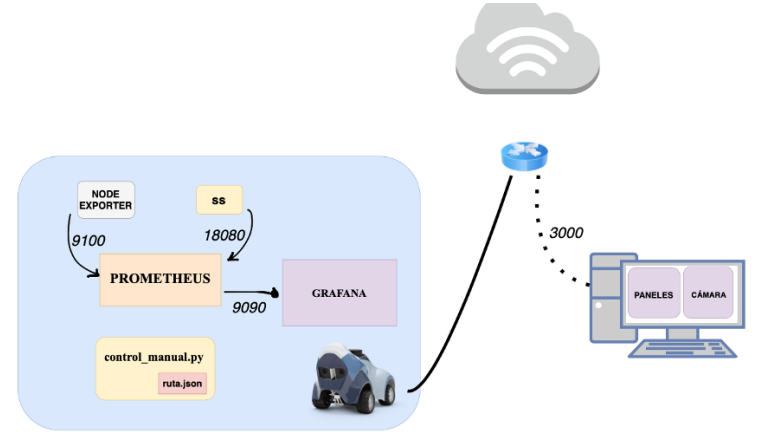

Figura 5.1: Diseño propuesto para el proyecto

Se medirán las estadísticas de red recogidas en una interfaz de red del coche. Esta interfaz de red estará conectada a un punto de acceso, que da conexión con la *red 5G a medir experimentalmente*.

Para esta medición, se hará uso de dos herramientas de *Prometheus*, tecnología que, como ya se ha explicado en la Sección 4, permite la recolección de este tipo de datos. Estas dos herramientas son *Node Exporter* y *ss*, y ambas enviarán a la plataforma principal las métricas recolectadas en cada caso por diferentes puertos *TCP*. Esta recolección estará *automatizada*,

iniciando y deteniendo el proceso mediante el uso de *scripts*.

Estas métricas se publicarán en otro puerto *TCP*, y serán utilizadas por *Grafana* como la información a procesar y representar. La visualización de esta representación se podrá realizar desde un equipo monitor que esté conectado al mismo punto de acceso, de forma que todo se haga en *local*. Esta visualización estará disponible conectándose por un navegador web al puerto *TCP* de *Grafana* encomendado para ello, pudiendo ver las estadísticas a *tiempo real*.

Además, se ha hecho uso de *OpenCV* para que, mientras el coche sigue su ruta y se observan las mediciones realizadas, se visualice además qué es lo que está viendo el coche. Es decir, ver la localización del coche haciendo uso del *stream* de vídeo capturado por la cámara.

Se ha optado por el desarrollo de un *script* que permita la *automatización* del proceso de movimiento. Mediante el diseño de un archivo de ruta, que se le pasa a este *script*, el coche realizará determinadas acciones. Este archivo de control de movimiento se basa en el control de flujo mostrado en la figura 4.2.

Sin entrar en detalle, como se observa, según el contenido del archivo de ruta el coche se arrancará, moverá o detendrá, mientras realiza o no otros procesos para las mediciones. Cabe destacar que estos procesos se realizan con hebras, cuya implementación se explicará más adelante. De esta forma se podrá efectuar el movimiento y la recolección de estadísticas al mismo tiempo.

El archivo de ruta es legible y fácil de entender, y en la subsección 5.5. se explica detalladamente su estructura. Según los cambios realizados, el coche se comportará de determinada forma, y como cabe esperar, con el mismo archivo de ruta, el coche tendrá el mismo comportamiento. Esto otorga *reproducibilidad* a los experimentos. Estos experimentos, al medir un entorno con baja altura, están *enfocados al usuario*.

5.1.1. Archivo de ruta

Se pasa a la explicación del archivo de ruta que se le pasa al *script* de control mencionado (véase el archivo ejemplo de ruta en Apéndice I). Dado que es un archivo que se puede presentar en diferentes formatos, y cada usuario puede reflejar su criterio de qué herramientas utilizar para medir correctamente la red según el objetivo, y cómo se debe mover el coche según el entorno, se ha incluido en esta sección de Diseño. Es independiente de la implementación.

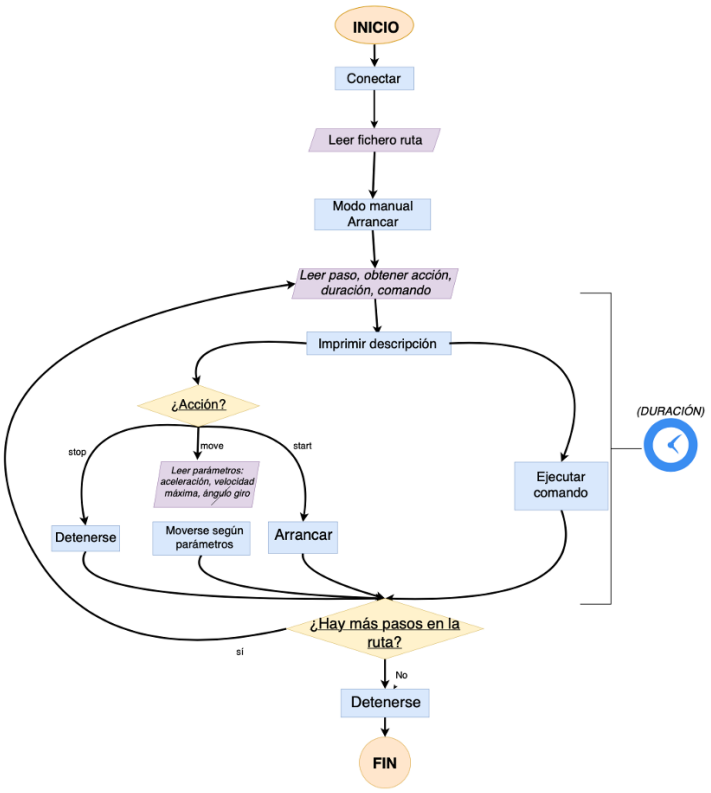

Figura 5.2: Control de flujo del archivo de movimiento

El archivo de ruta diseñado en este proyecto es un archivo `json` que contempla una serie de pasos, denominados `steps`, contenidos en un *array*. Cada paso contiene un campo descripción denominado `description`, la acción a realizar, que servirá como control de flujo en el *script* de control, llamada `action`; la duración del paso en cuestión, denominada `duration`; un mensaje informativo, y el comando a ejecutar. En el caso de que la acción sea `move`, se recogen en `params` los parámetros que necesita la función a ejecutar: `steering_angle`, `throttle` y `max_speed`. Estos tres corresponden al ángulo de giro, aceleración y velocidad máxima explicados previamente.

Además, en el campo `command` se definen los comandos que el método `run_command_thread(command)` del *script* de control ejecutará mediante hebras al mismo tiempo que se mueve el coche. Este método se detalla posteriormente. *command* puede ser cualquier comando que se pueda ejecutar en la línea de comandos, lo que proporciona mucha flexibilidad para incluir nuevos comandos en el futuro. En el caso del archivo de ruta diseñado, se tienen las siguientes herramientas:

- *Ping*: es una herramienta que sirve para verificar la conectividad entre

dos nodos con paquetes *ICMP*, que muestra en la terminal la longitud de los paquetes y su tiempo de ida y vuelta. De esta forma, también se puede comprobar la latencia de la red. En el caso de `ping -c 4 192.168.222.1 | tee -a ping.log` envía 4 paquetes de solicitud de eco a la dirección *IP* `192.168.222.1`.

- `ping`: ejecuta la herramienta *ping*.
- `-c 4`: esta opción le indica que se envíen 4 paquetes de solicitud de eco. El número 4 puede ser reemplazado por cualquier otro número para enviar una cantidad diferente de paquetes.
- `192.168.222.1`: es la dirección *IP* del dispositivo de destino al que se están enviando los paquetes de *ping*.

- *Iperf*: es una herramienta para medir el rendimiento de la red, necesitando un cliente y un servidor. Muestra el ancho de banda máximo alcanzable en una red. En el *script* de ruta, se ejecuta `iperf3 -c 192.168.222.1 -t 10`, que ejecuta una prueba de rendimiento de la red en modo un cliente a un servidor en la dirección *IP* `192.168.222.1` durante 10 segundos.

  - `iperf3`: ejecuta la herramienta en cuestión.
  - `-c 192.168.222.1`: esta opción especifica el modo cliente y la dirección *IP* del servidor al que se conecta.
  - `-t 10`: esta opción especifica la duración de la prueba en segundos. En este caso, la prueba de rendimiento de la red se ejecutará durante 10 segundos.

Los *logs* de cada uno se almacenarán en `ping.log` y `iperf.log`, respectivamente. Para ello, se incluye `| tee -a ping.log` tras los comandos anteriores (`iperf.log` en su caso).

- `|`: operador de tubería que se utiliza para encadenar comandos en la línea de comandos de *Unix/Linux*. Toma la salida del comando a su izquierda y la utiliza como entrada para el comando a su derecha.

- `tee`: comando de *Unix* que lee desde la entrada estándar y escribe en la salida estándar y en uno o más archivos.

- `-a`: indica que se agregue la salida al final del archivo especificado, en lugar de sobrescribirlo.

- `ping.log`: nombre del archivo en el que se almacenará la salida del comando *ping*.

5.2. Implementación del control del coche

En primera instancia, el control del coche será una replicación del control manual que ofrece la consola, como ya se ha comentado. Pero dado que este experimento debe ser automatizado, pues es uno de sus objetivos clave, este control manual estará automatizado de la forma descrita a continuación.

Se hará uso del repositorio [29]. El repositorio en cuestión es una edición de [30], cuyo funcionamiento se explica en [31]. Básicamente, mediante la herramienta 'Inspeccionar elemento', cuando se utiliza la consola se pueden observar tramas de solicitudes web a la *API*.

Analizándolas, el movimiento del coche al mover el *joystick* virtual se basa en la ejecución de diferentes funciones. Las funciones de este repositorio, por tanto, hacen uso de la estructura de la solicitud de la *API* utilizada para mover el coche y lo mandan a la web con métodos *PUT*, replicando las funciones que se ejecutan al mover el coche con el *joystick* virtual. A modo de aclaración, un método *PUT* es un método estándar de *HTTP* que sirve para actualizar los recursos especificados en el mismo.

Una vez entendido el funcionamiento básico, se pasa a entrar en algo más de detalle. Bien es cierto que el objetivo de los autores es uno totalmente diferente al de este proyecto, por lo que las funciones utilizadas son solo cuatro:

- `set_manual_mode()`: este método cambia al modo manual del coche, que será replicado por las siguientes funciones. El contenido de la solicitud se muestra en la Figura 4.3.
- `start_car()`: este método arranca el coche. El contenido de la solicitud se muestra en la Figura 4.4.

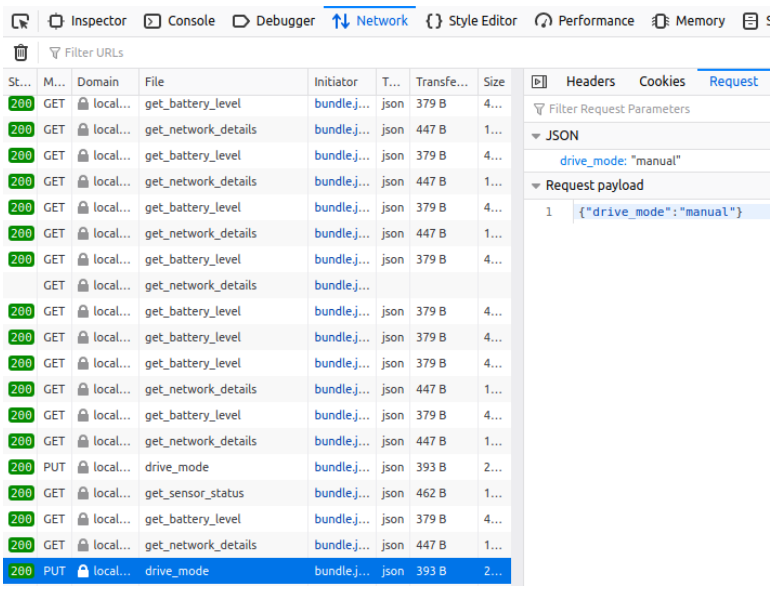

Figura 5.3: Payload de la trama solicitud a la API cuando se ejecuta set_manual_mode().

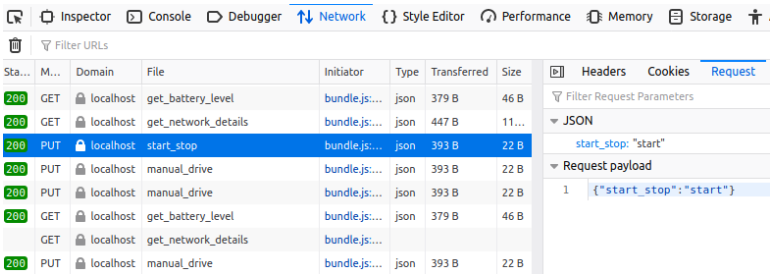

Figura 5.4: Payload de la trama solicitud a la API cuando se ejecuta start_car().

- `move(angle, throttle, max_speed)`: este método sirve para mover el coche, pasando una velocidad relativa a una velocidad máxima, un ángulo de giro y una aceleración. Con 'relativa' se quiere decir que, por ejemplo, una velocidad máxima de 0.6 y una aceleración de 1.0 dará como resultado una velocidad final de 0.6. Si la aceleración fuera de 0.5, el resultado sería de 0.3. Es decir, el 1 de la aceleración se refiere al valor de velocidad máximo alcanzable, y 0 para el mínimo, de forma directamente proporcional. Todos estos valores van de 0 a 1, excepto la aceleración, que va de -1 a 1 (los valores negativos representan el sentido marcha atrás). El contenido de la solicitud se muestra en la Figura 4.5.

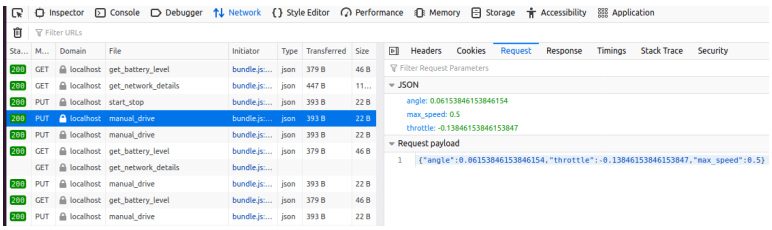

Figura 5.5: Payload de la trama solicitud a la API cuando se ejecuta move(angle, throttle, max_speed).

- `stop_car()`: este método detiene el coche. La solicitud se muestra en la Figura 4.6.

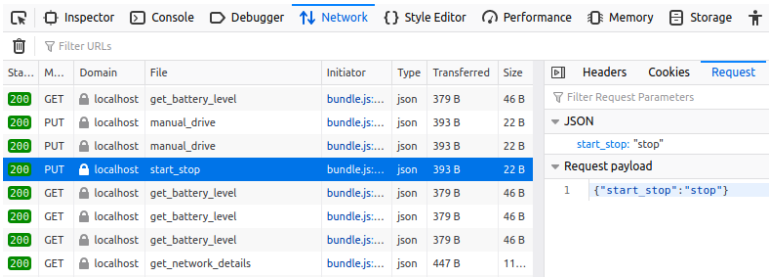

Figura 5.6: Payload de la trama solicitud a la API cuando se ejecuta stop_car().

Una vez identificados los métodos a usar, se puede crear un *script control_manual.py* de *Python* que contenga el control a seguir, según una ruta que establezca una serie de pasos a especificar por el experimentador. Véase el *script Python* de control en Apéndice H.

En primer lugar, será necesario crear una clase cliente, que interactúa con la consola web que ya se ha descrito. Es necesario especificar la contraseña que se utiliza para iniciar sesión en la misma, y la *IP* del coche con el que se realiza el experimento para acceder, aunque como se indicó en los objetivos, es local. Por esto, se especifica como `localhost`. Cabe mencionar que esto no es algo obligatorio, sino una implementación que permite no depender de la red o de la arquitectura de *AWS*, pero se puede hacer mediante *SSH* si se desea. Además, hacerlo en local permite incluir todos los paquetes y herramientas necesarios en el coche, sin tener que configurarlos además en el *PC* con el que se van a ejecutar estos experimentos.

Lo siguiente será crear un método que ejecute los comandos deseados para las mediciones, que se especificarán en el archivo de ruta diseñado en cada

caso. Es decir, ejecuta comandos de forma genérica, dependiendo el comando del archivo de ruta seleccionado. De esta forma, el experimentador podrá ejecutar los comandos que él considere oportunos para estas mediciones. Los resultados obtenidos por estos comandos, para su completo entendimiento, se mostrarán en la Sección 6.

Continuando con la explicación del *script*, el método `run_command()` ejecutará estos comandos haciendo uso de la librería `subprocess`. Será necesario pasarle el comando a ejecutar, y se imprimirá en la salida estándar el resultado del mismo, ya sea correcto o no.

El siguiente método creado, `run_command_thread(command)`, consiste en aislar este comando a ejecutar con el anterior método mediante el uso de hebras. El uso de hebras permitirá la ejecución de comandos simultáneos sin necesidad de esperar a que uno termine para poder comenzar el otro. Esto será realmente útil cuando se quieran realizar mediciones, es decir, ejecutar comandos como `iperf` o `ping`, mientras el coche está en movimiento. En caso de no usar hebras, las mediciones están limitadas al caso en el que el coche no está ejecutando ningún método `move`, pues se impediría el comienzo de estos comandos hasta que se acabe la duración de este movimiento.

En el método `main` se pasa como parámetro el archivo de ruta diseñado, mediante la librería `json` y el método de carga del mismo `load`. Es importante que el archivo de ruta se encuentre en el mismo directorio que este *script* de control.

A partir de aquí, el archivo de ruta determinará el flujo de este *script* de control. El archivo de *Python* recoge la acción a realizar (arrancar, detenerse, moverse) y lo recoge en `action`. Los parámetros a pasarle a la función de movimiento correspondiente se recogen en `params`, y se muestra un mensaje `message`, que es una cadena de caracteres que indica qué comando se está ejecutando. Cada `step` viene encabezado por una `description`, que describe lo que está sucediendo en ese momento del experimento. Por ejemplo, puede indicar que se ha llegado a la primera parada, o que el coche se mueve en línea recta.

Se continúa diferenciando las acciones a realizar según la `action` a ejecutar. En primer lugar, en caso de `move`, se utiliza el método `move` de la clase `client` al que se le pasan los parámetros de ángulo de giro, aceleración y velocidad máxima. Además, se establece una duración durante la cual se ejecutará este comando, recogido en el campo `duration` del `json`, y se pasa a un `sleep` de la librería `time`. Este `sleep` es importante para que no se ejecuten de forma continua, seguida y descontrolada el archivo de ruta. Este `sleep` recoge como atributo el valor de la `duration` especificada. Además,

se ejecuta en la hebra el comando deseado para las mediciones y se imprime con `print` en la salida estándar el resultado del mismo.

En segundo lugar, se tiene la `action stop`. Esta acción consiste en la parada del coche durante la `duration` especificada con la función `stop_car()` del repositorio y la ejecución en una hebra del comando con el que se quiere medir, imprimiendo una vez más el resultado en la salida estándar. Una vez finaliza la parada, se vuelve a arrancar el coche con `start_car()` para continuar con la ruta. Por último, si la acción especificada es `start`, simplemente se arranca el coche con la función del repositorio `start_car()`. Para terminar el *script*, se detiene el coche definitivamente con `stop_car()`.

El bucle de movimiento y ejecución de comandos continúa hasta que se completen todos los pasos.

5.3. Implementación de las plataformas de medición

Se continúa explicando la implementación de las mediciones 5G. Para ello se hará uso del repositorio [32] de Jorge Navarro, cuyo objetivo original es la medición de estadísticas de la red durante el uso de *MPTCP*, una extensión de *TCP*, usando una máquina virtual. Para una explicación ordenada de cómo se implementa en este proyecto y su utilidad en el mismo, se seguirá el proceso de instalación contenido en el archivo `README.md` (véase el archivo con las instrucciones de inicialización en Apéndice A).

En primer lugar, se crea el directorio `~/vagrant/stats` donde se descarga el paquete de *Grafana* versión 8.0.1, una plataforma de visualización de series temporales que se había explicado anteriormente. Una vez instalada, se descarga *Prometheus* v2.27.1, se descomprime y se instala la librería `prometheus_client` con `pip3`, que será el cliente de *Prometheus*. Lo siguiente será descargar e instalar la herramienta *Node Exporter* v1.1.2 de *Prometheus*.

Una vez preparadas las plataformas de mediciones, se clona el repositorio en el mismo directorio. Se detiene *Grafana* para realizar los cambios oportunos, y se copia tanto el archivo de configuración de *Prometheus* como el archivo de configuración de *Grafana*.

Una vez hecho esto, se continúa dando permisos y ejecutando los *scripts* de importación, que importan las fuentes de datos y *dashboards* a visualizar en *Grafana*. Se detiene una vez más este servicio para guardar los cambios.

Se comentan ahora las configuraciones realizadas en ambas plataformas, así como los *scripts* `Shell` de importación.

5.3.1. Configuración de Prometheus

Se explica a continuación la configuración de *Prometheus* que se realiza mediante la copia del archivo `prometheus.yml` (véase el archivo de configuración original de *Prometheus* en Apéndice B) contenido en el repositorio. Este archivo dicta cómo se recolectan las métricas, proceso denominado *scraping*, en formato *YML*, sin definir ninguna alerta de forma predeterminada. Respecto al proceso de *scraping* se define el intervalo de medición a 1 segundo, y posteriormente se establece la configuración para recolectar métricas de *Prometheus* en el puerto 9090 local, y las de *Node Exporter* en el puerto local 9100. Además, recolecta las del servicio `ss` en el puerto 8080. Este puerto se debe modificar, pues coincide con el servicio *ROS* que contiene *AWS DeepRacer*, por lo que se debe establecer a otro puerto.

Como aclaración, `ss` es una herramienta de *Linux* que permite recopilar estadísticas de *sockets*, permitiendo así recolectar información detallada sobre cada conexión. Esta herramienta será procesada en el archivo `ss_exporter.py` que se explicará de forma más detallada a continuación.

5.3.2. Configuración de Grafana

Se explica ahora la configuración de *Grafana* que se realiza mediante la copia del archivo `grafana.ini` (véase el archivo de configuración de *Grafana* original en Apéndice C). Este archivo dicta cómo se ejecutará *Grafana* para cumplir con la configuración que se describe en las siguientes líneas.

Sin entrar en demasiado detalle, el documento se divide en diferentes secciones encabezadas por su nombre de sección. Las líneas que comienzan por almohadilla o punto y coma están comentadas. Esto quiere decir que casi la totalidad de las secciones están a valores predeterminados, pues casi todo el archivo está comentado, y esta configuración predeterminada se establece después del `[nombre_sección]` de cada sección. A modo de ejemplo, la sección de base de datos establece el tipo de base de datos, la dirección para acceder a ella, y su nombre, entre otros.

La única línea que implica una modificación respecto al archivo de configuración original es la siguiente:

```
[dashboards]_min_refresh_interval = 1s
```

Esta línea, en la sección de `Dashboards`, establece que el intervalo mínimo para realizar una actualización del `dashboard` es de 1 segundo. De esta

forma, se pueden apreciar los cambios de una manera continuada, sin saturar el servidor con las solicitudes de `refresh`, que puede ser un factor limitante a la hora de realizar los experimentos. Esto se debe a que el `refresh` es una actualización que requiere de gran cantidad de *CPU* y memoria, además de tráfico en la red. Si se satura la red con estas solicitudes, el experimento estará consecuentemente ligado a unos malos resultados en la medición de los parámetros. Sin embargo, si se estableciese un `refresh` mínimo mayor, el usuario no podría ver la medición de una forma tan interactiva y en tiempo real, que es uno de los objetivos del proyecto.

5.3.3. Importación de *dashboards* y *datasources*

Los *scripts* `import_datasources.sh` e `import_dashboards.sh` (véase el *script bash* de importación de fuentes de datos (*datasources*) en Apéndice D y el de paneles en Apéndice E sirven para establecer la configuración de `datasources` y `dashboards`.

En el primer archivo, se establece la *URL* de la *API* de *Grafana* para la creación de los mismos, especificando que el contenido está en formato `json`, así como las credenciales de acceso a la misma. Se recorrerán todos los archivos del directorio `data_sources` y se publicarán en la *URL* especificada, que se encuentra en el puerto 3000.

En el segundo archivo, `import_dashboards.sh`, se importa la configuración de los `dashboards` que permitirán la representación de las métricas recogidas, así como su visualización en paneles. El *dashboard* importado se configura en `MPTCPstatisticsUGR5GNR+Wi-Fi6+LIFI-1683270090539.json`. Está estructurado en diferentes secciones, siguiendo un formato `JSON`. En `panels` se definen las medidas a representar, concretamente en el *array* `targets`. Habiendo un `target` para cada panel, la línea de `expr` indica la métrica a mostrar. En la mostrada, se observa que se hace uso de la función de `rate` de *Prometheus*, que mide la tasa de bits recibidos por segundo. Esto es posible ya que se ha exportado como *datasource* el archivo contenido en el repositorio. En este archivo, se selecciona *Prometheus* como tipo de fuente de datos, y la *URL* donde se recopilan las métricas a mostrar en *Grafana*, en este caso, `10.0.2.15:9090`, puerto que se indicó en el `target` de `prometheus.yml`. Dado que se quiere mostrar en *bytes*, se pasa a esa unidad multiplicando por su factor de conversión, 8. Sigue la sintaxis `rate(vector[Interval])`. En este caso, el vector es `node_network_device_bytes_total`, que es el nombre de la métrica en *Prometheus* que cuenta el número total de *bytes* recibidos por la interfaz de red que se está monitoreando. Esta interfaz se identifica con la etiqueta `device`, pudiendo tener los tres valores presentes que se utilizaron en el proyecto original. De esta forma, se filtra el tráfico de la red al de las interfaces de interés. `[3s]`

indica que se calcula la tasa cada 3 segundos. Es importante destacar que el operador `=~` sirve para que el nombre de las interfaces sea aproximadamente igual a los indicados, permitiendo versatilidad y escalabilidad, algo de lo que se hará uso en nuestro proyecto.

5.4. Realización de las mediciones

Para realizar las mediciones, será necesario iniciarlas con `start_stats.sh` (véase *script* en *bash* de la inicialización de servicios de recolección y monitorización de métricas en Apéndice F).

En primera instancia, inicia el servicio de *Grafana*, y seguidamente las plataformas de recolección de métricas que se habían indicado anteriormente. Estas son *Node exporter*, *Prometheus*, y *SS Exporter*. Aunque `ss` no es una plataforma como tal, se ha hecho uso de un *script* denominado `ss_exporter.py` (véase *script Python* de exportación de métricas desde *ss* a *Prometheus* en Apéndice G). En el mismo se definen diferentes métricas que después se exportan a *Prometheus* para que las recoja. Estas métricas son de tipo *Gauge*, es decir, valores numéricos individuales que pueden aumentar y disminuir, comúnmente usados en métodos de mediciones [33]. Además, se inicia un servidor en el puerto 8080 en el que *Prometheus* recolectará las métricas.

Dentro del bucle principal, se ejecuta `ss -i -t`, y se lee la salida línea por línea que se actualiza constantemente y que va al servidor donde *Prometheus* las recoge. Para ello, siguiendo la salida mostrada como ejemplo, se asocia a cada métrica un origen y un destino, y se establece el valor usando métodos de la librería `prom` de *Prometheus*, que son `set`, `inc` y `dec`. De esta forma se consigue un monitoreo eficiente a tiempo real. Una vez iniciada la monitorización de estadísticas y abierto *Grafana* en el navegador, se comienza la ruta con el vehículo mediante `python3 ./control_manual.py -r ruta.json`, cuyos parámetros ya se han explicado.

Una vez finalizada, se ejecutará `stop_stats()` para detener las mediciones. Este *script* se basa en la búsqueda de los procesos activos `grafana-server`, `node_exporter` y `prometheus`, y si se encuentran, se detienen. Por último, se detiene `ss_exporter`, filtrando en este caso por el puerto en el que se está ejecutando, 8080.

Como recopilación de lo explicado anteriormente, el repositorio hace uso de las herramientas de *Grafana* y *Prometheus*. Una vez se instalan, el archivo `grafana.ini` configura la base de *Grafana*, y se gestiona importando los *datasources* y el `dashboard` a utilizar con `import_dashboard.sh` e

`import_datasources.sh`. *Grafana* visualizará los contenidos que se recojan con *Prometheus*. Esta plataforma se configurará con `prometheus.yml`, y se le exportarán métricas *TCP* con `ss_exporter.py` usando la herramienta `ss`. Por último, para iniciar y detener las mediciones, se hace uso de `start_stats.sh` y `stop_stats.sh`. Mientras se realizan las mediciones, el coche estará en movimiento con el archivo de ruta seleccionado.

5.5. Ejecución de la plataforma

Este apartado explicará los pasos a seguir para la reproducibilidad del experimento, otro de los objetivos clave de este proyecto.

5.5.1. Preparación de entorno

Para llevar a cabo las rutas, será necesario seguir el apartado *Installation*, usando `pip3` en vez de `pip`, de [29], bastando con introducir la siguiente línea en la terminal:

```
pip3 install awsdeepracer_control
```

El resto de lo relativo al control del coche lo realiza el *script* de ruta incluido en la Sección Apéndices, véase ejemplo de ruta en Apéndice I. Se deberá cambiar el inicio de sesión en la consola, al crear la instancia de la clase `client`, sustituyendo el valor de `password` por el que tenga *AWS DeepRacer* en la pegatina de la parte inferior del vehículo.

```
client = awsdeepracer_control.Client(password="XXX",
                    ip="localhost")
```

Para preparar las plataformas de mediciones, se debe seguir el archivo `README.md` mencionado anteriormente. Cabe destacar que se ha modificado ligeramente, cambiando el nombre del directorio *5g-clarity* a `multitechnology`.

Otros cambios que se han realizado son los siguientes:

- `prometheus.yml`: se ha modificado el puerto de trabajo del *scraping*, de 8080 a 18080, debido a que el original ya está en uso por *AWS DeepRacer*. Véase el archivo modificado de configuración de *Prometheus* en Apéndice J.

- `ss_exporter.py`: el servidor de *Prometheus*, del que se va a realizar el *scraping*, se inicia en el puerto 18080 en vez del 8080, algo lógico según lo anterior. Además, para la ejecución de la herramienta `ss`, se ha excluido '`ip netns exec MPTCPns`', ya que en este TFG no se utilizan espacios de nombres (*namespaces*). Véase el *script Python* de exportación de métricas a *Prometheus* en Apéndice K.

- `start_stats.sh`: se ha añadido una regla de `iptables` que permite la conexión *TCP* en el puerto 3000 para la visualización de *Grafana* en el navegador de otro equipo que está conectado a la misma red que el coche. Se ha cambiado, igual que en el `README.md`, el directorio donde se ejecutará `ss_exporter.py`, de `~/vagrant/stats/5g-clarity_testbed_v0_stats` a `~/vagrant/stats/multitechnology_testbed_v0_stats`. Véase el *script bash* de inicialización de servicios de recolección y monitorización de métricas en Apéndice L.

- `stop_stats.sh`: se ha añadido una línea que borra la regla `iptables` que se crea con `start_stats.sh`, pues en caso contrario se crearía una regla nueva cada vez que se realiza un experimento. Además, para detener `ss_exporter`, se hace un `rev` adicional, pues es necesario revertir la operación para que se detenga correctamente. Véase el *script bash* de detención de servicios de recolección y monitorización de métricas en Apéndice M.

- `MPTCP statistics UGR 5GNR + Wi-Fi 6 + LIFI-1683270090539.json`: correspondía al *dashboard* que se le importa a *Grafana* para la visualización de las métricas. Se ha renombrado a `AWS DeepRacer 5G + WIFI-1716887935477.json`, véase archivo *JSON* de configuración de paneles en Apéndice N. Además de cambiar los títulos, se ha cambiado la interfaz de red de la que medir en el campo de `device` de los `targets` de los paneles, a `usb2` para medir la red 5G, y se ha añadido también `mlan0` que es la interfaz asignada a las conexiones *Wi-Fi* en el coche. `Usb2`, por su parte, es la interfaz que se le asigna cuando se hace uso de `USB-tethering`. Esta es la interfaz de la cual se medirán las prestaciones de las redes 5G, y se ha añadido también `mlan0` que es la interfaz asignada a las conexiones *Wi-Fi* en el coche. `Usb2`, por su parte, es la interfaz que se le asigna cuando se hace uso de *USB-tethering*. Esta es la interfaz de la cual se medirán las prestaciones de las redes 5G. Esto también se refleja en el campo de `transformations`, especificando las interfaces de red mencionadas en las expresiones regulares a buscar, `regex`, así como su nombre correspondiente. Así, se le asigna a las métricas obtenidas por `usb2` el nombre 5G NR, y a `mlan0` *WIFI*.

- `aa`: definía la fuente de datos de *Grafana*. Se ha modificado la *IP* `10.0.2.15` a `localhost`, para realizar las pruebas de forma local, como se desea. Véase el archivo de configuración de fuentes de datos de *Grafana* en Apéndice Ñ.

- `import_dashboards.sh`: en este proyecto no se hace uso de este *script*. En vez de eso, se importará la configuración del *dashboard* definido anteriormente de forma manual, como se muestra en la Figura 4.7.

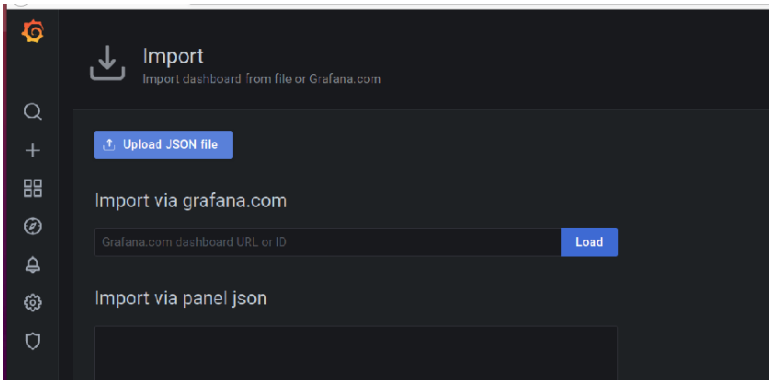

Figura 5.7: Importación manual de la configuración de los *dashboard*s

Por último, es necesario conectar el equipo en el que se pretende visualizar los *dashboards* de *Grafana* y el vehículo al mismo punto de acceso. Además, en el caso de la conexión de este punto con el vehículo, se utilizará la técnica de *USB-tethering* explicada en la Sección 4. De esta forma, se podrá trabajar en local, realizar la experimentación de medidas de redes 5G, y visualizar a tiempo real.

5.5.2. Operación

Una vez preparado el entorno, se puede proceder a la operación con el vehículo.

- Calibración del vehículo: se debe calibrar el vehículo siguiendo la interfaz `Calibration` de la consola de *AWS DeepRacer*, ajustando así la posición de los neumáticos y sentido de giro de los mismos.

- Edición de los archivos de rutas: se debe configurar el archivo de ruta con el que se quiere realizar el experimento. Gracias a que está en formato `json`, es legible y fácil de entender, sin necesitar un entendimiento técnico del funcionamiento de esta implementación. La estructura seguida ya ha sido descrita y está claramente estructurada, sin dar lugar a posibles confusiones. Se muestra un extracto en la Figura 4.8, a modo de ejemplo, del archivo de ruta adjuntado en el apartado en el Apéndice I.

  Este es el ejemplo de ruta más básico. Consiste en un primer paso, movimiento en línea recta, con su correspondiente descripción. Para especificar que este paso debe llevar a cabo un movimiento, se le asigna `move` al campo `action`. Los parámetros indican, respectivamente, el ángulo de giro, aceleración y velocidad máxima que se le pasan como argumentos a las funciones importadas. Además, con el campo

```
[
        {
                "description": "Mover en linea recta - Paso 1",
                "action": "move",
                "params": {
                        "steering_angle": 0.0,
                        "throttle": 0.7,
                        "max_speed": 0.6,
                        "duration": 5
                }
        },
        {
                "description": "Parada 1: Ejecutar ping e iperf3",
                "action": "stop",
                "duration": 10,
                "message": "Ejecutando ping en parada 1",
                "command": "ping -c 4 192.168.222.1 | tee -a ping.log"
        },
```

Figura 5.8: Extracto del archivo de ruta

`duration` se especifica la longitud del intervalo de tiempo durante el cual se quiere realizar este movimiento.

Ahora, en el segundo paso, se tiene una parada, que se ejecuta con el valor `stop` en el campo `action`. Se especifica la duración de la misma de forma análoga al caso anterior, obteniendo un mensaje en consola que informa cuándo se empieza a ejecutar el comando de medición. En este extracto, este comando es un *ping* de 4 paquetes a la dirección *IP* especificada, que actúa como servidor. Además, la sintaxis a la derecha, `| tee -a ping.log` hace que la salida del mismo se recoja en un archivo llamado `ping.log`.

Una vez hecho esto, ya se puede realizar la medición. Se iniciará la medición con `./start_stats.sh`, se colocará el vehículo en el punto de partida y se iniciará su marcha con `python3 ./control_manual.py -r ruta.json`. Al terminar, se ejecutará `./stop_stats.sh`.

- Visualización en equipo monitor: para mostrar los paneles de las mediciones con *Grafana*, se debe acceder con este equipo a `localhost:3000` con el navegador web. Además, ejecutando `python3 ./camara.py` se verá la posición actual del coche. Este *script* (véase *script Python* para la visualización del vídeo de la cámara en Apéndice O) hace uso de *OpenCV* para visualizar el *stream* de vídeo de la cámara. Para ello, importando la biblioteca `cv2` e indicando la *URL* del *stream*, `http://localhost:8080/stream?topic=/camera_pkg/display_mjpeg`, se captura y muestra el vídeo. Para ello se hace uso de las funciones `VideoCapture(url_video)` y `imshow(frame)`. Se ha incluido además un redimensionamiento a 720x480 porque originalmente la ventana era notoriamente pequeña, con `resize(frame, (resolución))`.

5.5.3. Depuración de errores

Se incluye una pequeña subsección explicativa de los errores comunes y persistentes que se han encontrado durante la realización del proyecto.

- **Bloqueo durante el *boot* del coche**: el problema consiste en que, al encender el computador, el *LED* no pasa de amarillo a azul, que indica un funcionamiento normal. En su lugar, se queda con un bloqueo en amarillo, tal y como se describe en [34]. Aunque en ese post se da una posible solución en las respuestas, y los propios desarrolladores del vehículo lo resuelven según [35], no presenta una solución real del problema. Estas soluciones consisten en hacer una restauración a valores de fábrica, por lo que se pierden los archivos existentes. En su lugar, se recomienda seguir los siguientes pasos.

  El primero consiste en prevenir que esto ocurra. La razón de este bloqueo es el escaso tamaño de almacenamiento del coche. Aunque no se instalen archivos adicionales, *ROS* genera *logs* de forma constante en `/root/.ros/logs`. Estos *logs* se generan de forma rápida, llegando a ocupar *gigabytes* de almacenamiento en muy poco tiempo. Esto causa que se llegue a un punto en el que algunos servicios que necesitan generar archivos temporales para su correcto funcionamiento, no puedan iniciarse y funcionar de forma adecuada. Por tanto, se recomienda borrar los *logs* contenidos en ese directorio como superusuario de forma periódica. Además, se puede comprobar la memoria que ocupan si, dentro del directorio, se ejecuta `du -hs`.

  sudo su cd /root/.ros/logs rm *

  El segundo, en caso de que ya se haya producido

  el bloqueo, consiste en utilizar el modo consola de *Ubuntu* durante el bloqueo al arrancar *Ubuntu*. Para ello hay que cambiar a este modo con `Ctrl+Alt+F2`. Este modo no presenta ningún bloqueo, por lo que se puede seguir lo indicado en el paso anterior y realizar un *reboot*.

- **Bloqueo de batería de conducción**: es posible que al intentar encender el motor, no se produzca el pitido que indica el estado de encendido, y el coche no se mueva. Esto se debe a que la batería puede quedar bloqueada, teniendo que seguir los pasos indicados en [36]. De forma resumida, se debe conectar el cable auxiliar corto con extremos blanco y rojo, con los cables de correspondiente color que están conectados a la batería.

- **Detección de la cámara**: es común que no se pueda visualizar el *stream* de la cámara. Esto se debe a que el servicio no se ha iniciado

correctamente. La solución más simple es dejar la cámara conectada y realizar un reinicio del ordenador del coche.

- **Errores de sintaxis en la creación de rutas**: aunque el archivo de rutas es fácil de entender, modificar y replicar, es posible que se tengan errores de sintaxis. Normalmente se deben a falta de caracteres especiales como comas, corchetes y llaves. Para localizar las líneas en las que se producen los fallos, instalando la herramienta `jq` [37], se puede comprobar que la sintaxis del fichero sea correcta. Para ello, se debe hacer uso de:

```
jq . ruta.json
```

Capítulo 6

Experimentación

En esta sección se explicarán los dos experimentos realizados, haciendo uso de la implementación y diseño descritos anteriormente. Se comienza mostrando el montaje (*setup*) utilizado para ellos, en la Figura 6.1.

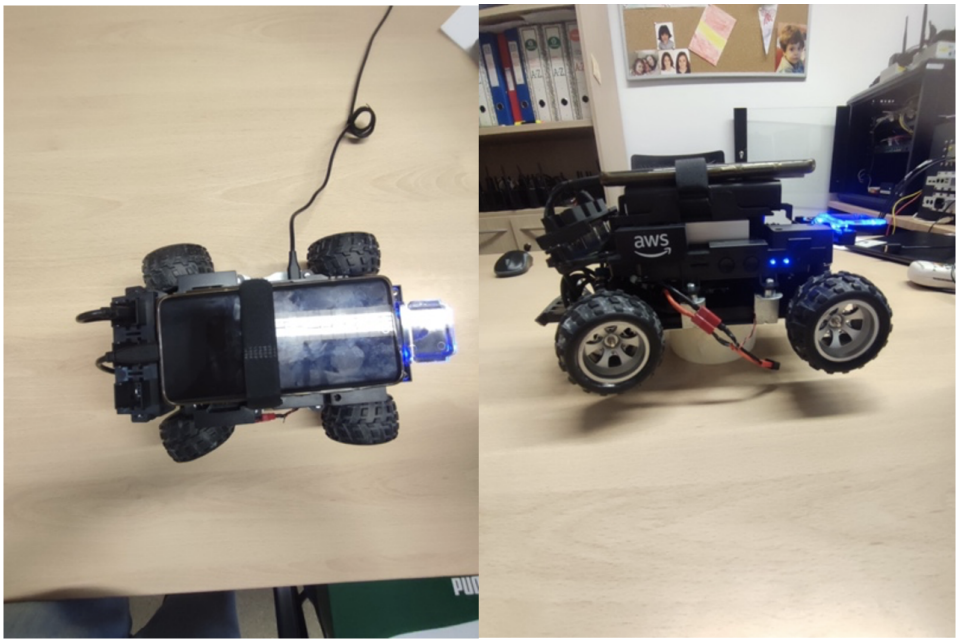

Figura 6.1: La imagen izquierda muestra una vista superior, mientras que la derecha muestra una vista lateral.

La imagen izquierda muestra una vista superior. Se está dando corriente al PC integrado directamente con el cable de alimentación, aunque luego se conectaría la batería del computador al mismo con un cable USB-C a USB-C. Dado que es previo a la calibración, las ruedas no parecen estar en línea recta. A modo de añadido, los valores para la misma han sido:

- Dirección central: -1
- Giro máximo a la izquierda: 22
- Giro máximo a la derecha: -12
- Velocidad de parada: -5
- Dirección hacia adelante: 20
- Velocidad máxima hacia adelante: 16
- Velocidad máxima hacia atrás: -10

La imagen derecha muestra una vista lateral. Los cables rojos que se observan por el exterior portan en el extremo la conexión de la batería del vehículo. Estos cables van dentro del propio chasis, con la batería sujeta con velcro. Sin embargo, en este momento estaba cargando.

Utilizando todos los recursos desarrollados, es decir, visión de la cámara, monitorización de las estadísticas recogidas y ejecución del *script* de control manual con la respectiva ruta, el usuario que ejecuta el experimento observaría algo como lo que se muestra en la Figura 6.2.

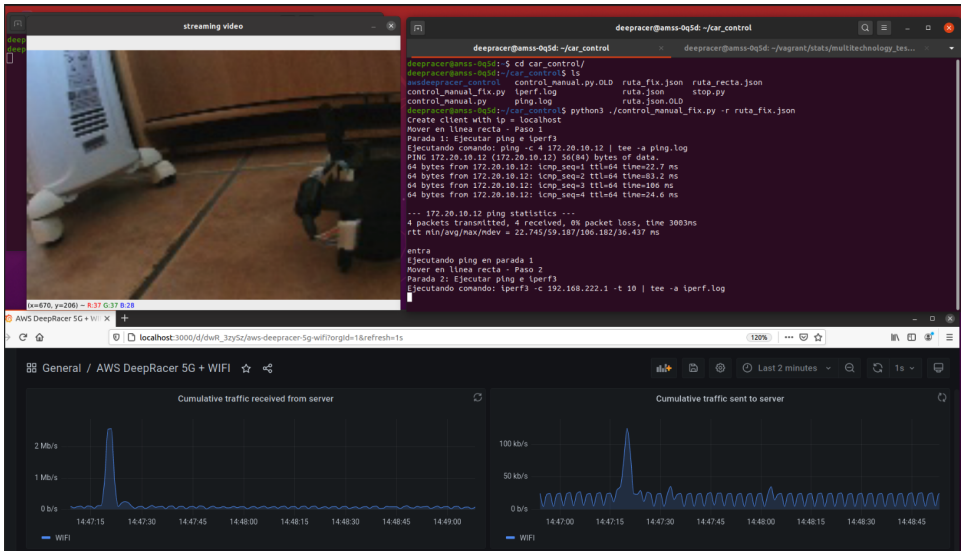

Figura 6.2: Visualización de la cámara, salida por consola de los comandos ejecutados según la ruta especificada al *script* de control y monitorización en *Grafana* de las estadísticas de red, todo en tiempo real.

6.1. Experimento en interior

El primer experimento, en un entorno interior, tuvo lugar en la ETSIIT, ubicada en C/ Periodista Daniel Saucedo Aranda S/N C.P. 18071, Granada. Concretamente, en el pasillo que recorre los diferentes despachos de la segunda planta de la escuela.

El punto de acceso que daba conexión vía *USB tethering* fue un *Xiaomi 12*, conectado al operador *O2*, como se ha expuesto en la Sección 2 para el cálculo de los costes.

Para la visualización en *Grafana*, se utilizó uno de los equipos del despacho 2.19, y se puso el coche en marcha desde la puerta. Su dirección era *192.168.222.1*, tal y como se refleja en la sintaxis de los comandos del fichero de ruta incluido en el Apéndice I. Como es lógico, el coche siguió una ruta en línea recta que realizaba medidas con los comandos *iperf* y *ping* mencionados anteriormente.

Tal y como se pretendía con el experimento, mostrado en [38], se pudo visualizar en tiempo real en el equipo monitor las mediciones que se realizaban con los diferentes servicios ejecutados según la implementación descrita anteriormente. Es cierto que en ese momento el coche no estaba bien calibrado dado que era algo con lo que no se había contado, y por eso parece que está girando hacia la derecha. La calibración óptima en este caso es la incluida en los párrafos superiores.

Figura 6.3: Provenientes de los paneles de los *dashboards* configurados en *Grafana*, que representan el tráfico de red de una conexión 5G NR y WiFi.

La Figura 6.3 muestra dos gráficos provenientes de los paneles de los *dashboards* configurados en *Grafana*, que representan el tráfico de red de una conexión 5G NR y WiFi. Se ha medido el *bitrate*, como se había comentado anteriormente en la configuración en cuestión, usando el vehículo *DeepRacer* como cliente y el equipo del despacho como servidor. Como se indica en la leyenda, el tráfico de 5G NR está representado en color rojo y el WiFi en color azul.

El gráfico izquierdo muestra, como indica el título, el tráfico recibido del servidor, es decir, el tráfico en sentido *downlink*. El eje vertical representa la velocidad de descarga en Mb/s. En el eje horizontal se representa la marca de tiempo de cada medición en formato HH:MM:SS. Se distinguen perfectamente los instantes en los que inicia el comando *ping*. El tráfico de 5G, en rojo, muestra una velocidad constante alrededor de 75 Mb/s, ya que el tráfico generado por *iperf* se enviaba desde el servidor hacia el coche.

El gráfico derecho muestra por su parte el tráfico enviado al servidor, es decir, el sentido *uplink* de la conexión. En el eje vertical se representa la velocidad del tráfico en Mb/s, y en el horizontal, una vez más, las marcas de tiempo de cada medición en formato HH:MM:SS. Fijándose de nuevo en la línea roja, se muestra una velocidad máxima en sentido ascendente de 4 Mb/s, debido a que se trata únicamente del tráfico generado por las confirmaciones de TCP. También se distingue perfectamente el momento de inicio y fin del comando en cuestión. En ambos gráficos, las marcas temporales coinciden en el principio y en el final.

Las velocidades obtenidas son considerablemente más bajas de lo esperado. Esto se debe, primordialmente, al entorno interior en el que se ha realizado el experimento y a la propia naturaleza de la estación utilizada. La velocidad de 5G (75 Mb/s) es baja porque estamos en interior y estamos conectados a la red de un operador real, por lo que había poca cobertura dentro del edificio.

De esta forma, queda demostrada la posibilidad de realizar experimentos de mediciones de diferentes métricas 5G con vehículos automatizados. Con la implementación y diseño detallados se han obtenido una serie de mediciones que muestran las velocidades en sentido ascendente y descendente de la red 5G. Esto es crucial para la determinación de áreas con mayor interferencia, menor cobertura, saturación de la red, así como problemas de *hardware* o de configuración de red.

Aunque la experimentación se haya realizado en un entorno controlado utilizando como punto de acceso un dispositivo que funciona como pasarela a la red de datos en vez de tener una conexión directa, se demuestra la

versatilidad y sinergia de este tipo de tecnologías. Aun así, se ha optado por demostrar que esto es también viable en un entorno exterior.

6.2. Experimento en exterior

El experimento en exterior se realizó de manera similar al experimento en interior. En este caso, tuvo lugar en la Villa de Otura, Granada, con unas condiciones meteorológicas favorables. Se utilizó el mismo dispositivo móvil que en el primer experimento, pero esta vez se tuvo que conectar mediante punto de acceso inalámbrico en lugar de *USB-tethering*, debido a dificultades técnicas.

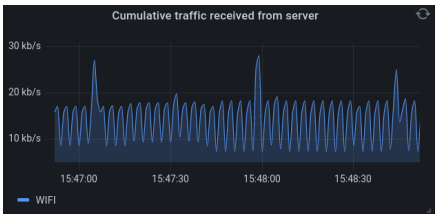

Figura 6.4: Tráfico acumulativo recibido del servidor.

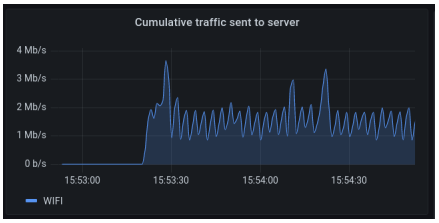

Figura 6.5: Tráfico acumulativo enviado al servidor.

En la Figura 6.4, se muestra el tráfico acumulativo recibido del servidor durante el experimento en exterior. Se observa un patrón de tráfico que varía con picos esporádicos alcanzando hasta 30 kb/s, lo que indica fluctuaciones en la conexión de datos móviles 5G.

La Figura 6.5 muestra el tráfico acumulativo enviado al servidor. En esta gráfica se aprecia que el tráfico de subida alcanza picos de hasta 4 Mb/s.

Aunque era predecible un peor rendimiento de la red al estar utilizando una conexión inalámbrica con el punto de acceso, que generalmente será menos estable que una conexión directamente por *USB-tethering*, ambas gráficas reflejan velocidades muy inferiores a las obtenidas en el experimento de interior. Esto se debe principalmente a una mala conexión móvil a la red 5G en la ubicación donde se ha realizado el experimento en exterior. A pesar de las dificultades técnicas encontradas, los resultados obtenidos son representativos y satisfactorios, ya que se cumple el objetivo de realizar mediciones experimentales de estas redes, mostrando en este caso un mal comportamiento de la red en esta zona.

Capítulo 7

Conclusión

Esta sección recapitulará el cumplimiento de los objetivos planteados en la Sección 1 gracias a las tecnologías usadas, así como la relevancia de los vehículos automatizados en la experimentación 5G y posibles mejoras para el proyecto.

- **Mediciones experimentales de 5G:** ha quedado demostrado que la medición experimental de las redes 5G es viable mediante vehículos automatizados. Con un correcto despliegue de tecnologías que satisfagan, por una parte, el movimiento automatizado del mismo en la zona de experimentación, y por otra, la recolección de las métricas deseadas y su procesamiento, se pueden obtener resultados experimentales de estas mediciones.

- **Automatización:** el experimento en su conjunto está automatizado. Por una parte, con el *script* diseñado y el fichero de ruta importado, se automatiza tanto el movimiento como la propia ejecución de comandos relevantes para la medición de la red. La recolección de las métricas, por su parte, también ha sido automatizada con *scripts* en *Shell* que hacen uso de diferentes herramientas de *Prometheus*. Incluso el posterior procesamiento de estas métricas y su visualización sin necesidad de intervención se permite con una configuración correcta de *Grafana*.

- **Enfocado al usuario:** dada la propia naturaleza de AWS DeepRacer como vehículo autónomo terrestre, como se ha explicado en la Sección 1, las medidas representan una aproximación fiel a lo que un usuario real podría experimentar. Este no habría sido el caso si se hubieran utilizado vehículos autónomos aéreos.

- **Reproducibilidad:** gracias a los ficheros de rutas y la automatización, los experimentos son plenamente reproducibles en caso de que se quieran replicar y repetir mediciones. Con la edición del archivo de ruta en formato *JSON*, que permite fácil entendimiento, legibilidad

y modificación por parte de un usuario con conocimientos básicos, se puede realizar la medición deseada siguiendo la ruta deseada. Además, se ha incluido cómo se realizaría la ejecución paso por paso, para hacer esta tarea lo más simple posible. Se han incluido capturas de lo que un usuario que ejecutase el experimento visualizaría, así como un vídeo demostración [38].

- **Tiempo real:** gracias a la visualización en tiempo real de los paneles de *Grafana* y la posición del coche con *OpenCV*, se puede tener en tiempo real tanto el conocimiento de la situación del vehículo como de la red.

- **Conexión en local:** la medición de red realizada no ha dependido del paso a través de la arquitectura de la nube de Amazon, ni de ningún retardo asociado al uso de enrutamiento para llegar al objetivo. Teniendo los equipos conectados al mismo punto de acceso y la configuración en *localhost*, se pueden realizar los experimentos en local.

- **Bajo costo:** la realización del experimento necesita como única inversión la compra del propio vehículo AWS DeepRacer EVO, o incluso la versión no mejorada en su caso. El resto del *software* es de código abierto y libre uso, por lo que no requerirá inversión adicional. Cada experimento no representa un coste añadido, una vez obtenido el *hardware* e instalado el *software*, se pueden realizar los experimentos deseados, en las condiciones deseadas, el número de veces deseadas.

Los resultados, además, como se ha detallado en la Sección 6, han sido satisfactorios. Aun así, es importante tener en cuenta una serie de factores en los experimentos realizados, como ya se ha mencionado en la Sección 1. Aunque se debe hacer especial énfasis en el peso que presenta el vehículo en cada experimento, pues determinará la velocidad. Y por otra parte, el nivel de batería, especialmente la del motor. Esto es debido a que a menor nivel de batería puede esperarse menor potencia en el motor, así como incluso no poder llegar a finalizar el experimento. Las condiciones meteorológicas en el caso en interior no han influido de forma lógica, y en el caso en exterior han sido favorables. Además, el terreno en ambos experimentos ha sido totalmente liso; en otro tipo de áreas, como zonas rurales, el desempeño de los experimentos podría variar. Y en general, cualquier escenario que presente dificultades para el movimiento del mismo.

Se demuestra también la sinergia comentada al principio del documento entre 5G y vehículos automatizados. Este campo interdisciplinario presenta un gran potencial de desarrollo e investigación, que no se ha aprovechado todavía.

Se han añadido posibles mejoras, o incluso podrían considerarse ideas para futuros experimentos, usando este como base. Estas se han extraído del trabajo con el coche a lo largo de todo el proyecto.

En primer lugar, el uso de un *hardware* que permita una mayor capacidad de cómputo y de memoria en disco duro. Durante la realización del proyecto, ha habido situaciones en las que la falta de potencia de procesamiento ha sido claramente palpable, teniendo incluso momentos de pantalla congelada. Esto, sumado a la memoria en disco duro tan limitada (de la que se ha hablado en la Sección 5), ha significado momentos de bloqueo constante. Como posible solución, se plantea añadir una *Raspberry Pi* como se indica en AWS IoT Moisture Raspberry Pi Setup. Con la configuración adecuada se puede obtener mayor capacidad de procesamiento y permite la integración de tarjetas microSD para aumentar el espacio de memoria Raspberry Pi.

En segundo lugar, la necesidad de un *software* que permita el aprovechamiento del código abierto al máximo. Esto se refiere a que, aunque en el apartado de la Sección 5 se indicara que se ha permitido el trabajo libre con las herramientas del sistema, ha sido solo hasta cierto punto. Esto se debe a que AWS DeepRacer cuenta con un módulo llamado *Secure Boot* que funciona como filtro de la política de seguridad para reforzar la seguridad del mismo. Este filtro en determinadas ocasiones no permite trabajar de una forma tan libre como se habría querido. Cabe destacar que, además, el kernel personalizado que presenta puede suponer dificultades para la instalación de diferentes *drivers*.

Por otra parte, para una mayor automatización de los experimentos, se plantean diferentes alternativas. Esto se debe a que el proceso de automatización actual reside, principalmente, en el ensayo y error, debido a que se debe probar la ruta para ver el desempeño del movimiento de la misma. Se introduce en primer lugar, en casos de interior, el diseño de un primer fichero de ruta a modo de guía usando herramientas de creación de mapas interiores mediante la importación de un plano de edificio a partir del cual se pueda crear una serie de puntos guía a visitar en la ruta. En un caso en exterior mediante la importación de datos geoespaciales en herramientas de generación de rutas de exterior como Open Source Routing Machine.

No obstante, se considera mucho más práctica la utilización de *OpenCV* para navegar de forma autónoma sin la necesidad de definir el movimiento en rutas. Para ello existen diferentes técnicas, como la detección de líneas de carriles con funciones como HoughLinesP(). También se pueden detectar objetos haciendo uso de la función readNet() del módulo DNN. Estas funciones hacen uso de modelos de aprendizaje automático preentrenados. Con una implementación y toma de decisiones adecuadas, el vehículo podría moverse

de forma libre. Esto tendría especial sentido en entornos de interior, en los que la dinámica de la inteligencia artificial no influirá en la reproducibilidad del experimento, pues se seguirá la misma ruta. Es cierto que en un caso en exterior, el coche podría seguir rutas diferentes, aunque por otra parte puede evitar obstáculos inesperados, como personas o animales.

Puede ser también interesante la combinación de mediciones experimentales 5G con vehículos automatizados tanto terrestres como aéreos. Cada vehículo aporta sus propias ventajas y desventajas. Al combinar ambos tipos, en vez de limitarse a solo uno, se tendría una recolección de datos más completa que significaría una mayor fiabilidad en las medidas. Incluso se podría llegar a realizar experimentos con varios vehículos automatizados de forma simultánea cuyas funciones se complementen, consiguiendo una gran eficiencia operativa.

Capítulo 8

Apéndices

Apéndice A. README.md

```
mkdir ~/vagrant/stats

cd ~/vagrant/stats
wget https://dl.grafana.com/oss/release/grafana_8.0.1_amd64.deb
sudo dpkg -i grafana_8.0.1_*.deb

cd ~/vagrant/stats
wget https://github.com/prometheus/prometheus/releases/download/v2.27.1/prometheus-2.27.1.linux-amd64.tar.gz
tar xvfz prometheus-2.27.1.linux-*.tar.gz

sudo apt-get -y install python3-pip
pip3 install prometheus_client

cd ~/vagrant/stats
wget https://github.com/prometheus/node_exporter/releases/download/v1.1.2/node_exporter-1.1.2.linux-amd64.tar.gz
tar xvfz node_exporter-1.1.2.linux-*.tar.gz

cd ~/vagrant/stats
git clone https://github.com/jorgenavarroortiz/5g-clarity_testbed_v0_stats.git

sudo systemctl stop grafana-server
cp ~/vagrant/stats/5g-clarity_testbed_v0_stats/prometheus.yml ~/vagrant/stats/prometheus-2.27.1.linux-amd64/
sudo cp ~/vagrant/stats/5g-clarity_testbed_v0_stats/grafana.ini /etc/grafana/grafana.ini
sudo systemctl start grafana-server
cd ~/vagrant/stats/5g-clarity_testbed_v0_stats
sudo chmod 777 *.sh
./import_datasources.sh
./import_dashboard.sh
sudo systemctl stop grafana-server
```

Apéndice B. prometheusOriginal.yml

A continuación se muestra el contenido del archivo `prometheusOriginal.yml`:

```
# my global config
global:
  scrape_interval:     1s # Set the scrape interval to every 15 seconds. Default is every 1 minute.
  evaluation_interval: 1s # Evaluate rules every 15 seconds. The default is every 1 minute.
  # scrape_timeout is set to the global default (10s).

# Alertmanager configuration
alerting:
  alertmanagers:
  - static_configs:
    - targets:
      # - alertmanager:9093

# Load rules once and periodically evaluate them according to the global 'evaluation_interval'.
rule_files:
  # - "first_rules.yml"
  # - "second_rules.yml"

# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries scraped from this config.
  - job_name: 'prometheus'

    # metrics_path defaults to '/metrics'
    # scheme defaults to 'http'.

    static_configs:
    - targets: ['localhost:9090']

  - job_name: node
    static_configs:
    - targets: ['localhost:9100']

  - job_name: ss
    static_configs:
    - targets: ['localhost:8080']
```

Apéndice C. grafanaOriginal.ini

A continuación se muestra el contenido del archivo `grafanaOriginal.ini`:

```
1  ##################### Grafana Configuration Example
       #####################
2  #
3  # Everything has defaults so you only need to
       uncomment things you want to
4  # change
5
6  # possible values : production, development
7  ;app_mode = production
8
9  # instance name, defaults to HOSTNAME environment
       variable value or hostname if HOSTNAME var is
       empty
10 ;instance_name = ${HOSTNAME}
11
12 ################################### Paths #
       ###################################
13 [paths]
14 # Path to where grafana can store temp files,
       sessions, and the sqlite3 db (if that is used)
15 ;data = /var/lib/grafana
16
17 # Temporary files in `data` directory older than
       given duration will be removed
18 ;temp_data_lifetime = 24h
19
20 # Directory where grafana can store logs
21 ;logs = /var/log/grafana
22
23 # Directory where grafana will automatically scan
       and look for plugins
24 ;plugins = /var/lib/grafana/plugins
25
26 # folder that contains provisioning config files
       that grafana will apply on startup and while
       running.
27 ;provisioning = conf/provisioning
28
29 ################################### Server #
       ###################################
30 [server]
31 # Protocol (http, https, h2, socket)
32 ;protocol = http
```

```
33
34 # The ip address to bind to, empty will bind to all
      interfaces
35 ;http_addr =
36
37 # The http port  to use
38 ;http_port = 3000
39
40 # The public facing domain name used to access
      grafana from a browser
41 ;domain = localhost
42
43 # Redirect to correct domain if host header does not
       match domain
44 # Prevents DNS rebinding attacks
45 ;enforce_domain = false
46
47 # The full public facing url you use in browser,
      used for redirects and emails
48 # If you use reverse proxy and sub path specify full
       url (with sub path)
49 ;root_url = %(protocol)s://%(domain)s:%(http_port)s/
50
51 # Serve Grafana from subpath specified in `root_url`
       setting. By default it is set to `false` for
      compatibility reasons.
52 ;serve_from_sub_path = false
53
54 # Log web requests
55 ;router_logging = false
56
57 # the path relative working path
58 ;static_root_path = public
59
60 # enable gzip
61 ;enable_gzip = false
62
63 # https certs & key file
64 ;cert_file =
65 ;cert_key =
66
67 # Unix socket path
68 ;socket =
69
```

```
70 # CDN Url
71 ;cdn_url =
72
73 # Sets the maximum time using a duration format (5s
      /5m/5ms) before timing out read of an incoming
      request and closing idle connections.
74 # `0` means there is no timeout for reading the
      request.
75 ;read_timeout = 0
76
77 ################################### Database #
      ###################################
78 [database]
79 # You can configure the database connection by
      specifying type, host, name, user and password
80 # as separate properties or as on string using the
      url properties.
81
82 # Either "mysql", "postgres" or "sqlite3", it's your
       choice
83 ;type = sqlite3
84 ;host = 127.0.0.1:3306
85 ;name = grafana
86 ;user = root
87 # If the password contains # or ; you have to wrap
      it with triple quotes. Ex """#password;"""
88 ;password =
89
90 # Use either URL or the previous fields to configure
       the database
91 # Example: mysql://user:secret@host:port/database
92 ;url =
93
94 # For "postgres" only, either "disable", "require"
      or "verify-full"
95 ;ssl_mode = disable
96
97 # Database drivers may support different transaction
       isolation levels.
98 # Currently, only "mysql" driver supports isolation
      levels.
99 # If the value is empty - driver's default isolation
       level is applied.
100 # For "mysql" use "READ-UNCOMMITTED", "READ-
```

```
   COMMITTED", "REPEATABLE-READ" or "SERIALIZABLE".
;isolation_level =

;ca_cert_path =
;client_key_path =
;client_cert_path =
;server_cert_name =

# For "sqlite3" only, path relative to data_path
   setting
;path = grafana.db

# Max idle conn setting default is 2
;max_idle_conn = 2

# Max conn setting default is 0 (mean not set)
;max_open_conn =

# Connection Max Lifetime default is 14400 (means
   14400 seconds or 4 hours)
;conn_max_lifetime = 14400

# Set to true to log the sql calls and execution
   times.
;log_queries =

# For "sqlite3" only. cache mode setting used for
   connecting to the database. (private, shared)
;cache_mode = private

#################################### Data sources #
   #########################
[datasources]
# Upper limit of data sources that Grafana will
   return. This limit is a temporary configuration
   and it will be deprecated when pagination will be
    introduced on the list data sources API.
;datasource_limit = 5000

#################################### Cache server #
   ############################
[remote_cache]
# Either "redis", "memcached" or "database" default
   is "database"
```

```
134 ;type = database
135
136 # cache connectionstring options
137 # database: will use Grafana primary database.
138 # redis: config like redis server e.g. `addr
       =127.0.0.1:6379,pool_size=100,db=0,ssl=false`.
       Only addr is required. ssl may be 'true', 'false'
       , or 'insecure'.
139 # memcache: 127.0.0.1:11211
140 ;connstr =
141
142 ################################### Data proxy #
       ##########################
143 [dataproxy]
144
145 # This enables data proxy logging, default is false
146 ;logging = false
147
148 # How long the data proxy waits to read the headers
       of the response before timing out, default is 30
       seconds.
149 # This setting also applies to core backend HTTP
       data sources where query requests use an HTTP
       client with timeout set.
150 ;timeout = 30
151
152 # How long the data proxy waits to establish a TCP
       connection before timing out, default is 10
       seconds.
153 ;dialTimeout = 10
154
155 # How many seconds the data proxy waits before
       sending a keepalive probe request.
156 ;keep_alive_seconds = 30
157
158 # How many seconds the data proxy waits for a
       successful TLS Handshake before timing out.
159 ;tls_handshake_timeout_seconds = 10
160
161 # How many seconds the data proxy will wait for a
       server's first response headers after
162 # fully writing the request headers if the request
       has an "Expect: 100-continue"
163 # header. A value of 0 will result in the body being
```

```
    sent immediately, without
# waiting for the server to approve.
;expect_continue_timeout_seconds = 1

# The maximum number of idle connections that
   Grafana will keep alive.
;max_idle_connections = 100

# The maximum number of idle connections per host
   that Grafana will keep alive.
;max_idle_connections_per_host = 2

# How many seconds the data proxy keeps an idle
   connection open before timing out.
;idle_conn_timeout_seconds = 90

# If enabled and user is not anonymous, data proxy
   will add X-Grafana-User header with username into
    the request, default is false.
;send_user_header = false

#################################### Analytics
   ####################################
[analytics]
# Server reporting, sends usage counters to stats.
   grafana.org every 24 hours.
# No ip addresses are being tracked, only simple
   counters to track
# running instances, dashboard and error counts. It
   is very helpful to us.
# Change this option to false to disable reporting.
;reporting_enabled = true

# The name of the distributor of the Grafana
   instance. Ex hosted-grafana, grafana-labs
;reporting_distributor = grafana-labs

# Set to false to disable all checks to https://
   grafana.net
# for new versions (grafana itself and plugins),
   check is used
# in some UI views to notify that grafana or plugin
   update exists
# This option does not cause any auto updates, nor
```

```
   send any information
# only a GET request to http://grafana.com to get
   latest versions
;check_for_updates = true

# Google Analytics universal tracking code, only
   enabled if you specify an id here
;google_analytics_ua_id =

# Google Tag Manager ID, only enabled if you specify
    an id here
;google_tag_manager_id =

#################################### Security
   ####################################
[security]
# disable creation of admin user on first start of
   grafana
;disable_initial_admin_creation = false

# default admin user, created on startup
;admin_user = admin

# default admin password, can be changed before
   first start of grafana,  or in profile settings
;admin_password = admin

# used for signing
;secret_key = SW2YcwTIb9zpOOhoPsMm

# disable gravatar profile images
;disable_gravatar = false

# data source proxy whitelist (ip_or_domain:port
   separated by spaces)
;data_source_proxy_whitelist =

# disable protection against brute force login
   attempts
;disable_brute_force_login_protection = false

# set to true if you host Grafana behind HTTPS.
   default is false.
;cookie_secure = false
```

```
228
229 # set cookie SameSite attribute. defaults to `lax`.
       can be set to "lax", "strict", "none" and "
       disabled"
230 ;cookie_samesite = lax
231
232 # set to true if you want to allow browsers to
       render Grafana in a <frame>, <iframe>, <embed> or
        <object>. default is false.
233 ;allow_embedding = false
234
235 # Set to true if you want to enable http strict
       transport security (HSTS) response header.
236 # This is only sent when HTTPS is enabled in this
       configuration.
237 # HSTS tells browsers that the site should only be
       accessed using HTTPS.
238 ;strict_transport_security = false
239
240 # Sets how long a browser should cache HSTS. Only
       applied if strict_transport_security is enabled.
241 ;strict_transport_security_max_age_seconds = 86400
242
243 # Set to true if to enable HSTS preloading option.
       Only applied if strict_transport_security is
       enabled.
244 ;strict_transport_security_preload = false
245
246 # Set to true if to enable the HSTS
       includeSubDomains option. Only applied if
       strict_transport_security is enabled.
247 ;strict_transport_security_subdomains = false
248
249 # Set to true to enable the X-Content-Type-Options
       response header.
250 # The X-Content-Type-Options response HTTP header is
        a marker used by the server to indicate that the
        MIME types advertised
251 # in the Content-Type headers should not be changed
       and be followed.
252 ;x_content_type_options = true
253
254 # Set to true to enable the X-XSS-Protection header,
        which tells browsers to stop pages from loading
```

```
255 # when they detect reflected cross-site scripting (
       XSS) attacks.
256 ;x_xss_protection = true
257
258 # Enable adding the Content-Security-Policy header
       to your requests.
259 # CSP allows to control resources the user agent is
       allowed to load and helps prevent XSS attacks.
260 ;content_security_policy = false
261
262 # Set Content Security Policy template used when
       adding the Content-Security-Policy header to your
        requests.
263 # $NONCE in the template includes a random nonce.
264 # $ROOT_PATH is server.root_url without the protocol
       .
265 ;content_security_policy_template = """script-src '
       self' 'unsafe-eval' 'unsafe-inline' 'strict-
       dynamic' $NONCE;object-src 'none';font-src 'self'
       ;style-src 'self' 'unsafe-inline' blob:;img-src *
        data:;base-uri 'self';connect-src 'self' grafana
       .com ws://$ROOT_PATH wss://$ROOT_PATH;manifest-
       src 'self';media-src 'none';form-action 'self'
       ;"""
266
267 ################################### Snapshots
       ############################
268 [snapshots]
269 # snapshot sharing options
270 ;external_enabled = true
271 ;external_snapshot_url = https://snapshots-origin.
       raintank.io
272 ;external_snapshot_name = Publish to snapshot.
       raintank.io
273
274 # Set to true to enable this Grafana instance act as
        an external snapshot server and allow
       unauthenticated requests for
275 # creating and deleting snapshots.
276 ;public_mode = false
277
278 # remove expired snapshot
279 ;snapshot_remove_expired = true
280
```

```
281 ################################### Dashboards
       History ##################
282 [dashboards]
283 # Number dashboard versions to keep (per dashboard).
        Default: 20, Minimum: 1
284 ;versions_to_keep = 20
285
286 # Minimum dashboard refresh interval. When set, this
        will restrict users to set the refresh interval
       of a dashboard lower than given interval. Per
       default this is 5 seconds.
287 # The interval string is a possibly signed sequence
       of decimal numbers, followed by a unit suffix (ms
       , s, m, h, d), e.g. 30s or 1m.
288 min_refresh_interval = 1s
289
290 # Path to the default home dashboard. If this value
       is empty, then Grafana uses StaticRootPath + "
       dashboards/home.json"
291 ;default_home_dashboard_path =
292
293 ################################### Users
       ################################
294 [users]
295 # disable user signup / registration
296 ;allow_sign_up = true
297
298 # Allow non admin users to create organizations
299 ;allow_org_create = true
300
301 # Set to true to automatically assign new users to
       the default organization (id 1)
302 ;auto_assign_org = true
303
304 # Set this value to automatically add new users to
       the provided organization (if auto_assign_org
       above is set to true)
305 ;auto_assign_org_id = 1
306
307 # Default role new users will be automatically
       assigned (if disabled above is set to true)
308 ;auto_assign_org_role = Viewer
309
310 # Require email validation before sign up completes
```

```
311 ;verify_email_enabled = false
312
313 # Background text for the user field on the login
        page
314 ;login_hint = email or username
315 ;password_hint = password
316
317 # Default UI theme ("dark" or "light")
318 ;default_theme = dark
319
320 # Path to a custom home page. Users are only
        redirected to this if the default home dashboard
        is used. It should match a frontend route and
        contain a leading slash.
321 ; home_page =
322
323 # External user management, these options affect the
         organization users view
324 ;external_manage_link_url =
325 ;external_manage_link_name =
326 ;external_manage_info =
327
328 # Viewers can edit/inspect dashboard settings in the
         browser. But not save the dashboard.
329 ;viewers_can_edit = false
330
331 # Editors can administrate dashboard, folders and
        teams they create
332 ;editors_can_admin = false
333
334 # The duration in time a user invitation remains
        valid before expiring. This setting should be
        expressed as a duration. Examples: 6h (hours), 2d
         (days), 1w (week). Default is 24h (24 hours).
        The minimum supported duration is 15m (15 minutes
        ).
335 ;user_invite_max_lifetime_duration = 24h
336
337 # Enter a comma-separated list of users login to
        hide them in the Grafana UI. These users are
        shown to Grafana admins and themselves.
338 ; hidden_users =
339
340 [auth]
```

```
341 # Login cookie name
342 ;login_cookie_name = grafana_session
343
344 # The maximum lifetime (duration) an authenticated
       user can be inactive before being required to
       login at next visit. Default is 7 days (7d). This
        setting should be expressed as a duration, e.g.
       5m (minutes), 6h (hours), 10d (days), 2w (weeks),
        1M (month). The lifetime resets at each
       successful token rotation.
345 ;login_maximum_inactive_lifetime_duration =
346
347 # The maximum lifetime (duration) an authenticated
       user can be logged in since login time before
       being required to login. Default is 30 days (30d)
       . This setting should be expressed as a duration,
        e.g. 5m (minutes), 6h (hours), 10d (days), 2w (
       weeks), 1M (month).
348 ;login_maximum_lifetime_duration =
349
350 # How often should auth tokens be rotated for
       authenticated users when being active. The
       default is each 10 minutes.
351 ;token_rotation_interval_minutes = 10
352
353 # Set to true to disable (hide) the login form,
       useful if you use OAuth, defaults to false
354 ;disable_login_form = false
355
356 # Set to true to disable the sign out link in the
       side menu. Useful if you use auth.proxy or auth.
       jwt, defaults to false
357 ;disable_signout_menu = false
358
359 # URL to redirect the user to after sign out
360 ;signout_redirect_url =
361
362 # Set to true to attempt login with OAuth
       automatically, skipping the login screen.
363 # This setting is ignored if multiple OAuth
       providers are configured.
364 ;oauth_auto_login = false
365
366 # OAuth state max age cookie duration in seconds.
```

```
   Defaults to 600 seconds.
;oauth_state_cookie_max_age = 600

# limit of api_key seconds to live before expiration
;api_key_max_seconds_to_live = -1

# Set to true to enable SigV4 authentication option
   for HTTP-based datasources.
;sigv4_auth_enabled = false

################################### Anonymous Auth
   ######################
[auth.anonymous]
# enable anonymous access
;enabled = false

# specify organization name that should be used for
   unauthenticated users
;org_name = Main Org.

# specify role for unauthenticated users
;org_role = Viewer

# mask the Grafana version number for
   unauthenticated users
;hide_version = false

################################### GitHub Auth
   ##########################
[auth.github]
;enabled = false
;allow_sign_up = true
;client_id = some_id
;client_secret = some_secret
;scopes = user:email,read:org
;auth_url = https://github.com/login/oauth/authorize
;token_url = https://github.com/login/oauth/
   access_token
;api_url = https://api.github.com/user
;allowed_domains =
;team_ids =
;allowed_organizations =

################################### GitLab Auth
```

```
   ##########################
[auth.gitlab]
;enabled = false
;allow_sign_up = true
;client_id = some_id
;client_secret = some_secret
;scopes = api
;auth_url = https://gitlab.com/oauth/authorize
;token_url = https://gitlab.com/oauth/token
;api_url = https://gitlab.com/api/v4
;allowed_domains =
;allowed_groups =

################################### Google Auth
   ##########################
[auth.google]
;enabled = false
;allow_sign_up = true
;client_id = some_client_id
;client_secret = some_client_secret
;scopes = https://www.googleapis.com/auth/userinfo.
   profile https://www.googleapis.com/auth/userinfo.
   email
;auth_url = https://accounts.google.com/o/oauth2/
   auth
;token_url = https://accounts.google.com/o/oauth2/
   token
;api_url = https://www.googleapis.com/oauth2/v1/
   userinfo
;allowed_domains =
;hosted_domain =

################################### Grafana.com
   Auth ####################
[auth.grafana_com]
;enabled = false
;allow_sign_up = true
;client_id = some_id
;client_secret = some_secret
;scopes = user:email
;allowed_organizations =

################################### Azure AD OAuth
   ######################
```

```
[auth.azuread]
;name = Azure AD
;enabled = false
;allow_sign_up = true
;client_id = some_client_id
;client_secret = some_client_secret
;scopes = openid email profile
;auth_url = https://login.microsoftonline.com/<
   tenant-id>/oauth2/v2.0/authorize
;token_url = https://login.microsoftonline.com/<
   tenant-id>/oauth2/v2.0/token
;allowed_domains =
;allowed_groups =

################################### Okta OAuth
   ########################
[auth.okta]
;name = Okta
;enabled = false
;allow_sign_up = true
;client_id = some_id
;client_secret = some_secret
;scopes = openid profile email groups
;auth_url = https://<tenant-id>.okta.com/oauth2/v1/
   authorize
;token_url = https://<tenant-id>.okta.com/oauth2/v1/
   token
;api_url = https://<tenant-id>.okta.com/oauth2/v1/
   userinfo
;allowed_domains =
;allowed_groups =
;role_attribute_path =
;role_attribute_strict = false

################################### Generic OAuth
   ##########################
[auth.generic_oauth]
;enabled = false
;name = OAuth
;allow_sign_up = true
;client_id = some_id
;client_secret = some_secret
;scopes = user:email,read:org
;empty_scopes = false
```

```
;email_attribute_name = email:primary
;email_attribute_path =
;login_attribute_path =
;name_attribute_path =
;id_token_attribute_name =
;auth_url = https://foo.bar/login/oauth/authorize
;token_url = https://foo.bar/login/oauth/
   access_token
;api_url = https://foo.bar/user
;allowed_domains =
;team_ids =
;allowed_organizations =
;role_attribute_path =
;role_attribute_strict = false
;tls_skip_verify_insecure = false
;tls_client_cert =
;tls_client_key =
;tls_client_ca =

################################### Basic Auth
   ##########################
[auth.basic]
;enabled = true

################################### Auth Proxy
   ##########################
[auth.proxy]
;enabled = false
;header_name = X-WEBAUTH-USER
;header_property = username
;auto_sign_up = true
;sync_ttl = 60
;whitelist = 192.168.1.1, 192.168.2.1
;headers = Email:X-User-Email, Name:X-User-Name
# Read the auth proxy docs for details on what the
   setting below enables
;enable_login_token = false

################################### Auth JWT
   ##########################
[auth.jwt]
;enabled = true
;header_name = X-JWT-Assertion
;email_claim = sub
```

```
515 ;username_claim = sub
516 ;jwk_set_url = https://foo.bar/.well-known/jwks.json
517 ;jwk_set_file = /path/to/jwks.json
518 ;cache_ttl = 60m
519 ;expected_claims = {"aud": ["foo", "bar"]}
520 ;key_file = /path/to/key/file
521
522 #################################### Auth LDAP
       ##########################
523 [auth.ldap]
524 ;enabled = false
525 ;config_file = /etc/grafana/ldap.toml
526 ;allow_sign_up = true
527
528 # LDAP background sync (Enterprise only)
529 # At 1 am every day
530 ;sync_cron = "0 0 1 * * *"
531 ;active_sync_enabled = true
532
533 #################################### AWS
       ############################
534 [aws]
535 # Enter a comma-separated list of allowed AWS
       authentication providers.
536 # Options are: default (AWS SDK Default), keys (
       Access && secret key), credentials (Credentials
       field), ec2_iam_role (EC2 IAM Role)
537 ; allowed_auth_providers = default,keys,credentials
538
539 # Allow AWS users to assume a role using temporary
       security credentials.
540 # If true, assume role will be enabled for all AWS
       authentication providers that are specified in
       aws_auth_providers
541 ; assume_role_enabled = true
542
543 #################################### Azure
       ###############################
544 [azure]
545 # Azure cloud environment where Grafana is hosted
546 # Possible values are AzureCloud, AzureChinaCloud,
       AzureUSGovernment and AzureGermanCloud
547 # Default value is AzureCloud (i.e. public cloud)
548 ;cloud = AzureCloud
```

```
549
550 # Specifies whether Grafana hosted in Azure service with Managed Identity configured (e.g. Azure Virtual Machines instance)
551 # If enabled, the managed identity can be used for authentication of Grafana in Azure services
552 # Disabled by default, needs to be explicitly enabled
553 ;managed_identity_enabled = false
554
555 # Client ID to use for user-assigned managed identity
556 # Should be set for user-assigned identity and should be empty for system-assigned identity
557 ;managed_identity_client_id =
558
559 #################################### SMTP / Emailing ##########################
560 [smtp]
561 ;enabled = false
562 ;host = localhost:25
563 ;user =
564 # If the password contains # or ; you have to wrap it with triple quotes. Ex """#password;"""
565 ;password =
566 ;cert_file =
567 ;key_file =
568 ;skip_verify = false
569 ;from_address = admin@grafana.localhost
570 ;from_name = Grafana
571 # EHLO identity in SMTP dialog (defaults to instance_name)
572 ;ehlo_identity = dashboard.example.com
573 # SMTP startTLS policy (defaults to 'OpportunisticStartTLS')
574 ;startTLS_policy = NoStartTLS
575
576 [emails]
577 ;welcome_email_on_sign_up = false
578 ;templates_pattern = emails/*.html
579
580 #################################### Logging ##########################
581 [log]
```

```
582 # Either "console", "file", "syslog". Default is
       console and  file
583 # Use space to separate multiple modes, e.g. "
       console file"
584 ;mode = console file
585
586 # Either "debug", "info", "warn", "error", "critical
       ", default is "info"
587 ;level = info
588
589 # optional settings to set different levels for
       specific loggers. Ex filters = sqlstore:debug
590 ;filters =
591
592 # For "console" mode only
593 [log.console]
594 ;level =
595
596 # log line format, valid options are text, console
       and json
597 ;format = console
598
599 # For "file" mode only
600 [log.file]
601 ;level =
602
603 # log line format, valid options are text, console
       and json
604 ;format = text
605
606 # This enables automated log rotate(switch of
       following options), default is true
607 ;log_rotate = true
608
609 # Max line number of single file, default is 1000000
610 ;max_lines = 1000000
611
612 # Max size shift of single file, default is 28 means
        1 << 28, 256MB
613 ;max_size_shift = 28
614
615 # Segment log daily, default is true
616 ;daily_rotate = true
617
```

```
618 # Expired days of log file(delete after max days),
       default is 7
619 ;max_days = 7
620
621 [log.syslog]
622 ;level =
623
624 # log line format, valid options are text, console
       and json
625 ;format = text
626
627 # Syslog network type and address. This can be udp,
       tcp, or unix. If left blank, the default unix
       endpoints will be used.
628 ;network =
629 ;address =
630
631 # Syslog facility. user, daemon and local0 through
       local7 are valid.
632 ;facility =
633
634 # Syslog tag. By default, the process' argv[0] is
       used.
635 ;tag =
636
637 [log.frontend]
638 # Should Sentry javascript agent be initialized
639 ;enabled = false
640
641 # Sentry DSN if you want to send events to Sentry.
642 ;sentry_dsn =
643
644 # Custom HTTP endpoint to send events captured by
       the Sentry agent to. Default will log the events
       to stdout.
645 ;custom_endpoint = /log
646
647 # Rate of events to be reported between 0 (none) and
        1 (all), float
648 ;sample_rate = 1.0
649
650 # Requests per second limit enforced an extended
       period, for Grafana backend log ingestion
       endpoint (/log).
```

```
651 ;log_endpoint_requests_per_second_limit = 3
652
653 # Max requests accepted per short interval of time
      for Grafana backend log ingestion endpoint (/log)
      .
654 ;log_endpoint_burst_limit = 15
655
656 ################################### Usage Quotas #
      ########################
657 [quota]
658 ; enabled = false
659
660 #### set quotas to -1 to make unlimited. ####
661 # limit number of users per Org.
662 ; org_user = 10
663
664 # limit number of dashboards per Org.
665 ; org_dashboard = 100
666
667 # limit number of data_sources per Org.
668 ; org_data_source = 10
669
670 # limit number of api_keys per Org.
671 ; org_api_key = 10
672
673 # limit number of alerts per Org.
674 ;org_alert_rule = 100
675
676 # limit number of orgs a user can create.
677 ; user_org = 10
678
679 # Global limit of users.
680 ; global_user = -1
681
682 # global limit of orgs.
683 ; global_org = -1
684
685 # global limit of dashboards
686 ; global_dashboard = -1
687
688 # global limit of api_keys
689 ; global_api_key = -1
690
691 # global limit on number of logged in users.
```

```
692 ; global_session = -1
693
694 # global limit of alerts
695 ;global_alert_rule = -1
696
697 ################################### Alerting #
      ############################
698 [alerting]
699 # Disable alerting engine & UI features
700 ;enabled = true
701 # Makes it possible to turn off alert rule execution
       but alerting UI is visible
702 ;execute_alerts = true
703
704 # Default setting for new alert rules. Defaults to
      categorize error and timeouts as alerting. (
      alerting, keep_state)
705 ;error_or_timeout = alerting
706
707 # Default setting for how Grafana handles nodata or
      null values in alerting. (alerting, no_data,
      keep_state, ok)
708 ;nodata_or_nullvalues = no_data
709
710 # Alert notifications can include images, but
      rendering many images at the same time can
      overload the server
711 # This limit will protect the server from render
      overloading and make sure notifications are sent
      out quickly
712 ;concurrent_render_limit = 5
713
714
715 # Default setting for alert calculation timeout.
      Default value is 30
716 ;evaluation_timeout_seconds = 30
717
718 # Default setting for alert notification timeout.
      Default value is 30
719 ;notification_timeout_seconds = 30
720
721 # Default setting for max attempts to sending alert
      notifications. Default value is 3
722 ;max_attempts = 3
```

```
723
724 # Makes it possible to enforce a minimal interval
       between evaluations, to reduce load on the
       backend
725 ;min_interval_seconds = 1
726
727 # Configures for how long alert annotations are
       stored. Default is 0, which keeps them forever.
728 # This setting should be expressed as a duration.
       Examples: 6h (hours), 10d (days), 2w (weeks), 1M
       (month).
729 ;max_annotation_age =
730
731 # Configures max number of alert annotations that
       Grafana stores. Default value is 0, which keeps
       all alert annotations.
732 ;max_annotations_to_keep =
733
734 #################################### Annotations #
       ########################
735 [annotations]
736 # Configures the batch size for the annotation clean
       -up job. This setting is used for dashboard, API,
        and alert annotations.
737 ;cleanupjob_batchsize = 100
738
739 [annotations.dashboard]
740 # Dashboard annotations means that annotations are
       associated with the dashboard they are created on
       .
741
742 # Configures how long dashboard annotations are
       stored. Default is 0, which keeps them forever.
743 # This setting should be expressed as a duration.
       Examples: 6h (hours), 10d (days), 2w (weeks), 1M
       (month).
744 ;max_age =
745
746 # Configures max number of dashboard annotations
       that Grafana stores. Default value is 0, which
       keeps all dashboard annotations.
747 ;max_annotations_to_keep =
748
749 [annotations.api]
```

```
750 # API annotations means that the annotations have
        been created using the API without any
751 # association with a dashboard.
752
753 # Configures how long Grafana stores API annotations
        . Default is 0, which keeps them forever.
754 # This setting should be expressed as a duration.
        Examples: 6h (hours), 10d (days), 2w (weeks), 1M
        (month).
755 ;max_age =
756
757 # Configures max number of API annotations that
        Grafana keeps. Default value is 0, which keeps
        all API annotations.
758 ;max_annotations_to_keep =
759
760 ################################### Explore #
        ############################
761 [explore]
762 # Enable the Explore section
763 ;enabled = true
764
765 ################################### Internal
        Grafana Metrics ##########################
766 # Metrics available at HTTP API Url /metrics
767 [metrics]
768 # Disable / Enable internal metrics
769 ;enabled           = true
770 # Graphite Publish interval
771 ;interval_seconds  = 10
772 # Disable total stats (stat_totals_*) metrics to be
        generated
773 ;disable_total_stats = false
774
775 #If both are set, basic auth will be required for
        the metrics endpoint.
776 ; basic_auth_username =
777 ; basic_auth_password =
778
779 # Metrics environment info adds dimensions to the `
        grafana_environment_info` metric, which
780 # can expose more information about the Grafana
        instance.
781 [metrics.environment_info]
```

```
782 #exampleLabel1 = exampleValue1
783 #exampleLabel2 = exampleValue2
784
785 # Send internal metrics to Graphite
786 [metrics.graphite]
787 # Enable by setting the address setting (ex
    localhost:2003)
788 ;address =
789 ;prefix = prod.grafana.%(instance_name)s.
790
791 ################################### Grafana.com
    integration  ##########################
792 # Url used to import dashboards directly from
    Grafana.com
793 [grafana_com]
794 ;url = https://grafana.com
795
796 ################################### Distributed
    tracing ############
797 [tracing.jaeger]
798 # Enable by setting the address sending traces to
    jaeger (ex localhost:6831)
799 ;address = localhost:6831
800 # Tag that will always be included in when creating
    new spans. ex (tag1:value1,tag2:value2)
801 ;always_included_tag = tag1:value1
802 # Type specifies the type of the sampler: const,
    probabilistic, rateLimiting, or remote
803 ;sampler_type = const
804 # jaeger samplerconfig param
805 # for "const" sampler, 0 or 1 for always false/true
    respectively
806 # for "probabilistic" sampler, a probability between
     0 and 1
807 # for "rateLimiting" sampler, the number of spans
    per second
808 # for "remote" sampler, param is the same as for "
    probabilistic"
809 # and indicates the initial sampling rate before the
     actual one
810 # is received from the mothership
811 ;sampler_param = 1
812 # sampling_server_url is the URL of a sampling
    manager providing a sampling strategy.
```

```
;sampling_server_url =
# Whether or not to use Zipkin propagation (x-b3-
   HTTP headers).
;zipkin_propagation = false
# Setting this to true disables shared RPC spans.
# Not disabling is the most common setting when
   using Zipkin elsewhere in your infrastructure.
;disable_shared_zipkin_spans = false

#################################### External image
   storage ##########################
[external_image_storage]
# Used for uploading images to public servers so
   they can be included in slack/email messages.
# you can choose between (s3, webdav, gcs,
   azure_blob, local)
;provider =

[external_image_storage.s3]
;endpoint =
;path_style_access =
;bucket =
;region =
;path =
;access_key =
;secret_key =

[external_image_storage.webdav]
;url =
;public_url =
;username =
;password =

[external_image_storage.gcs]
;key_file =
;bucket =
;path =

[external_image_storage.azure_blob]
;account_name =
;account_key =
;container_name =

[external_image_storage.local]
```

```
852 # does not require any configuration
853
854 [rendering]
855 # Options to configure a remote HTTP image rendering
        service, e.g. using https://github.com/grafana/
    grafana-image-renderer.
856 # URL to a remote HTTP image renderer service, e.g.
    http://localhost:8081/render, will enable Grafana
        to render panels and dashboards to PNG-images
    using HTTP requests to an external service.
857 ;server_url =
858 # If the remote HTTP image renderer service runs on
    a different server than the Grafana server you
    may have to configure this to a URL where Grafana
        is reachable, e.g. http://grafana.domain/.
859 ;callback_url =
860 # Concurrent render request limit affects when the /
    render HTTP endpoint is used. Rendering many
    images at the same time can overload the server,
861 # which this setting can help protect against by
    only allowing a certain amount of concurrent
    requests.
862 ;concurrent_render_request_limit = 30
863
864 [panels]
865 # If set to true Grafana will allow script tags in
    text panels. Not recommended as it enable XSS
    vulnerabilities.
866 ;disable_sanitize_html = false
867
868 [plugins]
869 ;enable_alpha = false
870 ;app_tls_skip_verify_insecure = false
871 # Enter a comma-separated list of plugin identifiers
        to identify plugins that are allowed to be
    loaded even if they lack a valid signature.
872 ;allow_loading_unsigned_plugins =
873 # Enable or disable installing plugins directly from
        within Grafana.
874 ;plugin_admin_enabled = false
875 ;plugin_admin_external_manage_enabled = false
876 ;plugin_catalog_url = https://grafana.com/grafana/
    plugins/
877
```

```
878 ################################### Grafana Live #
    ########################################
879 [live]
880 # max_connections to Grafana Live WebSocket endpoint
     per Grafana server instance. See Grafana Live
    docs
881 # if you are planning to make it higher than default
     100 since this can require some OS and
    infrastructure
882 # tuning. 0 disables Live, -1 means unlimited
    connections.
883 ;max_connections = 100
884
885 ################################### Grafana Image
    Renderer Plugin ##########################
886 [plugin.grafana-image-renderer]
887 # Instruct headless browser instance to use a
    default timezone when not provided by Grafana, e.
    g. when rendering panel image of alert.
888 # See ICUs metaZones.txt (https://cs.chromium.org
    /chromium/src/third_party/icu/source/data/misc/
    metaZones.txt) for a list of supported
889 # timezone IDs. Fallbacks to TZ environment variable
     if not set.
890 ;rendering_timezone =
891
892 # Instruct headless browser instance to use a
    default language when not provided by Grafana, e.
    g. when rendering panel image of alert.
893 # Please refer to the HTTP header Accept-Language to
     understand how to format this value, e.g. 'fr-CH
    , fr;q=0.9, en;q=0.8, de;q=0.7, *;q=0.5'.
894 ;rendering_language =
895
896 # Instruct headless browser instance to use a
    default device scale factor when not provided by
    Grafana, e.g. when rendering panel image of alert
    .
897 # Default is 1. Using a higher value will produce
    more detailed images (higher DPI), but will
    require more disk space to store an image.
898 ;rendering_viewport_device_scale_factor =
899
900 # Instruct headless browser instance whether to
```

```
   ignore HTTPS errors during navigation. Per
   default HTTPS errors are not ignored. Due to
# the security risk it's not recommended to ignore
   HTTPS errors.
;rendering_ignore_https_errors =

# Instruct headless browser instance whether to
   capture and log verbose information when
   rendering an image. Default is false and will
# only capture and log error messages. When enabled,
    debug messages are captured and logged as well.
# For the verbose information to be included in the
   Grafana server log you have to adjust the
   rendering log level to debug, configure
# [log].filter = rendering:debug.
;rendering_verbose_logging =

# Instruct headless browser instance whether to
   output its debug and error messages into running
   process of remote rendering service.
# Default is false. This can be useful to enable (
   true) when troubleshooting.
;rendering_dumpio =

# Additional arguments to pass to the headless
   browser instance. Default is --no-sandbox. The
   list of Chromium flags can be found
# here (https://peter.sh/experiments/chromium-
   command-line-switches/). Multiple arguments is
   separated with comma-character.
;rendering_args =

# You can configure the plugin to use a different
   browser binary instead of the pre-packaged
   version of Chromium.
# Please note that this is not recommended, since
   you may encounter problems if the installed
   version of Chrome/Chromium is not
# compatible with the plugin.
;rendering_chrome_bin =

# Instruct how headless browser instances are
   created. Default is 'default' and will create a
   new browser instance on each request.
```

```
924 # Mode 'clustered' will make sure that only a
       maximum of browsers/incognito pages can execute
       concurrently.
925 # Mode 'reusable' will have one browser instance and
        will create a new incognito page on each request
       .
926 ;rendering_mode =
927
928 # When rendering_mode = clustered you can instruct
       how many browsers or incognito pages can execute
       concurrently. Default is 'browser'
929 # and will cluster using browser instances.
930 # Mode 'context' will cluster using incognito pages.
931 ;rendering_clustering_mode =
932 # When rendering_mode = clustered you can define
       maximum number of browser instances/incognito
       pages that can execute concurrently..
933 ;rendering_clustering_max_concurrency =
934
935 # Limit the maximum viewport width, height and
       device scale factor that can be requested.
936 ;rendering_viewport_max_width =
937 ;rendering_viewport_max_height =
938 ;rendering_viewport_max_device_scale_factor =
939
940 # Change the listening host and port of the gRPC
       server. Default host is 127.0.0.1 and default
       port is 0 and will automatically assign
941 # a port not in use.
942 ;grpc_host =
943 ;grpc_port =
944
945 [enterprise]
946 # Path to a valid Grafana Enterprise license.jwt
       file
947 ;license_path =
948
949 [feature_toggles]
950 # enable features, separated by spaces
951 ;enable =
952
953 [date_formats]
954 # For information on what formatting patterns that
       are supported https://momentjs.com/docs/#/
```

```
   displaying/
955
956 # Default system date format used in time range
       picker and other places where full time is
       displayed
957 ;full_date = YYYY-MM-DD HH:mm:ss
958
959 # Used by graph and other places where we only show
       small intervals
960 ;interval_second = HH:mm:ss
961 ;interval_minute = HH:mm
962 ;interval_hour = MM/DD HH:mm
963 ;interval_day = MM/DD
964 ;interval_month = YYYY-MM
965 ;interval_year = YYYY
966
967 # Experimental feature
968 ;use_browser_locale = false
969
970 # Default timezone for user preferences. Options are
       'browser' for the browser local timezone or a
       timezone name from IANA Time Zone database, e.g.
       'UTC' or 'Europe/Amsterdam' etc.
971 ;default_timezone = browser
972
973 [expressions]
974 # Enable or disable the expressions functionality.
975 ;enabled = true
```

Apéndice D. importdatasourcesOriginal.sh

A continuación se muestra el contenido del script `importdatasourcesOriginal.sh`:

```
1 #!/bin/bash
2
3 for i in data_sources/*; do \
4     curl -X "POST" "http://localhost:3000/api/datasources" \
5     -H "Content-Type: application/json" \
6      --user admin:admin \
7      --data-binary @$i
8 done
```

Apéndice E. importdashboardsOriginal.sh

A continuación se muestra el contenido del script `importdashboardsOriginal.sh`:

```
#!/bin/bash

curl -XPOST -i http://localhost:3000/api/dashboards/import -u admin:
    admin --data-binary @./mptcp_dashboard.json -H "Content-Type:
    application/json"
```

Apéndice F. startstatsOriginal

A continuación se muestra el contenido del script `startstatsOriginal`:

```
#!/bin/bash



sudo systemctl start grafana-server

cd ~/vagrant/stats/node_exporter-1.1.2.linux-amd64
./node_exporter &

cd ~/vagrant/stats/prometheus-2.27.1.linux-amd64
./prometheus --config.file=./prometheus.yml &

cd ~/vagrant/stats/5g-clarity_testbed_v0_stats
python3 ./ss_exporter.py &
```

Apéndice G. ss_exporterOriginal

A continuación se muestra el contenido del script `ss_exporterOriginal`:

```
# SS exporter for Prometheus (i.e. RTT, CWND, ... TCP flow parameters)


# See https://sysdig.com/blog/prometheus-metrics/

# Example of output from ss -i -t:
#State                Recv-Q                Send-Q
                                    Local Address:Port
                                    Peer Address:Port
#ESTAB                0                     0
                                        10.0.2.15:ssh
                                    10.0.2.2:48234
#       olia rto:204 rtt:0.342/0.082 ato:40 mss:1460 pmtu:1500 rcvmss
    :536 advmss:1460 cwnd:10 bytes_acked:15121 bytes_received:2512
    segs_out:43 segs_in:63 data_segs_out:34 data_segs_in:28 send 341.5
    Mbps lastsnd:13508 lastrcv:13512 lastack:13508 pacing_rate 681.5
    Mbps delivery_rate 192.1Mbps app_limited busy:36ms rcv_space:14600
     rcv_ssthresh:64076 minrtt:0.106

import prometheus_client as prom
```

```
12 import random
13 import time
14 import os
15
16 DEBUG=False
17
18 if __name__ == '__main__':
19
20    tcp_info_rtt_gauge = prom.Gauge('tcp_info_rtt_gauge', 'TCP mean RTT
       ', ['source', 'destination'])
21    tcp_info_rttstddev_gauge = prom.Gauge('tcp_info_rttstddev_gauge', '
       TCP standard deviation of RTT', ['source', 'destination'])
22    tcp_info_minrtt_gauge = prom.Gauge('tcp_info_minrtt_gauge', 'TCP
       minimum RTT', ['source', 'destination'])
23    tcp_info_cwnd_gauge = prom.Gauge('tcp_info_cwnd_gauge', 'TCP
       congestion window', ['source', 'destination'])
24    tcp_info_segs_in_gauge = prom.Gauge('tcp_info_segs_in_gauge', 'TCP
       segments in', ['source', 'destination'])
25    tcp_info_segs_out_gauge = prom.Gauge('tcp_info_segs_out_gauge', '
       TCP segments out', ['source', 'destination'])
26    tcp_info_data_segs_in_gauge = prom.Gauge('
       tcp_info_data_segs_in_gauge', 'TCP data segments in', ['source'
       , 'destination'])
27    tcp_info_data_segs_out_gauge = prom.Gauge('
       tcp_info_data_segs_out_gauge', 'TCP data segments out', ['
       source', 'destination'])
28    tcp_info_bytes_acked_gauge = prom.Gauge('tcp_info_bytes_acked_gauge
       ', 'TCP bytes acknowledged', ['source', 'destination'])
29    tcp_info_bytes_received_gauge = prom.Gauge('
       tcp_info_bytes_received_gauge', 'TCP bytes received', ['source'
       , 'destination'])
30    tcp_info_pacing_rate_gauge = prom.Gauge('tcp_info_pacing_rate_gauge
       ', 'TCP pacing rate', ['source', 'destination'])
31    tcp_info_delivery_rate_gauge = prom.Gauge('
       tcp_info_delivery_rate_gauge', 'TCP delivery rate', ['source',
       'destination'])
32
33    prom.start_http_server(8080)
34
35 # Go to sleep until Ctrl+C!
36 try:
37    # *** ADD THIS PERIOD AS COMMAND LINE PARAMETER ***
38    wait_time = 1.0
39    while True:
40      # Execute "ss -i -t" and save results
41
42 #     stream = os.popen('ss -i -t')
43      stream = os.popen('sudo ip netns exec MPTCPns ss -i -t')
44      output = stream.read()
45      # Parse output from "ss -i -t"
46      i = 0
47      previousFirstWord = ""
48      for line in output.splitlines():
49        i = i + 1
50        if DEBUG: print("Line " + str(i) + ": " + line)
51        words = line.split()
52        if DEBUG: print("No. words: " + str(len(words)))
53
54        # Check if previous line was the header (field titles) of an
           established flow
55        # If so, check each word to save stats for Prometheus
56        if previousFirstWord == "ESTAB":
```

```
57          for j in range(len(words)):
58             if DEBUG: print("word[" + str(j) + "]: " + words[j])
59             splittedWord = words[j].split(":")
60             if len(splittedWord) >= 2:
61                if DEBUG: print("splittedWord[0]: " + splittedWord[0])
62                if DEBUG: print("splittedWord[1]: " + splittedWord[1])
63                # Save results for Prometheus
64                if splittedWord[0] == "rtt":
65                  tmpSplittedWord = splittedWord[1].split("/")
66                  if DEBUG: print("tmpSplittedWord[0]: " +
                        tmpSplittedWord[0]) # Mean
67                  if DEBUG: print("tmpSplittedWord[1]: " +
                        tmpSplittedWord[1]) # Standard deviation
68                  tcp_info_rtt_gauge.labels(sourceStr, destinationStr).
                        set(tmpSplittedWord[0])
69                  tcp_info_rttstddev_gauge.labels(sourceStr,
                        destinationStr).set(tmpSplittedWord[1])
70                elif splittedWord[0] == "minrtt":
71                  tcp_info_minrtt_gauge.labels(sourceStr,
                        destinationStr).set(splittedWord[1])
72                elif splittedWord[0] == "cwnd":
73                  tcp_info_cwnd_gauge.labels(sourceStr, destinationStr)
                        .set(splittedWord[1])
74                elif splittedWord[0] == "segs_in":
75                  tcp_info_segs_in_gauge.labels(sourceStr,
                        destinationStr).set(splittedWord[1])
76                elif splittedWord[0] == "segs_out":
77                  tcp_info_segs_out_gauge.labels(sourceStr,
                        destinationStr).set(splittedWord[1])
78                elif splittedWord[0] == "data_segs_in":
79                  tcp_info_segs_in_gauge.labels(sourceStr,
                        destinationStr).set(splittedWord[1])
80                elif splittedWord[0] == "data_segs_out":
81                  tcp_info_segs_out_gauge.labels(sourceStr,
                        destinationStr).set(splittedWord[1])
82                elif splittedWord[0] == "bytes_acked":
83                  tcp_info_bytes_acked_gauge.labels(sourceStr,
                        destinationStr).set(splittedWord[1])
84                elif splittedWord[0] == "bytes_received":
85                  tcp_info_bytes_received_gauge.labels(sourceStr,
                        destinationStr).set(splittedWord[1])
86                elif splittedWord[0] == "pacing_rate":
87                  tcp_info_pacing_rate_gauge.labels(sourceStr,
                        destinationStr).set(splittedWord[1])
88                elif splittedWord[0] == "delivery_rate":
89                  tcp_info_delivery_rate_gauge.labels(sourceStr,
                        destinationStr).set(splittedWord[1])
90
91          if len(words)>=5:
92            if words[0] == "ESTAB":
93               previousFirstWord = words[0]
94               sourceStr = words[3]
95               destinationStr = words[4]
96          else:
97            previousFirstWord = ""
98
99       time.sleep(wait_time)
100
101 except KeyboardInterrupt:
102    pass
```

Apéndice H. control_manual.py

A continuación se muestra el contenido del script `control_manual.py`:

Listing 1: Script para el control manual del AWS DeepRacer

```
import json
import time
import argparse
import subprocess
import threading
import awsdeepracer_control

#client = awsdeepracer_control.Client(password="ctlbjuPB", ip="
    localhost")

client = awsdeepracer_control.Client(password="ctlbjuPB", ip="
    localhost")

def run_command(command):
    if command:
        print(f"Ejecutando comando: {command}")
        result = subprocess.run(command, shell=True, capture_output=
            True, text=True)
        print(result.stdout)
        if result.stderr:
            print(result.stderr)

def run_command_thread(command):
    if command:
        thread = threading.Thread(target=run_command, args=(command,))
        thread.start()
        thread.join()
        print("entra")

def main(route_file):
    with open(route_file, 'r') as file:
        route_steps = json.load(file)

    client.set_manual_mode()
    client.start_car()

    for step in route_steps:
        action = step["action"]
        params = step.get("params", {})
        message = step.get("message", "")
        duration = step.get("duration", 0)
        command = step.get("command")
        # Imprimir la descripcion del paso
        print(step["description"])

        if action == "move":
            angle = params.get("steering_angle", 0)
            throttle = params.get("throttle", 0)
            max_speed = params.get("max_speed", 1)
            run_command_thread(command)
            # Ejecutar comandos durante la parada
            if message:
                print(message)
            client.move(angle, throttle, max_speed)
            time.sleep(duration)
```

```
53          elif action == "stop":
54              client.stop_car()
55              run_command_thread(command)
56              time.sleep(duration)
57              # Ejecutar comandos durante la parada
58              if message:
59                  print(message)
60              client.start_car()
61          elif action == "start":
62              client.start_car()
63
64      client.stop_car()
65
66  if __name__ == "__main__":
67      parser = argparse.ArgumentParser(description='Car Control Script')
68      parser.add_argument('-r', '--route', required=True, help='Path to
            the JSON route file')
69      args = parser.parse_args()
70
71      main(args.route)
```

Apéndice I. ruta.json

A continuación se muestra el contenido del archivo `ruta.json`:

```
1  [
2          {
3                  "description": "Mover en linea recta -
                       Paso 1",
4                  "action": "move",
5                  "duration": 5,
6                  "params": {
7                          "steering_angle": 0.0,
8                          "throttle": 0.7,
9                          "max_speed": 0.6
10                 }
11         },
12         {
13                 "description": "Parada 1: Ejecutar ping e
                       iperf3",
14                 "action": "stop",
15                 "duration": 10,
16                 "message": "Ejecutando ping en parada 1",
17                 "command": "ping -c 4 192.168.222.1 | tee
                       -a ping.log"
18         },
19         {
20                 "description": "Mover en linea recta -
                       Paso 2",
21                 "action": "move",
22                 "duration": 5,
```

```
23                  "params": {
24                          "steering_angle": 0.0,
25                          "throttle": 0.7,
26                          "max_speed": 0.6
27                  }
28          },
29          {
30                  "description": "Parada 2: Ejecutar ping e
                        iperf3",
31                  "action": "stop",
32                  "duration": 10,
33                  "message": "Ejecutando iperf3 en parada 2
                        ",
34                  "command": "iperf3 -c 192.168.222.1 -t 10
                        | tee -a iperf.log"
35          },
36          {
37                  "description": "Mover en linea recta -
                        Paso 3",
38                  "action": "move",
39                  "duration": 5,
40                  "params": {
41                          "steering_angle": 0.0,
42                          "throttle": 0.7,
43                          "max_speed": 0.6
44                  }
45          },
46          {
47                  "description": "Parada 3: Ejecutar ping e
                        iperf3",
48                  "action": "stop",
49                  "duration": 10,
50                  "message": "Ejecutando ping en parada 3",
51                  "command": "ping -c 4 192.168.222.1 | tee
                        -a ping.log"
52          },
53          {
54                  "description": "Mover en linea recta -
                        Paso 4",
55                  "action": "move",
56                  "duration": 5,
57                  "params": {
58                          "steering_angle": 0.0,
59                          "throttle": 0.7,
60                          "max_speed": 0.6
61                  }
62          },
63          {
64                  "description": "Parada 4: Ejecutar ping e
```

```
                    iperf3",
                "action": "stop",
                "duration": 10,
                "message": "Ejecutando ping en parada 4",
                "command": "ping -c 4 192.168.222.1 | tee
                    -a ping.log"
        },
        {
                "description": "Mover en linea recta -
                   Paso 5",
                "action": "move",
                "duration": 5,
                "params": {
                        "steering_angle": 0.0,
                        "throttle": 0.7,
                        "max_speed": 0.6
                }
        },
        {
                "description": "Parada 5: Ejecutar ping e
                    iperf3",
                "action": "stop",
                "duration": 10,
                "message": "Ejecutando ping en parada 5",
                "command": "ping -c 4 192.168.222.1 | tee
                    -a ping.log"
        }
]
```

Apéndice J. prometheusModificado.yml

A continuación se muestra el contenido del archivo `prometheusModificado.yml`:

```
# my global config
global:
  scrape_interval:     1s # Set the scrape interval to every 15
      seconds. Default is every 1 minute.
  evaluation_interval: 1s # Evaluate rules every 15 seconds.
     The default is every 1 minute.
  # scrape_timeout is set to the global default (10s).

# Alertmanager configuration
alerting:
  alertmanagers:
  - static_configs:
    - targets:
      # - alertmanager:9093

```

```
# Load rules once and periodically evaluate them according to the global 'evaluation_interval'.
rule_files:
  # - "first_rules.yml"
  # - "second_rules.yml"

# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries scraped from this config.
  - job_name: 'prometheus'

    # metrics_path defaults to '/metrics'
    # scheme defaults to 'http'.

    static_configs:
    - targets: ['localhost:9090']

  - job_name: node
    static_configs:
    - targets: ['localhost:9100']

  - job_name: ss
    static_configs:
    - targets: ['localhost:18080']
```

Apéndice K. ss_exporterModificado.py

A continuación se muestra el contenido del script `ss_exporterModificado.py`:

```
# SS exporter for Prometheus (i.e. RTT, CWND, ... TCP flow parameters)


# See https://sysdig.com/blog/prometheus-metrics/

# Example of output from ss -i -t:
#State               Recv-Q               Send-Q
                                  Local Address:Port
                                  Peer Address:Port
#ESTAB               0                    0
                                        10.0.2.15:ssh
                                     10.0.2.2:48234
#       olia rto:204 rtt:0.342/0.082 ato:40 mss:1460 pmtu:1500 rcvmss:536 advmss:1460 cwnd:10 bytes_acked:15121 bytes_received:2512 segs_out:43 segs_in:63 data_segs_out:34 data_segs_in:28 send 341.5Mbps lastsnd:13508 lastrcv:13512 lastack:13508 pacing_rate 681.5Mbps delivery_rate 192.1Mbps app_limited busy:36ms rcv_space:14600 rcv_ssthresh:64076 minrtt:0.106

import prometheus_client as prom
import random
import time
```

```
14 import os
15
16 DEBUG=False
17
18 if __name__ == '__main__':
19
20    tcp_info_rtt_gauge = prom.Gauge('tcp_info_rtt_gauge', 'TCP mean RTT
       ', ['source', 'destination'])
21    tcp_info_rttstddev_gauge = prom.Gauge('tcp_info_rttstddev_gauge', '
       TCP standard deviation of RTT', ['source', 'destination'])
22    tcp_info_minrtt_gauge = prom.Gauge('tcp_info_minrtt_gauge', 'TCP
       minimum RTT', ['source', 'destination'])
23    tcp_info_cwnd_gauge = prom.Gauge('tcp_info_cwnd_gauge', 'TCP
       congestion window', ['source', 'destination'])
24    tcp_info_segs_in_gauge = prom.Gauge('tcp_info_segs_in_gauge', 'TCP
       segments in', ['source', 'destination'])
25    tcp_info_segs_out_gauge = prom.Gauge('tcp_info_segs_out_gauge', '
       TCP segments out', ['source', 'destination'])
26    tcp_info_data_segs_in_gauge = prom.Gauge('
       tcp_info_data_segs_in_gauge', 'TCP data segments in', ['source'
       , 'destination'])
27    tcp_info_data_segs_out_gauge = prom.Gauge('
       tcp_info_data_segs_out_gauge', 'TCP data segments out', ['
       source', 'destination'])
28    tcp_info_bytes_acked_gauge = prom.Gauge('tcp_info_bytes_acked_gauge
       ', 'TCP bytes acknowledged', ['source', 'destination'])
29    tcp_info_bytes_received_gauge = prom.Gauge('
       tcp_info_bytes_received_gauge', 'TCP bytes received', ['source'
       , 'destination'])
30    tcp_info_pacing_rate_gauge = prom.Gauge('tcp_info_pacing_rate_gauge
       ', 'TCP pacing rate', ['source', 'destination'])
31    tcp_info_delivery_rate_gauge = prom.Gauge('
       tcp_info_delivery_rate_gauge', 'TCP delivery rate', ['source',
       'destination'])
32
33    prom.start_http_server(18080)
34
35 # Go to sleep until Ctrl+C!
36 try:
37    # *** ADD THIS PERIOD AS COMMAND LINE PARAMETER ***
38    wait_time = 1.0
39    while True:
40      # Execute "ss -i -t" and save results
41
42      stream = os.popen('sudo ss -i -t')
43 #     stream = os.popen('sudo ip netns exec MPTCPns ss -i -t')
44      output = stream.read()
45      # Parse output from "ss -i -t"
46      i = 0
47      previousFirstWord = ""
48      for line in output.splitlines():
49        i = i + 1
50        if DEBUG: print("Line " + str(i) + ": " + line)
51        words = line.split()
52        if DEBUG: print("No. words: " + str(len(words)))
53
54        # Check if previous line was the header (field titles) of an
           established flow
55        # If so, check each word to save stats for Prometheus
56        if previousFirstWord == "ESTAB":
57          for j in range(len(words)):
58             if DEBUG: print("word[" + str(j) + "]: " + words[j])
```

```
59              splittedWord = words[j].split(":")
60              if len(splittedWord) >= 2:
61                 if DEBUG: print("splittedWord[0]: " + splittedWord[0])
62                 if DEBUG: print("splittedWord[1]: " + splittedWord[1])
63                 # Save results for Prometheus
64                 if splittedWord[0] == "rtt":
65                   tmpSplittedWord = splittedWord[1].split("/")
66                   if DEBUG: print("tmpSplittedWord[0]: " +
                         tmpSplittedWord[0]) # Mean
67                   if DEBUG: print("tmpSplittedWord[1]: " +
                         tmpSplittedWord[1]) # Standard deviation
68                   tcp_info_rtt_gauge.labels(sourceStr, destinationStr).
                         set(tmpSplittedWord[0])
69                   tcp_info_rttstddev_gauge.labels(sourceStr,
                         destinationStr).set(tmpSplittedWord[1])
70                 elif splittedWord[0] == "minrtt":
71                   tcp_info_minrtt_gauge.labels(sourceStr,
                         destinationStr).set(splittedWord[1])
72                 elif splittedWord[0] == "cwnd":
73                   tcp_info_cwnd_gauge.labels(sourceStr, destinationStr)
                         .set(splittedWord[1])
74                 elif splittedWord[0] == "segs_in":
75                   tcp_info_segs_in_gauge.labels(sourceStr,
                         destinationStr).set(splittedWord[1])
76                 elif splittedWord[0] == "segs_out":
77                   tcp_info_segs_out_gauge.labels(sourceStr,
                         destinationStr).set(splittedWord[1])
78                 elif splittedWord[0] == "data_segs_in":
79                   tcp_info_segs_in_gauge.labels(sourceStr,
                         destinationStr).set(splittedWord[1])
80                 elif splittedWord[0] == "data_segs_out":
81                   tcp_info_segs_out_gauge.labels(sourceStr,
                         destinationStr).set(splittedWord[1])
82                 elif splittedWord[0] == "bytes_acked":
83                   tcp_info_bytes_acked_gauge.labels(sourceStr,
                         destinationStr).set(splittedWord[1])
84                 elif splittedWord[0] == "bytes_received":
85                   tcp_info_bytes_received_gauge.labels(sourceStr,
                         destinationStr).set(splittedWord[1])
86                 elif splittedWord[0] == "pacing_rate":
87                   tcp_info_pacing_rate_gauge.labels(sourceStr,
                         destinationStr).set(splittedWord[1])
88                 elif splittedWord[0] == "delivery_rate":
89                   tcp_info_delivery_rate_gauge.labels(sourceStr,
                         destinationStr).set(splittedWord[1])
90
91         if len(words)>=5:
92           if words[0] == "ESTAB":
93              previousFirstWord = words[0]
94              sourceStr = words[3]
95              destinationStr = words[4]
96         else:
97           previousFirstWord = ""
98
99       time.sleep(wait_time)
100
101 except KeyboardInterrupt:
102    pass
```

Apéndice L. start_statsModificado.sh

A continuación se muestra el contenido del script `start_statsModificado.sh`:

```
#!/bin/bash



sudo systemctl start grafana-server

sudo iptables -A INPUT -p tcp -m tcp --dport 3000 -j ACCEPT

cd ~/vagrant/stats/node_exporter-1.1.2.linux-amd64
./node_exporter &

cd ~/vagrant/stats/prometheus-2.27.1.linux-amd64
./prometheus --config.file=./prometheus.yml &

cd ~/vagrant/stats/multitechnology_testbed_v0_stats
python3 ./ss_exporter.py &
```

Apéndice M. stop_statsModificado

A continuación se muestra el contenido del script `stop_statsModificado`:

```
#!/bin/bash



# Stop Grafana
sudo systemctl stop grafana-server

sudo iptables -D INPUT -p tcp -m tcp --dport 3000 -j ACCEPT

# Stop node_exporter
PID=`sudo pgrep node_exporter`
if [[ $PID != "" ]]; then sudo kill -9 $PID; fi

# Stop prometheus
PID=`sudo pgrep prometheus`
if [[ $PID != "" ]]; then sudo kill -9 $PID; fi

# Stop ss_exporter
   # Get PID (process name is python...)
PID=`sudo netstat -tulpn | grep 8080 | rev | cut -d"/" -f 2 | cut -d"
    " -f 1 | rev`
if [[ $PID != "" ]]; then sudo kill -9 $PID; fi
```

Apéndice N. AWSDeepRacer5G+WIFI-1716887935477.json

A continuación se muestra el contenido del archivo `AWS DeepRacer 5G + WIFI-1716887935477.json`:

```
{
  "annotations": {
    "list": [
      {
        "builtIn": 1,
        "datasource": "-- Grafana --",
        "enable": true,
        "hide": true,
        "iconColor": "rgba(0, 211, 255, 1)",
        "name": "Annotations & Alerts",
        "type": "dashboard"
      }
    ]
  },
  "editable": true,
  "gnetId": null,
  "graphTooltip": 0,
  "id": 1,
  "links": [],
  "panels": [
    {
      "datasource": null,
      "description": "",
      "fieldConfig": {
        "defaults": {
          "color": {
            "mode": "palette-classic"
          },
          "custom": {
            "axisLabel": "",
            "axisPlacement": "auto",
            "barAlignment": 0,
            "drawStyle": "line",
            "fillOpacity": 20,
            "gradientMode": "none",
            "hideFrom": {
              "legend": false,
              "tooltip": false,
              "viz": false
            },
            "lineInterpolation": "smooth",
            "lineWidth": 1,
            "pointSize": 5,
            "scaleDistribution": {
              "type": "linear"
            },
            "showPoints": "auto",
            "spanNulls": false,
```

```
            "stacking": {
              "group": "A",
              "mode": "normal"
            },
            "thresholdsStyle": {
              "mode": "off"
            }
          },
          "mappings": [],
          "thresholds": {
            "mode": "absolute",
            "steps": [
              {
                "color": "green",
                "value": null
              },
              {
                "color": "red",
                "value": 80
              }
            ]
          },
          "unit": "bps"
        },
        "overrides": [
          {
            "matcher": {
              "id": "byName",
              "options": "5G NR"
            },
            "properties": [
              {
                "id": "color",
                "value": {
                  "fixedColor": "red",
                  "mode": "fixed"
                }
              }
            ]
          },
          {
            "matcher": {
              "id": "byName",
              "options": "Wi-Fi 6"
            },
            "properties": [
              {
                "id": "color",
                "value": {
```

```
                "fixedColor": "green",
                "mode": "fixed"
              }
            }
          ]
        },
        {
          "matcher": {
            "id": "byName",
            "options": "WIFI"
          },
          "properties": [
            {
              "id": "color",
              "value": {
                "fixedColor": "blue",
                "mode": "fixed"
              }
            }
          ]
        }
      ]
    },
    "gridPos": {
      "h": 7,
      "w": 12,
      "x": 0,
      "y": 0
    },
    "id": 4,
    "options": {
      "legend": {
        "calcs": [],
        "displayMode": "list",
        "placement": "bottom"
      },
      "tooltip": {
        "mode": "single"
      }
    },
    "targets": [
      {
        "exemplar": true,
        "expr": "8*rate(
           node_network_receive_bytes_total{device=~\"
           usb2|mlan0|wlo1\"}[3s])",
        "hide": true,
        "interval": "",
        "legendFormat": "{{device}}",
```

```
145             "refId": "A"
146           },
147           {
148             "exemplar": true,
149             "expr": "8*rate(
                  node_network_receive_bytes_total{device=~\"
                  usb2\"}[3s])",
150             "hide": false,
151             "interval": "",
152             "legendFormat": "{{device}}",
153             "refId": "B"
154           },
155           {
156             "exemplar": true,
157             "expr": "8*rate(
                  node_network_receive_bytes_total{device=~\"
                  wlo1\"}[3s])",
158             "hide": false,
159             "interval": "",
160             "legendFormat": "{{device}}",
161             "refId": "C"
162           },
163           {
164             "exemplar": true,
165             "expr": "8*rate(
                  node_network_receive_bytes_total{device=~\"
                  mlan0\"}[3s])",
166             "hide": false,
167             "interval": "",
168             "legendFormat": "{{device}}",
169             "refId": "D"
170           }
171         ],
172         "title": "Cumulative traffic received from server",
173         "transformations": [
174           {
175             "id": "renameByRegex",
176             "options": {
177               "regex": "usb2",
178               "renamePattern": "5G NR"
179             }
180           },
181           {
182             "id": "renameByRegex",
183             "options": {
184               "regex": "mlan0",
185               "renamePattern": "WIFI"
186             }
187           },
```

```
        {
          "id": "renameByRegex",
          "options": {
            "regex": "wlo1",
            "renamePattern": "Wi-Fi 6"
          }
        }
      ],
      "type": "timeseries"
    },
    {
      "datasource": null,
      "description": "",
      "fieldConfig": {
        "defaults": {
          "color": {
            "mode": "palette-classic"
          },
          "custom": {
            "axisLabel": "",
            "axisPlacement": "auto",
            "barAlignment": 0,
            "drawStyle": "line",
            "fillOpacity": 20,
            "gradientMode": "none",
            "hideFrom": {
              "legend": false,
              "tooltip": false,
              "viz": false
            },
            "lineInterpolation": "smooth",
            "lineWidth": 1,
            "pointSize": 5,
            "scaleDistribution": {
              "type": "linear"
            },
            "showPoints": "auto",
            "spanNulls": false,
            "stacking": {
              "group": "A",
              "mode": "normal"
            },
            "thresholdsStyle": {
              "mode": "off"
            }
          },
          "mappings": [],
          "thresholds": {
            "mode": "absolute",
```

```
            "steps": [
              {
                "color": "green",
                "value": null
              },
              {
                "color": "red",
                "value": 80
              }
            ]
          },
          "unit": "bps"
        },
        "overrides": [
          {
            "matcher": {
              "id": "byName",
              "options": "5G NR"
            },
            "properties": [
              {
                "id": "color",
                "value": {
                  "fixedColor": "red",
                  "mode": "fixed"
                }
              }
            ]
          },
          {
            "matcher": {
              "id": "byName",
              "options": "Wi-Fi 6"
            },
            "properties": [
              {
                "id": "color",
                "value": {
                  "fixedColor": "green",
                  "mode": "fixed"
                }
              }
            ]
          },
          {
            "matcher": {
              "id": "byName",
              "options": "WIFI"
            },
```

```
            "properties": [
              {
                "id": "color",
                "value": {
                  "fixedColor": "blue",
                  "mode": "fixed"
                }
              }
            ]
          }
        ]
      },
      "gridPos": {
        "h": 7,
        "w": 12,
        "x": 12,
        "y": 0
      },
      "id": 20,
      "options": {
        "legend": {
          "calcs": [],
          "displayMode": "list",
          "placement": "bottom"
        },
        "tooltip": {
          "mode": "single"
        }
      },
      "targets": [
        {
          "exemplar": true,
          "expr": "8*rate(
             node_network_transmit_bytes_total{device
             =~\"usb2|mlan0|wlo1\"}[3s])",
          "hide": true,
          "interval": "",
          "legendFormat": "{{device}}",
          "refId": "A"
        },
        {
          "exemplar": true,
          "expr": "8*rate(
             node_network_transmit_bytes_total{device
             =~\"usb2\"}[3s])",
          "hide": false,
          "interval": "",
          "legendFormat": "{{device}}",
          "refId": "B"
```

```
        },
        {
          "exemplar": true,
          "expr": "8*rate(
             node_network_transmit_bytes_total{device
             =~\"wlo1\"}[3s])",
          "hide": false,
          "interval": "",
          "legendFormat": "{{device}}",
          "refId": "C"
        },
        {
          "exemplar": true,
          "expr": "8*rate(
             node_network_transmit_bytes_total{device
             =~\"mlan0\"}[3s])",
          "hide": false,
          "interval": "",
          "legendFormat": "{{device}}",
          "refId": "D"
        }
      ],
      "title": "Cumulative traffic sent to server",
      "transformations": [
        {
          "id": "renameByRegex",
          "options": {
            "regex": "usb2",
            "renamePattern": "5G NR"
          }
        },
        {
          "id": "renameByRegex",
          "options": {
            "regex": "mlan0",
            "renamePattern": "WIFI"
          }
        },
        {
          "id": "renameByRegex",
          "options": {
            "regex": "wlo1",
            "renamePattern": "Wi-Fi 6"
          }
        }
      ],
      "type": "timeseries"
    },
    {
```

```
      "datasource": null,
      "description": "",
      "fieldConfig": {
        "defaults": {
          "color": {
            "mode": "palette-classic"
          },
          "custom": {
            "axisLabel": "",
            "axisPlacement": "auto",
            "barAlignment": 0,
            "drawStyle": "line",
            "fillOpacity": 0,
            "gradientMode": "none",
            "hideFrom": {
              "legend": false,
              "tooltip": false,
              "viz": false
            },
            "lineInterpolation": "smooth",
            "lineWidth": 1,
            "pointSize": 5,
            "scaleDistribution": {
              "type": "linear"
            },
            "showPoints": "auto",
            "spanNulls": false,
            "stacking": {
              "group": "A",
              "mode": "none"
            },
            "thresholdsStyle": {
              "mode": "off"
            }
          },
          "mappings": [],
          "thresholds": {
            "mode": "absolute",
            "steps": [
              {
                "color": "green",
                "value": null
              },
              {
                "color": "red",
                "value": 80
              }
            ]
          },
```

```
            "unit": "bps"
          },
          "overrides": [
            {
              "matcher": {
                "id": "byName",
                "options": "5G NR"
              },
              "properties": [
                {
                  "id": "color",
                  "value": {
                    "fixedColor": "red",
                    "mode": "fixed"
                  }
                }
              ]
            },
            {
              "matcher": {
                "id": "byName",
                "options": "Wi-Fi 6"
              },
              "properties": [
                {
                  "id": "color",
                  "value": {
                    "fixedColor": "green",
                    "mode": "fixed"
                  }
                }
              ]
            },
            {
              "matcher": {
                "id": "byName",
                "options": "WIFI"
              },
              "properties": [
                {
                  "id": "color",
                  "value": {
                    "fixedColor": "blue",
                    "mode": "fixed"
                  }
                }
              ]
            }
          ]
```

```
474         },
475         "gridPos": {
476           "h": 7,
477           "w": 12,
478           "x": 0,
479           "y": 7
480         },
481         "id": 19,
482         "options": {
483           "legend": {
484             "calcs": [],
485             "displayMode": "list",
486             "placement": "bottom"
487           },
488           "tooltip": {
489             "mode": "single"
490           }
491         },
492         "targets": [
493           {
494             "exemplar": true,
495             "expr": "8*rate(
                  node_network_receive_bytes_total{device=~\"
                  usb2|mlan0|wlo1\"}[3s])",
496             "hide": true,
497             "interval": "",
498             "legendFormat": "{{device}}",
499             "refId": "A"
500           },
501           {
502             "exemplar": true,
503             "expr": "8*rate(
                  node_network_receive_bytes_total{device=~\"
                  usb2\"}[3s])",
504             "hide": false,
505             "interval": "",
506             "legendFormat": "{{device}}",
507             "refId": "B"
508           },
509           {
510             "exemplar": true,
511             "expr": "8*rate(
                  node_network_receive_bytes_total{device=~\"
                  wlo1\"}[3s])",
512             "hide": false,
513             "interval": "",
514             "legendFormat": "{{device}}",
515             "refId": "C"
516           },
```

```
517           {
518             "exemplar": true,
519             "expr": "8*rate(
                   node_network_receive_bytes_total{device=~\"
                   mlan0\"}[3s])",
520             "hide": false,
521             "interval": "",
522             "legendFormat": "{{device}}",
523             "refId": "D"
524           }
525         ],
526         "title": "Traffic received from server",
527         "transformations": [
528           {
529             "id": "renameByRegex",
530             "options": {
531               "regex": "usb2",
532               "renamePattern": "5G NR"
533             }
534           },
535           {
536             "id": "renameByRegex",
537             "options": {
538               "regex": "mlan0",
539               "renamePattern": "WIFI"
540             }
541           },
542           {
543             "id": "renameByRegex",
544             "options": {
545               "regex": "wlo1",
546               "renamePattern": "Wi-Fi 6"
547             }
548           }
549         ],
550         "type": "timeseries"
551       },
552       {
553         "datasource": null,
554         "description": "",
555         "fieldConfig": {
556           "defaults": {
557             "color": {
558               "mode": "palette-classic"
559             },
560             "custom": {
561               "axisLabel": "",
562               "axisPlacement": "auto",
563               "barAlignment": 0,
```

```
            "drawStyle": "line",
            "fillOpacity": 0,
            "gradientMode": "none",
            "hideFrom": {
              "legend": false,
              "tooltip": false,
              "viz": false
            },
            "lineInterpolation": "smooth",
            "lineWidth": 1,
            "pointSize": 5,
            "scaleDistribution": {
              "type": "linear"
            },
            "showPoints": "auto",
            "spanNulls": false,
            "stacking": {
              "group": "A",
              "mode": "none"
            },
            "thresholdsStyle": {
              "mode": "off"
            }
          },
          "mappings": [],
          "thresholds": {
            "mode": "absolute",
            "steps": [
              {
                "color": "green",
                "value": null
              },
              {
                "color": "red",
                "value": 80
              }
            ]
          },
          "unit": "bps"
        },
        "overrides": [
          {
            "matcher": {
              "id": "byName",
              "options": "5G NR"
            },
            "properties": [
              {
                "id": "color",
```

```
                "value": {
                  "fixedColor": "red",
                  "mode": "fixed"
                }
              }
            ]
          },
          {
            "matcher": {
              "id": "byName",
              "options": "Wi-Fi 6"
            },
            "properties": [
              {
                "id": "color",
                "value": {
                  "fixedColor": "green",
                  "mode": "fixed"
                }
              }
            ]
          },
          {
            "matcher": {
              "id": "byName",
              "options": "WIFI"
            },
            "properties": [
              {
                "id": "color",
                "value": {
                  "fixedColor": "blue",
                  "mode": "fixed"
                }
              }
            ]
          }
        ]
      },
      "gridPos": {
        "h": 7,
        "w": 12,
        "x": 12,
        "y": 7
      },
      "id": 18,
      "options": {
        "legend": {
          "calcs": [],
```

```
662           "displayMode": "list",
663           "placement": "bottom"
664         },
665         "tooltip": {
666           "mode": "single"
667         }
668       },
669       "targets": [
670         {
671           "exemplar": false,
672           "expr": "8*rate(
                node_network_transmit_bytes_total{device
                =~\"usb2|mlan0|wlo1\"}[3s])",
673           "hide": true,
674           "interval": "",
675           "legendFormat": "{{device}}",
676           "refId": "A"
677         },
678         {
679           "exemplar": true,
680           "expr": "8*rate(
                node_network_transmit_bytes_total{device
                =~\"usb2\"}[3s])",
681           "hide": false,
682           "interval": "",
683           "legendFormat": "{{device}}",
684           "refId": "B"
685         },
686         {
687           "exemplar": true,
688           "expr": "8*rate(
                node_network_transmit_bytes_total{device
                =~\"wlo1\"}[3s])",
689           "hide": false,
690           "interval": "",
691           "legendFormat": "{{device}}",
692           "refId": "C"
693         },
694         {
695           "exemplar": true,
696           "expr": "8*rate(
                node_network_transmit_bytes_total{device
                =~\"mlan0\"}[3s])",
697           "hide": false,
698           "interval": "",
699           "legendFormat": "{{device}}",
700           "refId": "D"
701         }
702       ],
```

```
      "title": "Traffic sent to server",
      "transformations": [
        {
          "id": "renameByRegex",
          "options": {
            "regex": "usb2",
            "renamePattern": "5G NR"
          }
        },
        {
          "id": "renameByRegex",
          "options": {
            "regex": "mlan0",
            "renamePattern": "WIFI"
          }
        },
        {
          "id": "renameByRegex",
          "options": {
            "regex": "wlo1",
            "renamePattern": "Wi-Fi 6"
          }
        }
      ],
      "type": "timeseries"
    },
    {
      "datasource": null,
      "description": "",
      "fieldConfig": {
        "defaults": {
          "color": {
            "mode": "palette-classic"
          },
          "custom": {
            "axisLabel": "",
            "axisPlacement": "auto",
            "barAlignment": 0,
            "drawStyle": "line",
            "fillOpacity": 0,
            "gradientMode": "none",
            "hideFrom": {
              "legend": false,
              "tooltip": false,
              "viz": false
            },
            "lineInterpolation": "smooth",
            "lineWidth": 1,
            "pointSize": 5,
```

```
            "scaleDistribution": {
              "type": "linear"
            },
            "showPoints": "auto",
            "spanNulls": false,
            "stacking": {
              "group": "A",
              "mode": "none"
            },
            "thresholdsStyle": {
              "mode": "off"
            }
          },
          "mappings": [],
          "thresholds": {
            "mode": "absolute",
            "steps": [
              {
                "color": "green",
                "value": null
              },
              {
                "color": "red",
                "value": 80
              }
            ]
          },
          "unit": "ms"
        },
        "overrides": []
      },
      "gridPos": {
        "h": 7,
        "w": 24,
        "x": 0,
        "y": 14
      },
      "id": 6,
      "options": {
        "legend": {
          "calcs": [],
          "displayMode": "list",
          "placement": "bottom"
        },
        "tooltip": {
          "mode": "single"
        }
      },
      "targets": [
```

```
        {
          "exemplar": true,
          "expr": "tcp_info_rtt_gauge",
          "interval": "",
          "legendFormat": "{{source}}",
          "refId": "A"
        }
      ],
      "title": "Mean RTT to MPTCP proxy (5G-CLARITY
         scheduler)",
      "transformations": [
        {
          "id": "renameByRegex",
          "options": {
            "regex": "10.1.1.1(.*)",
            "renamePattern": "5G NR"
          }
        },
        {
          "id": "renameByRegex",
          "options": {
            "regex": "10.1.1.2(.*)",
            "renamePattern": "WIFI"
          }
        },
        {
          "id": "renameByRegex",
          "options": {
            "regex": "10.1.1.3(.*)",
            "renamePattern": "Wi-Fi 6"
          }
        }
      ],
      "type": "timeseries"
    }
  ],
  "refresh": "1s",
  "schemaVersion": 30,
  "style": "dark",
  "tags": [],
  "templating": {
    "list": []
  },
  "time": {
    "from": "now-2m",
    "to": "now"
  },
  "timepicker": {
    "refresh_intervals": [
```

```
        "1s",
        "5s",
        "10s",
        "30s",
        "1m",
        "5m",
        "15m",
        "30m",
        "1h",
        "2h",
        "1d"
      ]
    },
    "timezone": "",
    "title": "AWS DeepRacer 5G + WIFI",
    "uid": "dwR_3zySz",
    "version": 3
}
```

Apéndice Ñ. aaModificado

A continuación se muestra el contenido del archivo `aaModificado`:

```
{"id":1,"uid":"p8vVSgz7z","orgId":1,"name":"Prometheus","
    type":"prometheus","typeName":"Prometheus","
    typeLogoUrl":"public/app/plugins/datasource/
    prometheus/img/prometheus_logo.svg","access":"proxy",
    "url":"http://localhost:9090","password":"","user":""
    ,"database":"","basicAuth":false,"isDefault":true,"
    jsonData":{"httpMethod":"GET","timeInterval":"1s"},"
    readOnly":false}
```

Apéndice O. camara.py

A continuación se muestra el contenido del script `camara.py`:

```
import cv2

video = "http://localhost:8080/stream?topic=/camera_pkg/display_mjpeg"

cap = cv2.VideoCapture(video)

if not cap.isOpened():
    print("no se abre streaming")
    exit()

while True:
```

```
12      ret, frame = cap.read()
13
14      if not ret:
15          print("no se recibe")
16          break
17
18      # Redimensionar el frame a una resoluci n mayor (720x480 en este
            caso)
19      frame_resized = cv2.resize(frame, (720, 480))
20
21      cv2.imshow('streaming video', frame_resized)
22
23      if cv2.waitKey(1) & 0xFF == ord('1'):
24          break
25
26  cap.release()
27  cv2.destroyAllWindows()
```

Bibliografía

[1] J.K. Verma D.S. Duhan, U. Shrivastava. A study on 5g technology and its applications in telecommunications. *IEEE Access*, 2022. Disponible en https://ieeexplore.ieee.org/document/9752402 (último acceso el 20/06/2024).

[2] Fortune Business Insights. Internet of things (iot) market, 2023. Disponible en https://www.fortunebusinessinsights.com (último acceso el 20/06/2024).

[3] Markets and Markets. Augmented reality and virtual reality software market, 2023. Disponible en https://www.marketsandmarkets.com (último acceso el 20/06/2024).

[4] Zion Market Research. Autonomous vehicle market, 2023. Disponible en https://www.zionmarketresearch.com (último acceso el 20/06/2024).

[5] Telcomaglobal. 5g key performance indicators, 2020. Disponible en https://telcomaglobal.com (último acceso el 20/06/2024).

[6] IEEE. Service level agreement. *IEEE Communications Magazine*, 2022. Disponible en https://ieeexplore.ieee.org (último acceso el 20/06/2024).

[7] PPDR. Technologies for public protection and disaster relief, 2020. Disponible en https://ppdrinfo.org (último acceso el 20/06/2024).

[8] Amazon Web Services. Aws deepracer league, 2020. Disponible en https://aws.amazon.com (último acceso el 20/06/2024).

[9] Amazon Web Services. Aws deepracer pricing, 2023. Disponible en https://aws.amazon.com/deepracer/pricing/ (último acceso el 20/06/2024).

[10] Movistar. Xiaomi 12 5g 256gb negro, 2023. Disponible en https://www.movistar.es/moviles/xiaomi-xiaomi-12-5g-256gb-negro/ (último acceso el 20/06/2024).

[11] ASUS. Asus expertcenter d7 sff, 2023. Disponible en https://www.asus.com/commercial-desktops/expertcenter-d7-sff-d700sd/ (último acceso el 20/06/2024).

[12] MediaMarkt. Monitor xiaomi a27i, 2023. Disponible en https://www.mediamarkt.es/ (último acceso el 20/06/2024).

[13] O2. Tarifa estándar, 2023. Disponible en https://www.o2online.es/tarifas/ (último acceso el 20/06/2024).

[14] IEEE. Uavs for 5g networks, 2019. Disponible en https://ieeexplore.ieee.org (último acceso el 20/06/2024).

[15] IEEE. 5g for public safety, 2021. Disponible en https://ieeexplore.ieee.org (último acceso el 20/06/2024).

[16] ScienceDirect. Challenges in uav networks. *Computer Networks*, 2020. Disponible en https://www.sciencedirect.com (último acceso el 20/06/2024).

[17] IEEE. Autonomous vehicles for network quality, 2020. Disponible en https://ieeexplore.ieee.org (último acceso el 20/06/2024).

[18] L. M. Lopez. Tfg: Aws deepracer, 2023. Disponible en https://roboticslaburjc.github.io (último acceso el 20/06/2024).

[19] J. M. Plaza. Tfg: Autonomous drone navigation, 2018. Disponible en https://gsyc.urjc.es (último acceso el 20/06/2024).

[20] LinkedIn. Tfg: Rf transmitter localization, 2023. Disponible en https://www.linkedin.com (último acceso el 20/06/2024).

[21] Amazon Web Services. What is aws?, 2023. Disponible en https://aws.amazon.com (último acceso el 20/06/2024).

[22] ROS. Robot operating system (ros) documentation, 2023. Disponible en https://www.ros.org (último acceso el 20/06/2024).

[23] Prometheus. Prometheus concepts, 2023. Disponible en https://prometheus.io (último acceso el 20/06/2024).

[24] Grafana Labs. Grafana documentation, 2023. Disponible en https://grafana.com (último acceso el 20/06/2024).

[25] Python Software Foundation. Python, 2023. Disponible en https://www.python.org (último acceso el 20/06/2024).

[26] Oracle. What is json?, 2023. Disponible en https://www.oracle.com (último acceso el 20/06/2024).

[27] Git SCM. Git, 2023. Disponible en https://www.git-scm.com (último acceso el 20/06/2024).

[28] GitHub. Github, 2023. Disponible en https://github.com (último acceso el 20/06/2024).

[29] AWS DeepRacer Control. Aws deepracer control, 2023. Disponible en https://github.com (último acceso el 20/06/2024).

[30] DeepRacer Vehicle API. Deepracer vehicle api, 2023. Disponible en https://github.com (último acceso el 20/06/2024).

[31] Siu Hong Leung. Drive your aws deepracer with one line of command, 2023. Disponible en https://www.linkedin.com (último acceso el 20/06/2024).

[32] Jorge Navarro. Multi-technology testbed v0 stats, 2023. Disponible en https://github.com (último acceso el 20/06/2024).

[33] Prometheus. Prometheus concepts, 2023. Disponible en https://prometheus.io (último acceso el 20/06/2024).

[34] AWS DeepRacer. Deepracer boot error, 2023. Disponible en https://repost.aws (último acceso el 20/06/2024).

[35] AWS DeepRacer. Deepracer troubleshooting, 2023. Disponible en https://docs.aws.amazon.com (último acceso el 20/06/2024).

[36] AWS DeepRacer. Unlock dead vehicle batteries, 2023. Disponible en https://docs.aws.amazon.com (último acceso el 20/06/2024).

[37] JQ. Jq - json processor, 2023. Disponible en https://www.baeldung.com (último acceso el 20/06/2024).

[38] Ángel Marín Munuera. Vídeo demostración del experimento en interior, 2024. Disponible en `https://drive.google.com/file/d/1avRLJoACV_k5gG1lLTJVEuDwpbnOf8Qi/view?usp=share_link` (último acceso el 24/06/2024).