



UNIVERSIDAD DE VALLADOLID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE
TELECOMUNICACIÓN

TRABAJO FIN DE MASTER

MASTER EN INGENIERÍA DE TELECOMUNICACIÓN

**Diseño e implementación de un *testbed* de
Edge computing para el soporte de vehículos
conectados**

Autor:

D. Ignacio Royuela González

Tutor:

Dr. D. Juan Carlos Aguado Manzano

VALLADOLID, SEPTIEMBRE 2022

TRABAJO FIN DE MÁSTER

TÍTULO: Diseño e implementación de un *testbed* de *Edge computing* para el soporte de vehículos conectados
AUTOR: **D. Ignacio Royuela González**
TUTOR: **Dr. D. Juan Carlos Aguado Manzano,**
DEPARTAMENTO: **Teoría de la Señal y Comunicaciones e Ingeniería Telemática**

TRIBUNAL

PRESIDENTE: **Dr. Ignacio de Miguel Jiménez**
VOCAL: **Dr. Ramón de la Rosa Steinz**
SECRETARIO: **Dr. Ramón Durán Barroso**
SUPLENTE: **Dr. Patricia Fernández Reguero**
SUPLENTE: **Dr. María Jesús Verdú Pérez**

FECHA:
CALIFICACIÓN:

Resumen de Trabajo Fin de Máster

En este Trabajo Final de Máster se ha elaborado un banco de pruebas de servicios de vehículo conectado a través de redes 4G y *Edge Computing*, permitiendo probar las posibilidades de estas redes con diferentes tipos de servicios. Se implementa nuestra propia red 4G a través de SDR (*Software Defined Radio*) con diferentes métodos y se realiza un análisis, modificación y programación de unos vehículos en miniatura para ser utilizados en el banco. Por otro lado, en este trabajo se implementa y evalúa, gracias al banco de pruebas desarrollado, un caso de uso basado en la conducción autónoma de un vehículo realizando *off-loading* en la red, es decir, descargando las tareas de procesamiento en un servidor externo en tiempo real. Para ello se diseña y evalúa un algoritmo de conducción autónoma ejecutable tanto en el vehículo como en un servidor, y una serie de herramientas de comunicación para poder ejecutar el algoritmo en tiempo real fuera del vehículo independientemente de la red utilizada. Se evalúa su funcionamiento tanto en redes Wifi como en nuestra propia red 4G.

Abstract

In this Final Master Project we have developed a test bench for connected vehicle services through 4G networks and Edge Computing, allowing to test the possibilities of these networks with different types of services. Our own 4G network is implemented through SDR (Software Defined Radio) with different methods and an analysis, modification and programming of miniature vehicles to be used in the bench is performed. On the other hand, this work implements and evaluates, thanks to the developed test bench, a use case based on the autonomous driving of a vehicle performing off-loading in the network, i.e. downloading the processing tasks to an external server in real time. For this purpose, an autonomous driving algorithm executable both in the vehicle and in a server is designed and evaluated, as well as a set of communication tools to be able to execute the algorithm in real time outside the vehicle regardless of the network used. Its operation is evaluated both in Wifi networks and in our own 4G network.

Keywords

MEC, 4G, 5G, SDR, SDN, connected vehicle

Agradecimientos

En primer lugar, quiero agradecer a mi tutor Juan Carlos Aguado Manzano por haberme dado la oportunidad de trabajar en este proyecto y los medios para hacerlo. Siempre dispuesto a escuchar, ayudar y reflexionar en cualquier momento, con infinita paciencia y compromiso pleno. Su constancia y pasión por este y otros proyectos me ha inspirado y me ha permitido llegar más lejos de lo que yo hubiera pensado. Sin ninguna duda, ha sido toda una suerte poder trabajar con él y, lo más importante, tengo una amistad que no perderé nunca.

Me gustaría agradecer a Paula y a mis padres Javier y Lola por, como siempre han hecho, darme mucho cariño y apoyarme al máximo con todos y cada uno de los proyectos/líos en los que me meto. Aplaudirme en los momentos buenos, acogerme incondicionalmente en los malos y, sobretodo, soportarme con estrés.

También quiero agradecer a mis compañeros del laboratorio 28 por amenizar todas las tardes en las que coincidimos y compartir risas y cafés. En buena compañía y con buen humor se trabaja mejor.

Este trabajo ha sido financiado por la Consejería de Educación de la Junta de Castilla y León y el Fondo Europeo de Desarrollo Regional (proyecto VA23IP20), y el Ministerio de Ciencia e Innovación y la Agencia Estatal de Investigación (proyecto PID2020-112675RB-C42 financiado por MCIN/AEI/10.13039/501100011033)

Este trabajo ha sido realizado gracias a Vodafone, que autorizó de manera gratuita a la UVA a utilizar algunas de las frecuencias que tiene licenciadas para servicios de telefonía móvil para uso propio en actividades de investigación y docencia.

Índice

1.	Introducción.....	1
1.1.	Motivación del proyecto.....	1
1.2.	Objetivos.....	3
1.3.	Etapas y métodos.....	3
1.4.	Recursos.....	4
2.	Estado del arte.....	5
2.1.	Redes 4G-LTE, 5G-NSA y MEC.....	5
2.1.1.	Introducción a las redes 4G-LTE.....	5
2.1.2.	Arquitectura de las redes 4G-LTE.....	7
2.1.3.	Introducción y arquitectura de las redes 5G-NSA.....	9
2.1.4.	MEC en redes 4G y 5G.....	10
2.2.	Opciones de implementación y estado del arte.....	12
2.2.1.	Opciones para la implementación de una red móvil 4G-LTE/5G-NSA.....	13
2.2.2.	Opciones para la implementación de un MEC sobre redes 4G-LTE/5G-NSA.....	19
3.	Maqueta, vehículo y <i>software</i> de control.....	22
3.1.	Maqueta.....	22
3.2.	Vehículo.....	25
3.2.1.	Modificaciones físicas del vehículo.....	26
3.2.2.	<i>Software</i> original.....	28
3.2.3.	<i>Software</i> para la gestión de aplicaciones de conducción.....	30
3.3.	Servicio de conducción autónoma en red.....	40
3.4.	Algoritmo de control autónomo del vehículo.....	40
3.4.1.	Percepción.....	41
3.4.2.	Planificación.....	44
3.4.3.	Control.....	46
3.4.4.	Clase <code>artemis_autonomous_car</code>	48
4.	Red móvil 4G mediante SDR.....	51
4.1.	Configuración del dispositivo de usuario.....	51
4.2.	Configuración del SDR.....	52
4.2.1.	BladeRFxA9.....	52
4.2.2.	USRP NI-B2901.....	53
4.3.	RAN de la red con srsRAN.....	54
4.3.1.	Instalación.....	54
4.3.2.	Configuración.....	55
4.3.3.	Recomendaciones para mejorar el rendimiento del <i>software</i> srsenb.....	56
4.4.	Núcleo de la red.....	57

4.4.1.	srsEPC	57
4.4.2.	Open5GS.....	59
5.	Casos de uso y resultados.....	63
5.1.	Guiado del vehículo con procesamiento local.....	63
5.1.1.	Preparación de la red	63
5.1.2.	Preparación de las medidas.....	63
5.1.3.	Resultados	63
5.2.	Guiado del vehículo con procesamiento en servidor a través de wifi.....	64
5.2.1.	Preparación de la red	64
5.2.2.	Preparación de las medidas.....	65
5.2.3.	Resultados	67
5.3.	Guiado del vehículo con procesamiento en servidor a través de 4G sin <i>handover</i>	68
5.3.1.	Preparación de la red	68
5.3.2.	Preparación de las medidas.....	69
5.3.3.	Resultados	71
5.4.	Guiado del vehículo con procesamiento en servidor a través de 4G con <i>handover</i>	73
5.4.1.	Preparación de la red	73
5.4.2.	Resultados	74
6.	Conclusiones y líneas futuras.....	76
6.1.	Conclusiones.....	76
6.2.	Líneas futuras.....	77
7.	Bibliografía	78
Anexo I.....		83
Anexo II.....		85
Anexo III.....		87
Anexo IV		88
Anexo V		89

Índice de figuras

Figura 1. Modos de comunicación soportados por C-V2X [4].	1
Figura 2. Roadmap de casos de uso de conducción avanzada, tecnologías de conectividad y necesidades de espectro radioeléctrico [4].	2
Figura 3. Línea de tiempo de las generaciones de redes móviles y los estándares tecnológicos ligados a ellas. [14].	5
Figura 4. Evolución de la tecnología 4G LTE por versiones del 3GPP [18].	6
Figura 5. Arquitectura de la red LTE.	7
Figura 6. Opciones de implementación de redes 5G. [23]	9
Figura 7. Opción 3 de red 5G NSA [24].	10
Figura 8. Despliegue de un MEC utilizando el método <i>Bump in the wire</i> [27].	11
Figura 9. Despliegue del MEC con EPC distribuido [27].	11
Figura 10. Despliegue de un MEC S-GW y P-GW [27].	12
Figura 11. MEC en la arquitectura de las redes 5G. [26]	12
Figura 12. Resumen de la arquitectura de una plataforma SDR [28].	13
Figura 13. <i>Software</i> y dispositivos SDR analizados.	14
Figura 14. Servicios del núcleo de la red 5G implementados por OAI [31].	15
Figura 15. Conjunto de capacidades LTE de la plataforma Open Air Interface [32].	16
Figura 16. <i>Throughput</i> en el enlace de descarga para OAI-RAN y srsRAN [33].	17
Figura 17. Arquitectura de referencia de LightEdge [38].	20
Figura 18. Aplicación de conducción autónoma desarrollada para testear LightEdge.	20
Figura 19. Latencia con el tiempo para un creciente número de consultas (uno cada 2seg).	21
Figura 21. Conjunto de piezas diseñadas para la maqueta.	23
Figura 22. Materiales utilizados para la elaboración de la maqueta.	23
Figura 23. Medidas de acotadas de algunas de las piezas diseñadas para la versión 1 de la maqueta.	24
Figura 24. Medidas de acotadas de algunas de las piezas diseñadas para la versión 2 de la maqueta.	24
Figura 25. Ejemplos de montaje de la primera (izquierda) y segunda (derecha) versión de la maqueta.	25
Figura 26. Vista frontal y trasera del vehículo Amazon DeepRacer Evo [11].	25
Figura 27. Vista interior del vehículo con conexiones de la batería de impulso.	26
Figura 28. Nuevo cable conector conjunto batería-motor-ordenador, cable original y batería original.	27
Figura 29. Nuevos muelles montados en el vehículo.	27
Figura 30. Nueva batería y módem 4G instalado.	28
Figura 31. Nodos y tópicos utilizados en el <i>software</i> original del vehículo (exceptuando el nodo /rosout).	30
Figura 32. Comunicación del nodo admin_comunications_node con el resto de nodos mediante servicios. En gris nodos y servicios preinstalados en el vehículo. En naranja, nuevos nodos y servicios implementados.	34
Figura 33. Diagrama de flujo simplificado del funcionamiento del nodo admin_comunications_node .	35
Figura 34. Diagrama de flujo simplificado del funcionamiento del nodo autonomous_control_node .	37
Figura 35. Formato de tramas enviadas UDP/IP.	38
Figura 36. Diagrama de flujo simplificado del funcionamiento del nodo cloud_control_node .	39
Figura 37. Diagrama de comunicación entre el administrador, los vehículos y el servidor de control.	40
Figura 38. Ejemplo: Imagen original obtenida con la cámara del vehículo e imagen con perspectiva transformada.	41
Figura 39. Ejemplo: Imagen con perspectiva transformada e imagen filtrada.	42

Figura 40. Bases de color RGB y HSV.	42
Figura 41. Ejemplo: Recorte de imagen filtrada y obtención de contornos y puntos.	43
Figura 42. Ejemplo: Imagen perspectiva transformada con matriz de puntos representada.....	43
Figura 43. Ejemplo completo de capa de planificación con pista de 3 carriles.....	44
Figura 44. Ejemplo: Definición de ruta y elección de comando de selección de ruta por capa de planificación.	46
Figura 45. Variables utilizadas en el algoritmo de control de dirección de Stanley.	47
Figura 46. Problema por medición de segmento de ruta adelantado.....	48
Figura 47. Métodos analizados para dar estabilidad al algoritmo.	48
Figura 48. Esquema de funcionamiento de las funciones principales del algoritmo de conducción autónoma.	49
Figura 49. Tarjetas SIM sysmolSIM-SJA2 junto con dispositivo USB que permite reprogramarlas.....	51
Figura 50. BladeRFxA9 utilizada en el proyecto.	52
Figura 51. USRP NI-B2901 utilizada en el proyecto.	53
Figura 52. Interfaces de red configurables en srsENB.....	56
Figura 53. Interfaces configurables en srsEPC.....	58
Figura 54. Ejemplo de asignación de IPs a UEs.....	59
Figura 55. Servicios, direcciones IPs y puertos por defecto configurados, y nombre de los parámetros de configuración de las interfaces del núcleo de la red 4G implementada con Open5Gs.....	62
Figura 56. Topología de red para el guiado del vehículo con procesamiento en local.	63
Figura 57. Retardos con guiado del vehículo con procesamiento en local.	64
Figura 58. Topología de red para el guiado del vehículo con procesamiento en servidor de aplicaciones a través de wifi.....	65
Figura 59. Ejemplo de funcionamiento del <i>software</i> Qosium para la medición del tráfico de una aplicación concreta.	66
Figura 60. Retardos con guiado del vehículo con procesamiento en servidor de aplicaciones a través de wifi.....	67
Figura 61. Topología de red para el guiado del vehículo con procesamiento en servidor de aplicaciones a través de nuestra propia red 4G.....	69
Figura 62. Ejemplo de envío de mensaje desde el vehículo hasta el servidor de aplicaciones para el caso de uso de guiado del vehículo con procesamiento en servidor de aplicaciones a través de nuestra propia red 4G. Direcciones IPs, NAT y encapsulado GTP.....	70
Figura 63. Retardos con guiado del vehículo con procesamiento en servidor de aplicaciones a través nuestra red 4G.	72
Figura 64. Topología de red para la evaluación del <i>handover</i> en nuestra propia red 4G.....	73

Índice de tablas

Tabla 1. Parámetros configurados para comparación entre srsRAN y OAI-RAN [33].....	16
Tabla 2. Máxima tasa de descarga y máximo MCS logrado para OAI-RAN y srsRAN [33].	17
Tabla 3. Tiempo de ejecución de la CPU para OAI-RAN y srsRAN [33].	17
Tabla 4. Memoria utilizada por OAI-RAN y srsRAN [33].....	17
Tabla 5. Comparativa entre srsEPC, OAI-CN y Open5GS [34] [31] [35]	18
Tabla 6. Nodos ROS preinstalados en el vehículo junto a los tópicos en los que publican y a los que se suscriben, y los servicios que ofrecen.	29
Tabla 7. Mensajes MQTT enviados por el vehículo.....	32
Tabla 8. Estados del vehículo.	33
Tabla 9. Tipos de tramas definidos con su carácter de identificación.....	38
Tabla 10. Selección de trayectoria en función de matriz de puntos y el comando de selección de ruta seleccionado.....	44
Tabla 11. Atributos de la clase proceso_fotograma.....	49

1. Introducción

1.1. Motivación del proyecto

Actualmente, los requisitos de muy baja latencia, gran ancho de banda, fiabilidad y seguridad de los nuevos servicios de movilidad están obligando a cambiar la arquitectura de las redes móviles. Si bien la capa física de las redes móviles está en constante evolución, con las redes 4G completamente desplegadas y las redes 5G ya montadas en la mayoría de las grandes ciudades, las mejoras en las redes de acceso son ineficientes cuando los servicios se ofrecen desde nubes centralizadas. Es por esto que las necesidades de los nuevos servicios están conduciendo la arquitectura de las redes a plataformas de servicios descentralizadas basada en *Multi-access Edge Computing* (MEC) y/o *fog computing* [1]. En estas plataformas, parte de los recursos de computación en la nube (*cloud*) se deslocalizan de los grandes datacenters para situarse en una jerarquía que se extiende hasta el borde de la red de acceso radio (*edge*), permitiendo, principalmente, una menor latencia y un uso más eficiente de la red. Aunque los investigadores han realizado un enorme esfuerzo en la definición de nuevas arquitecturas y en la evaluación de los servicios a través de la simulación, existe una necesidad acuciante de probar los servicios sobre el terreno para ver cómo contribuyen los distintos elementos del sistema al rendimiento global [2] o, al menos, de probar dichos servicios en bancos de pruebas.

Un caso particular que requiere este tipo de pruebas son los servicios relacionados con la Movilidad Cooperativa, Conectada y Automatizada (CCAM). El estándar 5G incluye CCAM como un conjunto específico de casos de uso que deben probarse [3] mediante el uso de C-V2X (*Celular vehicle to everything communication*). C-V2X incluye tanto la conectividad vehicular tradicional V2V (vehículo a vehículo), V2I (vehículo a infraestructura) o V2P (vehículo a peatón); como la conectividad V2N (vehículo a la red) (Figura 1). Son los casos de uso que impliquen la utilización de V2N los que puedan requerir la implementación de plataformas MEC.

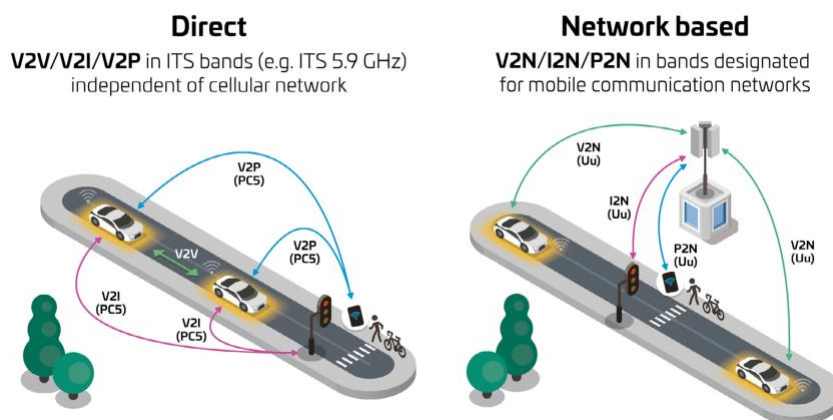


Figura 1. Modos de comunicación soportados por C-V2X [4].

Se ha previsto el despliegue de un conjunto completo de nuevos servicios durante los próximos 10 años [4] de los cuales, la mayoría, requieren un acceso a servicios en red de baja latencia y alta fiabilidad (Figura 2). Conducción teledirigida, aparcamiento automatizado de vehículos autónomos, compartición de datos de sensorica en tiempo real entre vehículos o gestión dinámica de intersecciones son algunos ejemplos de servicios que requieren conectividad con redes móviles que incorporen un MEC.

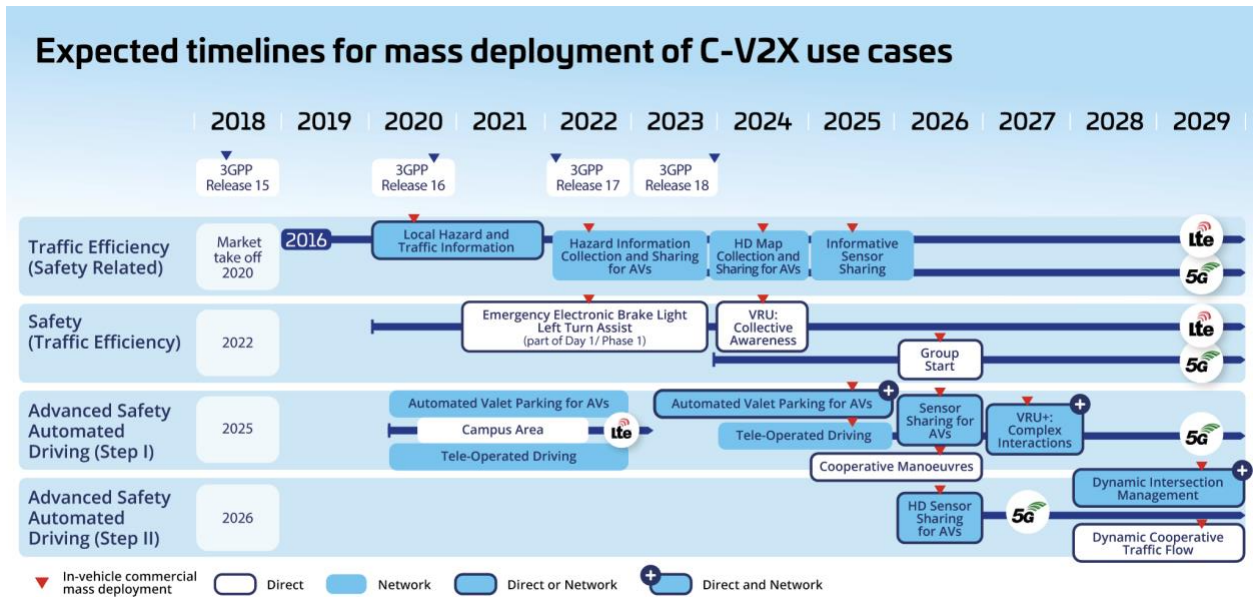


Figura 2. Roadmap de casos de uso de conducción avanzada, tecnologías de conectividad y necesidades de espectro radioeléctrico [4].

Muchos de esos servicios son cruciales para obtener las ventajas que promete la nueva movilidad. Por ejemplo, se ha demostrado ampliamente que una mayor penetración en el mercado de los vehículos automatizados no mejorará el flujo del tráfico debido a su conservadurismo a la hora de negociar situaciones de conducción complejas (véase, por ejemplo [5]). Una solución es crear servicios en la infraestructura para ayudar a los coches autónomos a tomar estas decisiones complejas, conscientes del contexto y coordinadas.

Si bien es posible encontrar resultados de investigación sobre los vehículos conectados, un montaje experimental completo sólo está al alcance de unas pocas empresas y grupos de investigación. A diferencia de los casos de uso relacionados con los drones, para los que se pueden encontrar muchos montajes experimentales, sólo hay unos pocos con vehículos, como por ejemplo [6]. De este modo, el desarrollo de bancos de pruebas experimentales que incluyan vehículos y tecnologías inalámbricas como 5G con MEC pueden ofrecer muchas ventajas.

La contribución de este Trabajo Fin de Máster es doble. Por un lado, se realiza un diseño y construcción de un **banco de pruebas** donde se pueden testear diferentes servicios ofrecidos a vehículos conectados y autónomos. Este incluye un circuito de carretera, vehículos a escala con ordenadores incorporados, y una infraestructura de red con varias tecnologías de estaciones inalámbricas y redes de acceso y núcleo. Por otro lado, se presenta una serie de **casos de uso** analizados con dicho banco de pruebas, relacionados con la descarga de tareas de procesamiento de un vehículo autónomo en la red. Para dichos casos de uso, se desarrolla un algoritmo de conducción autónoma simple y se programa al vehículo para que, en vez de ejecutar este algoritmo en local, envíe la información de sus diversos sensores a un servidor, de manera que este implemente el algoritmo de guiado y devuelva, en tiempo real, instrucciones de control al vehículo, todo ello a través de la red montada (4G o WiFi).

1.2. Objetivos

Este Trabajo Fin de Máster tiene dos objetivos principales. Por un lado, la **elaboración de un banco de pruebas** de servicios de vehículo conectado a través de redes 4G/5G NSA, permitiendo probar las posibilidades de estas redes con diferentes tipos de servicios. Por otro lado, la **implementación de un caso de uso** basado en la conducción de un vehículo autónomo realizando *off-loading* en la red, es decir, descargando las tareas de procesamiento en un servidor externo en tiempo real. Este caso de uso, incluido en el conjunto CCAM tal y como se explica en la sección anterior (Motivación del proyecto), muestra uno de los muchos casos de uso que se podrán probar en el banco de pruebas implementado.

Los objetivos específicos que cubren este proyecto son los siguientes:

- Diseño y elaboración de una maqueta¹ que permita evaluar multitud de casos de uso de vehículo conectado.
- Modificaciones físicas del vehículo teledirigido seleccionado para adaptarlo a la maqueta y darle conectividad 4G/5G.
- Análisis del funcionamiento del *software* de fábrica instalado en el vehículo.
- Modificación del *software* del vehículo para que pueda ser controlable desde el exterior mediante el intercambio de mensajes de control y/o datos de sensórica del dispositivo.
- Implementación de algoritmo de guiado autónomo para el vehículo preparado para que se mueva por la maqueta compatible con cruces, bifurcaciones y cambios de carril e instalación de dicho algoritmo dentro del vehículo y en un servidor.
- Análisis de la arquitectura de una red 4G y 5G NSA.
- Análisis de diversos métodos para la implementación de una red 4G o 5G NSA completa mediante SDR (*Software Defined Radio*).
- Implementación de una red 4G o 5G NSA con una o varias estaciones base con diferentes SDRs y diferentes *softwares* que implementen los servicios RAN (*Radio Access Network*) y diferentes *softwares* implementen los servicios del núcleo de la red.
- Realización de un caso de uso basado en la ejecución del algoritmo de guiado fuera del vehículo intercambiando la información necesaria a través de la red 4G en tiempo real entre este y el servidor.
- Medición de diferentes parámetros de la red en el caso de uso mencionado.

1.3. Etapas y métodos

Para llevar a cabo este proyecto, se han seguido una serie de fases:

1. Inicialmente, se realizó un **análisis** de los diferentes métodos disponibles en la literatura para implementar una **red 4G o 5G NSA** funcional mediante SDN.
2. A continuación, se diseñó e implementó la **maqueta** de acuerdo con una serie de requisitos.
3. Después, se realizó un **análisis, modificación y reprogramación** del **vehículo** seleccionado para el banco de pruebas.
4. Posteriormente se desarrolló un **algoritmo de conducción autónoma** capaz de ser ejecutado dentro o fuera del vehículo.
5. Más adelante, se **instalaron y probaron** diferentes combinaciones de *software* y *hardware* para implementar la **red móvil** mediante SDN, completando así el banco de pruebas.

¹ En este Trabajo Fin de Máster se llamará maqueta únicamente a la estructura física, es decir, a la pista de pruebas; mientras que se llamará banco de pruebas al sistema completo, que incluye la maqueta, el vehículo, la red móvil y los servidores.

6. Seguidamente se implementó un **sistema de mediciones** capaz de obtener en tiempo real múltiples **parámetros de la red** montada.
7. Finalmente, se realizaron **evaluaciones** del sistema mediante pruebas del caso de uso de conducción autónoma con procesado en la red modificando diferentes combinaciones *hardware* y *software* a la hora de montar la red móvil.

Es importante recordar los dos objetivos principales de este proyecto, mencionados en la sección 1.2. Las fases 1, 2, 3, 4, 5 y 6 son necesarias para la elaboración del banco de pruebas de vehículo conectado, permitiendo obtener la maqueta, el vehículo modificado y reprogramado, el algoritmo de conducción de autónoma que permite al vehículo moverse a través de la maqueta, la red móvil con la que se comunica el vehículo y un sistema de medición de parámetros de la red que permita evaluar el rendimiento de esta para diferentes aplicaciones. Por otro lado, es la fase 7 la implicada en la implementación del caso de uso en el cual, aprovechando el banco de pruebas desarrollado, se realiza una evaluación de la red para una aplicación particular.

1.4. Recursos

Para desarrollar este proyecto, se ha utilizado el siguiente material electrónico, elegido específicamente para este propósito:

- 6 Intel NUC BXNUC9I9QNX [7] con procesador Intel Core i9 de 9ª generación de 6 núcleos y 16 GB de memoria RAM para utilizarlos como servidores para servicios de red móvil y ejecución de aplicaciones.
- 2 USRP NI-2901 de National Instruments [8] con dos canales RX y TX MIMO, un rango de frecuencias de 70 MHz a 6 GHz y 56 MHz de ancho de banda para utilizarlos como *hardware* SDR.
- 6 *BladeRF 2.0 micro xA9* de Nuand [9] con dos canales RX y TX MIMO, un rango de frecuencias de 47MHz a 6 GHz y 30 MHz de ancho de banda para utilizarlos como *hardware* SDR.
- 10 SIMs programables sysmoSIM-SJA2 de la empresa Sysmocom [10].
- 4 vehículos Amazon Deep Racer con el pack de expansión Evo [11] con un Intel Atom de 1.4 GHz, 2 GB de RAM, giroscopio y acelerómetro, conexiones USB, conexión WiFi, cámara estéreo y un LiDAR Slamtec RPLIDAR A1 [12] de una capa con alcance de 12 metros.

Por otro lado, para la implementación de la maqueta se ha utilizado:

- Goma negra de 5mm para simular la carretera.
- Filtro negro de 90 cm de anchura para simular la carretera.
- Placas de PVC.
- Césped artificial.
- Pintura mate de diferentes colores.

2. Estado del arte

En esta revisión del estado del arte se realiza un análisis de la literatura para describir el funcionamiento de las redes 4G, 5G-NSA y el MEC; y los métodos de implementación en laboratorios utilizados hasta el momento. Para ello se divide en 2 secciones. En la primera sección (2.1) se da una introducción al 4G y LTE (2.1.1), se explica la arquitectura de las redes LTE (2.1.2), se introduce y explica la arquitectura de las redes 5G-NSA (2.1.3) y se muestra de forma teórica en qué consiste el MEC y las posibles arquitecturas de despliegue en dichas redes (2.1.4). Por otro lado, en la segunda sección (2.2), se muestran diferentes opciones de despliegue de redes 4G/5GNSA en entornos controlados mediante SDR (*Software Defined Radio*) y MEC. En esta última sección, también se muestran algunos casos de uso prácticos analizados en la literatura.

2.1. Redes 4G-LTE, 5G-NSA y MEC

2.1.1. Introducción a las redes 4G-LTE

La cuarta generación de tecnología móvil fue definida en Noviembre de 2008 a través de una serie de recomendaciones lanzadas por la Unión Internacional de Telecomunicaciones – sector Radiocomunicaciones (ITU-R) denominados *International Mobile Telecommunications Advanced (IMT-Advanced)* [13]. Estos documentos establecían los requisitos mínimos que las redes 4G debían ofrecer, como eficiencia espectral, ancho de banda, latencia, tiempo de interrupción en *handover*, etc. Todo ello en función de la velocidad de desplazamiento de los clientes de la red. Entre las mayores diferencias respecto a la generación anterior se encuentra una velocidad de pico de 1Gbps para comunicaciones con usuarios de baja movilidad, como peatones o usuarios detenidos, y de 100Mbps para comunicaciones de alta movilidad, como usuarios en diferentes medios de transporte.

Si bien se especificaban las condiciones para considerar una red móvil 4G, no se especificaba ni la tecnología, ni la arquitectura, ni los protocolos a utilizar. Por lo tanto, para cumplir los objetivos marcados por la red 4G, pueden existir una gran variedad de escenarios de desarrollo, diferentes capacidades de servicios y diferentes tecnologías (Figura 3).

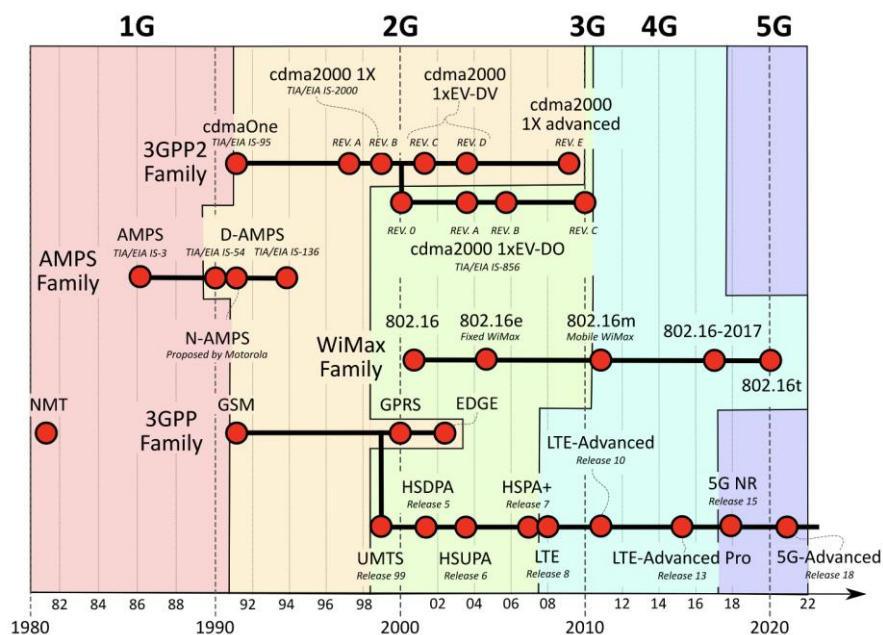


Figura 3. Línea de tiempo de las generaciones de redes móviles y los estándares tecnológicos ligados a ellas. [14]

Las primeras tecnologías que se lanzaron con el propósito de cumplir los requerimientos de 4G fueron LTE (Long Term Evolution) [15], estándar definido por 3GPP² en su versión 8, y *Mobile WiMAX* [16], definido por IEEE (incluido en la norma 802.16). Sin embargo, realmente estas tecnologías no reunían todos los requisitos del estándar y, si bien comercialmente se denominaban tecnologías 4G, se instó a que fueran llamadas tecnologías 3,9G ya que sí que habían marcado una diferencia relevante respecto de la generación anterior. Ambas tecnologías funcionan de manera similar, difiriendo principalmente en el uso de las frecuencias utilizadas. Este documento se centrará en la tecnología LTE al ser la utilizada por la mayoría de las operadoras para los casos de uso típicos de redes móviles, ya sea conectividad con teléfonos móviles, dispositivos IoT o vehículos conectados.

La tecnología LTE es un estándar para comunicaciones inalámbricas para dispositivos móviles basado en los estándares GSM/EDGE y UMTS/HSPA. Este mejora la capacidad y la velocidad de dichos estándares utilizando una interfaz de radio diferente y una serie de modificaciones en el núcleo de la red [17]. Son capaces de ofrecer en su última versión (*release 8*), 300 Mbps de velocidad de pico de bajada y 75 Mbps de velocidad de pico de subida, mucho menor que las velocidades exigidas por el *IMT-Advanced* (4G). Son compatibles con las tecnologías de las anteriores generaciones, lo que propició su rápida expansión en el mercado.

En 2011, tres años después del lanzamiento de LTE, se lanzó la versión 10 llamada LTE-Advanced (también conocido como LTE-A o LTE+ - Figura 4), evolución de la tecnología original que cumplía todos los requisitos impuestos por el estándar 4G, mejorando las tasas de transmisión de datos. Estas ventajas las consiguió principalmente gracias al uso de varias bandas de frecuencia de manera simultánea, conocido como agregación de portadora, donde un dispositivo móvil se puede conectar a la vez a dos estaciones base, multiplicando su ancho de banda disponible. También incorporó la modulación 256 QAM (en LTE la modulación máxima era de 64 QAM) y mejoró el MIMO de bajada de 4x4 a 8x8 y el de subida de 2x2 a 4x4.

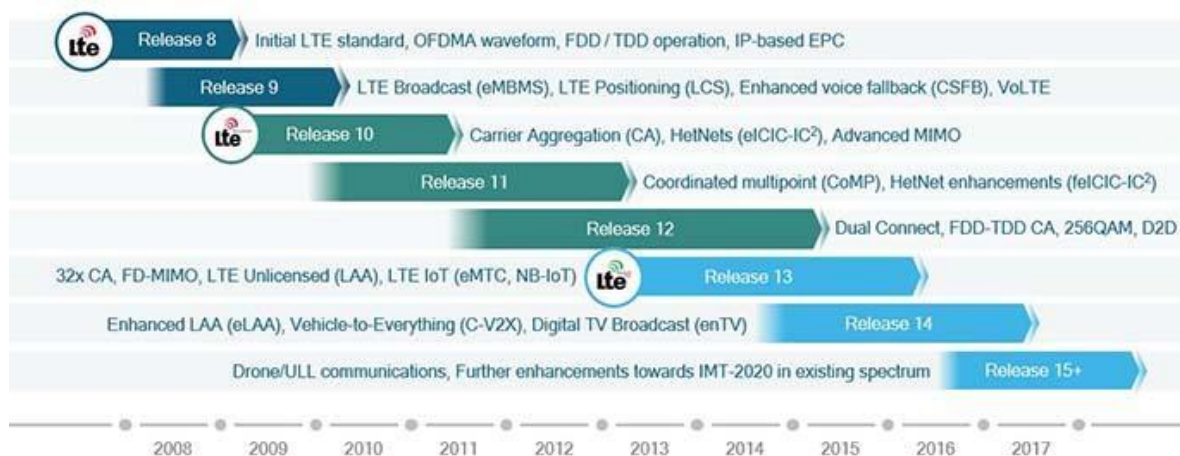


Figura 4. Evolución de la tecnología 4G LTE por versiones del 3GPP [18].

² Es importante destacar que la ITU es un organismo que define los estándares de las generaciones de redes móviles (2G, 3G, 4G o 5G), únicamente definen los requisitos de la red. Por otro lado, 3GPP es una organización de ingeniería que sí que desarrolla especificaciones técnicas, no estándares.

2.1.2. Arquitectura de las redes 4G-LTE

La arquitectura de las redes LTE se divide en 3 partes bien diferenciadas: equipo de usuario (UE, *User Equipment*), red de acceso radio (RAN, *Radio Access Network*) y red troncal (CN, *Core Network*). En la Figura 5 se puede ver un esquema simplificado de la arquitectura LTE, indicando el nombre de los diferentes servicios y sus interfaces de conexión.

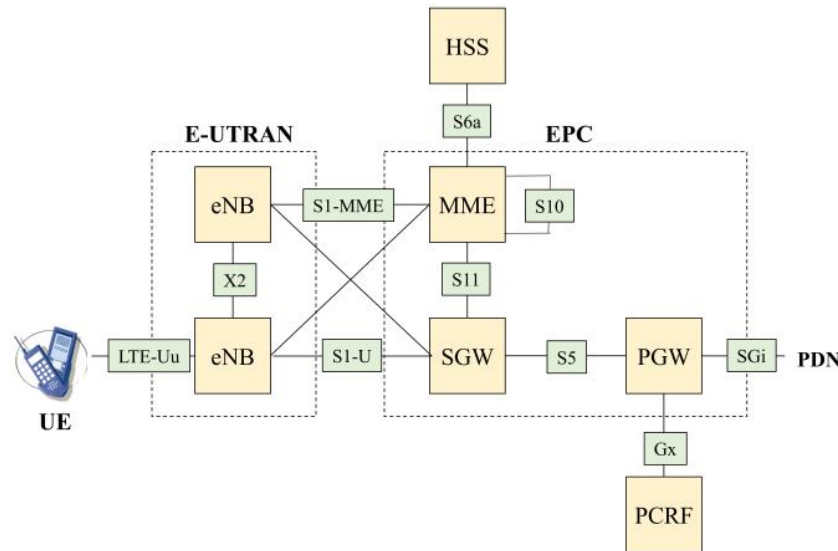


Figura 5. Arquitectura de la red LTE.

- **El equipo de usuario, UE**, de una red LTE incluye la misma arquitectura que uno utilizando UMTS (3G) o GSM (2G). Incluye los siguientes módulos:
 - Terminación móvil (MT), encargado de gestionar todas las funciones de comunicación.
 - Equipo terminal, encargado de recibir los flujos de datos.
 - Tarjeta Universal de Circuito Integrado universal (UICC), también conocida como la tarjeta SIM (Módulo de Identidad de Suscriptor) para los equipos LTE. Implementa la aplicación conocida como el Módulo Universal de Identidad de Suscriptor (USIM).
- **La red de acceso radio** LTE se denomina E-UTRAN (*evolved UMTS Terrestrial Radio Access Network*). Esta gestiona las comunicaciones radio entre el dispositivo móvil y el núcleo. Incluye a los **eNodeB** o eNB (*evolved base station*), las estaciones base que controlan los dispositivos conectados en una o varias celdas. Incluyen dos funciones principales:
 - Gestionar los recursos radio y las transmisiones para todos los dispositivos móviles conectados utilizando procesado analógico y digital de las señales. Debe gestionar tanto en enlace de bajada (DL, *DownLink*) como el enlace de subida (UP, *UpLink*). Esta interfaz de comunicación se denomina LTE-Uu, y su capa física puede utilizar multiplexado por división en frecuencia (FDD) en conjunción con OFDM o multiplexado por división en tiempo (TDD) [19].
 - Controlar las operaciones de bajo nivel de todos los dispositivos conectados a ellos enviando y recibiendo diferentes mensajes de control para obtener medidas de calidad del enlace, gestionar *handovers*, etc.
 - Enrutar paquetes del plano de datos al servicio *Gateway* del núcleo de la red.

Dos eNodesB se pueden comunicar entre sí a través de la interfaz opcional X2 que les permite compartir información y realizar el *handover* X2, conocido como *Intra-MME/SGW handover*, donde los servicios MME y SGW continúan siendo los mismos. Por otro lado, los eNodeB se comunican con el núcleo de la red a través de las interfaces S1, siendo S1-MME la

interfaz para conectarse al servicio MME (donde se intercambian los mensajes de control de la red), y SI-U la interfaz para conectarse al servicio SGW (donde se intercambian los paquetes del usuario para establecer los servicios de llamadas, mensajería, acceso a internet u otros).

- El **núcleo de la red LTE** se denomina EPC (*Evolved Packet Core*). Tiene dos planos principales: el plano de control y el de usuario. Cada plano engloba una serie de servicios concretos [20]:
 - El **plano de control** es el encargado de gestionar, mantener o modificar el estado de la red. Incluye los siguientes servicios:
 - **MME** (*Mobility Management Entity*) es el servicio principal del plano de control del núcleo. Gestiona la señalización relacionada con la movilidad, la paginación, las sesiones y seguridad del acceso a la red, y es responsable del seguimiento de la UE en estado ocioso.
 - **HSS** (*Home Subscriber Server*) es una base de datos que contiene la información de los usuarios y suscriptores de la red. Genera los vectores de autenticación de la SIM y mantiene el perfil de los abonados.
 - **PCRF** (*Policy and Charging Rules Function*) es el nodo responsable de las políticas la red. Gestiona el sistema de tasación de una manera centralizada en tiempo real y niveles de calidad de servicio. Permite crear conjuntos de reglas y tomar decisiones de políticas automáticamente para cada suscriptor de la red.
 - El **plano de usuario** es el encargado de transportar los paquetes de datos del usuario desde los eNodeB a la red pública (PDN, *Public Domain Network*), internet. Incluye los siguientes servicios:
 - **SGW** (*Serving GateWay*). Transporta el flujo de datos IP entre el UE el núcleo de la red. Sirve a los UEs enrutando los paquetes IP de entrada y salida al núcleo.
 - **PGW** (*PDN GateWay*) es el nodo que conecta el EPC a la red IP externa. Se encarga de enrutar los paquetes a y desde las redes IP externas. Aloja y asigna la IP de las UEs e implementa diferentes políticas observando el tráfico de usuario IP, como por ejemplo filtrado de paquetes.

Si bien ambas pasarelas son especificadas por separado, generalmente están combinadas en un único bloque.

A parte de las interfaces con la parte E-UTRAN ya explicadas anteriormente, en el núcleo de la red intervienen una serie de interfaces.

- **S6a** define la conexión entre el HSS y el MME. Permite que el MME pueda recuperar los perfiles de los suscriptores alojados en el HSS.
- **S10** define la conexión entre dos nodos MME. Permite el mecanismo de reubicación de la entidad MME de manera que cuando un usuario debe cambiar de servidor (por movilidad), a través de esta interfaz se realizará la transferencia del contexto de este usuario entre MMEs.
- **S11** define la conexión entre el MME y el SGW. Permite al MME controlar el funcionamiento del plano de usuario.
- **S5**: Permite la transferencia de paquetes de usuario entre SGW y PGW.
- **Gx**: define la conexión entre el PCRF y el PGW.
- **SGi**: define la conexión entre la red LTE con redes IP externas.

2.1.3. Introducción y arquitectura de las redes 5G-NSA

En octubre de 2015, la ITU comenzó a publicar los documentos que definían el 5G a través de los requisitos IMT-2020 (*International Mobile Telecommunications-2020*) [21], es decir, los requisitos que debía tener una red móvil para poder considerarse 5G. Esta generación de redes móviles no solo pretende ofrecer nuevos niveles de rendimiento y eficiencia, sino también ampliar las redes móviles creando un tejido de conectividad unificador para una amplia gama de casos de usos.

En 2016, la organización de ingeniería 3GPP comenzó a trabajar en la especificación técnica de una tecnología de acceso radio, llamada 5G NR (*New Radio*), y una nueva arquitectura de red, llamada 5G NG (*Next Generation*), para abordar los nuevos requisitos impuestos por la ITU. Al contrario que con las redes anteriores, en esta ya no existen varios organismos de normalización que compitan entre sí y trabajen en posibles soluciones para el 5G, como WiMax y LTE con el 4G. En esta generación existe un único estándar técnico y es el desarrollado por 3GPP [22].

3GPP define una serie de modos de implementación de la tecnología 5G en función de si están combinadas con las redes 4G LTE. Estos modos de implementación los define para permitir a las operadoras una instalación gradual de la nueva generación en sus redes 4G LTE ya desplegadas. En la Figura 6 se pueden ver las opciones de implementación de redes 5G. Las opciones 3, 7 y 4 son despliegues en los que la red 5G está combinada con elementos de la arquitectura *legacy* 4G LTE. La opción 3 concretamente, engloba a las redes conocidas como 5G NSA (*Non StandAlone*), al no disponer conjuntamente de estación base 5G y núcleo 5G.

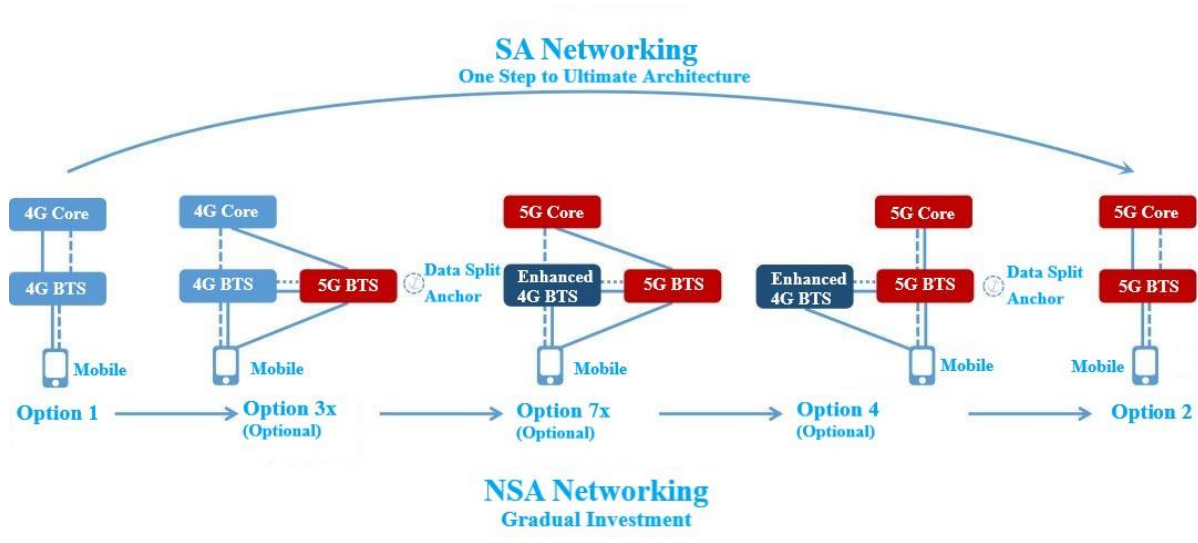


Figura 6. Opciones de implementación de redes 5G. [23]

La opción 3 es la más implementada por las operadoras en la actualidad. Tal y como vemos en la figura anterior, consiste en una red 4G LTE a la que se le ha añadido con una estación base 5G NR (gNodeB). En estos despliegues, el núcleo de la red consiste en un EPC mientras que la parte de acceso radio se comparte entre un eNodeB y un gNodeB. De esta forma, la parte de la red 4G LTE es la responsable del plano de control mientras que la parte 5G se centra exclusivamente en el plano de usuario. Con estas redes se consigue una rápida implementación al tener que montar únicamente la estación base 5G conectada con la estación base 4G ya desplegada mediante la interfaz estandarizada X2 (Figura 7). Permite, por tanto, no cambiar todo el núcleo de la red mientras se permite a los usuarios disfrutar de algunas de las ventajas de velocidad y latencia del acceso radio 5G (no del núcleo).

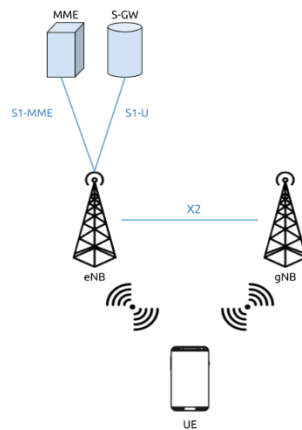


Figura 7. Opción 3 de red 5G NSA [24].

2.1.4. MEC en redes 4G y 5G

El *edge computing* multiacceso (MEC) es un tipo de arquitectura de red que permite proporcionar ciertas funciones del *cloud computing* en el borde de la red, es decir, más cerca del cliente. Esto permite mejorar la calidad de los servicios ofrecidos con latencias más bajas y mejor eficiencia en el funcionamiento de la red. En general, permite que los usuarios finales estén más cerca de las aplicaciones, y viceversa, permite que los recursos de computación donde se ejecutan las aplicaciones estén más cerca de los datos que generan dichas aplicaciones [25]. Se considera una tecnología clave para poder explotar una amplia gama de nuevos casos de uso que, sin esta arquitectura, serían inviables.

Como tecnología de acceso, MEC se ofrece para cualquier caso de uso que requiera de baja latencia, independientemente del tipo de red de acceso en la que se implemente. No obstante, el objetivo original del MEC es la red móvil y, si bien la arquitectura 5G SA viene preparada para implementar un MEC de forma nativa, se espera que el 4G siga teniendo éxito en los próximos años, por lo que una gran parte de la industria está trabajando para ejecutar MEC en redes 4G existentes.

La arquitectura MEC de referencia definida por la ETSI en su estándar GS MEC 003 [26], completamente ligada a las redes móviles, define 3 métodos para implementar un MEC en las redes 4G LTE [27]:

- **Bump in the wire:** Este método engloba todos los escenarios en los que la instalación de la plataforma MEC se encuentra entre la propia estación base y el núcleo de la red. Existen dos escenarios principales englobados en este método:
 - eNodeB y plataforma MEC en una única implementación: En este caso, la plataforma MEC es capaz de enrutar paquetes IP simples hacia y desde las aplicaciones MEC y, por otro lado, enrutar paquetes encapsulados en GTP hacia y desde la pasarela SGW, es decir, la interfaz S1-U heredada. Este despliegue es muy conveniente en múltiples casos, como por ejemplo escenarios empresariales para permitir que el tráfico de la intranet de la empresa se desprenda hacia los servicios locales.
 - Cualquier otra ubicación entre el eNodeB y el núcleo: En el resto de casos, independientemente de la cercanía o lejanía del MEC a la estación base, esta se encuentra en la interfaz S1 de la arquitectura 4G. En este escenario, el MEC procesa todo el tráfico de usuario desencapsulando los paquetes GTP-U, y reenviando los paquetes que no están relacionados con servicios ofrecidos por el MEC al núcleo y procesando en local los paquetes que sí están relacionados.

Estas operaciones no son triviales ya que cierta porción de los datos que se dirigen al núcleo de la red puede ser generada o manipulada internamente por las plataformas MEC. Por otro lado, el tráfico de los usuarios a los servicios ofrecidos por el MEC, al no pasar por el núcleo de la red, dificulta las operaciones de cobro o interceptación legal. Para solventar estos problemas se plantea la implementación de un nodo MEC GW (tal y como se ve en la Figura 8) conectado directamente a los servicios del núcleo de la red que llevan a cabo dichos procesos de métrica.

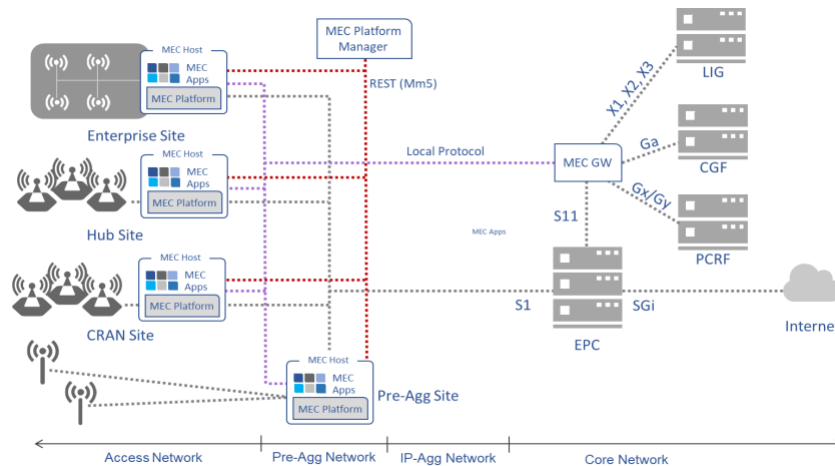


Figura 8. Despliegue de un MEC utilizando el método *Bump in the wire* [27].

- EPC distribuido:** En este despliegue el host MEC incluye todo o parte de los componentes del EPC de las redes 4G LTE, encontrándose el plano de datos del MEC en la interfaz SGi. En estos escenarios, el servicio PGW del EPC es el encargado de asignar la IP y la información DNS concreta para resolver las aplicaciones MEC, por lo tanto, es el que decide qué paquetes se procesarán en el MEC, y cuáles deben de dirigirse a la PDN. Estos escenarios son muy útiles en situaciones donde el EPC está cerca de las estaciones base y, por lo tanto, el hecho de que los paquetes atraviesen el EPC no añade grandes pérdidas de calidad en las conexiones. Además, requieren menos cambios en la red del operador ya que se aprovechan las entidades e interfaces 3GPP estándar para operaciones como la gestión de la sesión o el cobro.

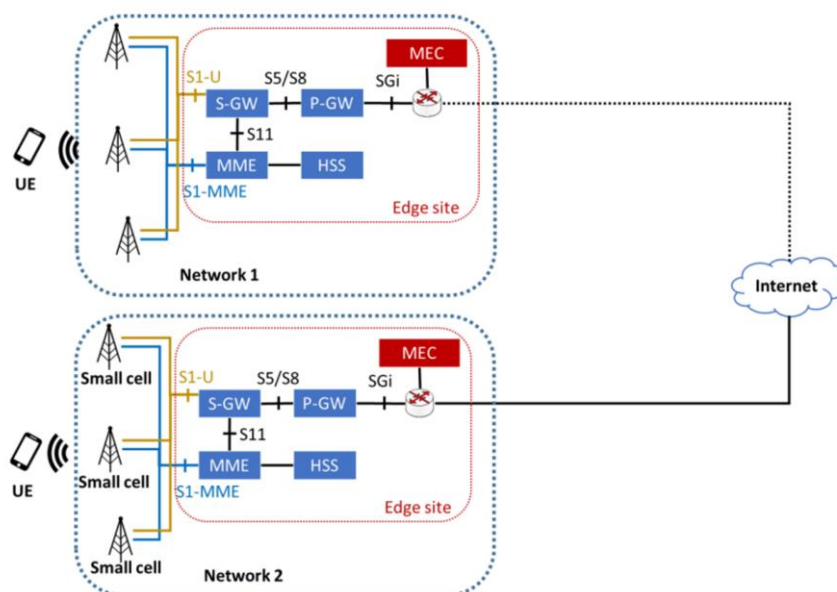


Figura 9. Despliegue del MEC con EPC distribuido [27].

- **S/PGW distribuido:** Esta opción es similar a la anterior, salvo que únicamente se distribuyen las entidades S-GW y P-GW (plano de usuario) habiendo un MEC por cada par pero manteniendo el MME y el HSS en el núcleo del operador, pudiendo haber uno para varios MEC.

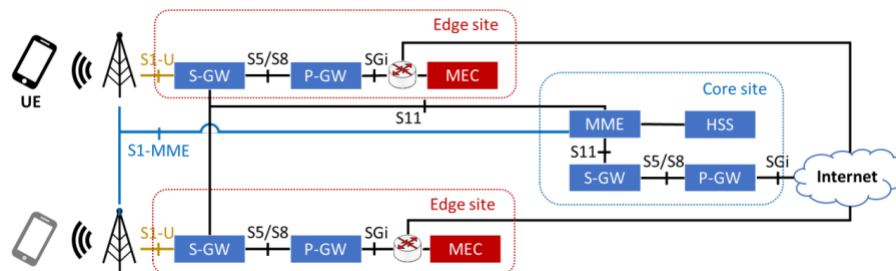


Figura 10. Despliegue de un MEC S-GW y P-GW [27].

En el caso de las redes 5G, a diferencia de las redes 4G LTE, estas fueron concebidas para permitir un despliegue más flexible del plano de datos, con el objetivo de soportar de forma nativa el *edge computing*. La arquitectura 5G especificada por 3GPP TS23.501 contiene nuevas funcionales del plano de control como la función de aplicación (AF) y una serie de nuevas funciones del plano de datos como la función del plano de usuario (UPF). La AF (función de aplicación) realiza operaciones de enrutamiento de tráfico de aplicaciones e interacción con la PCF (función de control de políticas) para el control de políticas. Por otro lado, la UPF permite la detección de aplicaciones mediante plantillas de filtrado de tráfico, permitiendo que el procesamiento de los paquetes se pueda realizar más cerca del borde de la red. Este hecho hace que la integración y el desarrollo del MEC en la arquitectura 5G sea una opción conveniente. Tal y como vemos en la Figura 11, dichos servicios son los que facilitarían la implementación del MEC, donde la UPF llevaría asignada la aplicación a ejecutar en el MEC y la AF las tareas de enrutado y políticas.

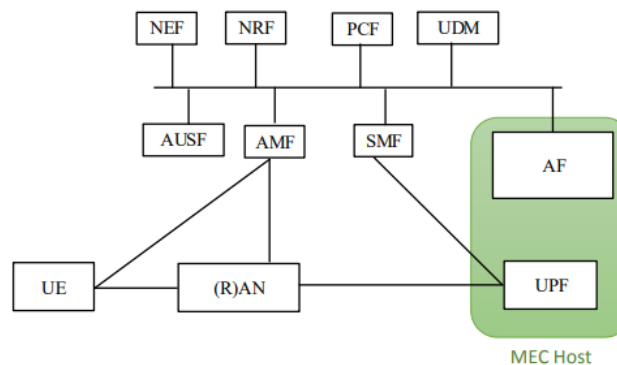


Figura 11. MEC en la arquitectura de las redes 5G. [26]

A pesar de los beneficios que aporta el 5G, pasará algún tiempo hasta que el actual sistema 4G transite a un sistema 5G completo. Las redes comerciales 5G se están desplegando inicialmente en modo NSA lo que significa que la red de acceso radio 5G sigue interactuando con el núcleo EPC de 4G. Aunque este enfoque permite disponer inmediatamente de algunas de las ventajas de 5G, impide el despliegue de nuevas aplicaciones y servicios en los extremos de la red de forma nativa, y por lo tanto, obliga a los operadores a implementar alguno de los métodos explicados anteriormente.

2.2. Opciones de implementación y estado del arte

En esta sección se muestra una serie de opciones de implementación de una red 4G/5G NSA basándose en SDR (*Software Defined Radio*), plataformas que se explican a continuación, y servicios

virtualizados. A su vez, se analizan varios *softwares* de código abierto disponibles para la implementación de un MEC en dichas redes.

Una plataforma SDR es un sistema de radiocomunicaciones en el que los componentes que típicamente se han implementado en forma *hardware* son sustituidos por *software* que se ejecuta en ordenadores o FPGAs. Para transmitir y recibir las señales, se utiliza un dispositivo denominado también SDR. Este dispositivo se encarga de hacer de frontera entre el ordenador o FPGA y el tratamiento analógico de la señal. Este *hardware* para recibir señales de radio y enviarlas digitalizadas al ordenador incluye un mezclador con oscilador local que desplaza en frecuencia la señal recibida a banda base, seguido de un conversor analógico a digital. Por otro lado, para transmitir señales incluye un conversor digital-analógico seguido de un mezclador que eleva la frecuencia de la señal a la necesaria. En estos dispositivos generalmente se configuran los parámetros de frecuencia, ancho de banda y tasa de muestreo de transmisión y de recepción. Este sistema aporta mucha flexibilidad al poder implementar en menor tiempo, con menor coste y con ordenadores convencionales cualquier sistema de radiocomunicaciones.

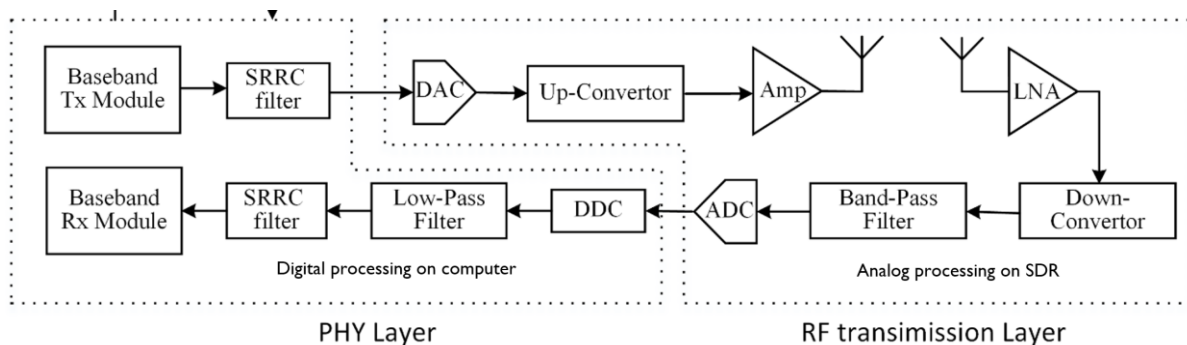


Figura 12. Resumen de la arquitectura de una plataforma SDR [28].

2.2.1. Opciones para la implementación de una red móvil 4G-LTE/5G-NSA

Para lograr implementar una red móvil existen tres necesidades principales: núcleo, RAN y dispositivo SDR. Para cada uno de ellos, se han analizado las opciones más utilizadas en la literatura (Figura 9):

- **Implementación de los servicios del núcleo de la red móvil:** Los servicios de la red móvil se pueden ejecutar en uno o varios ordenadores mediante *software* que virtualice dichas funciones. Existen múltiples *softwares* de código abierto que, instalándose y ejecutándose en un ordenador, permiten virtualizar ciertos servicios de las redes móviles 4G LTE y 5G. En este estado del arte se analizan el *software* srsEPC (perteneciente a la plataforma srsRAN), OAI-CN (perteneciente a la plataforma Open Air Interface) y Open5GS.
- **Implementación de los servicios de la RAN de la red móvil (eNodeB/gNodeB):** Al igual que en el caso anterior, existe *software* de código abierto que virtualizan dichas funciones, capaces de realizar el procesamiento de señal digital mediante *software*. En este estado del arte se analiza el *software* srsENB (perteneciente a la plataforma srsRAN) y OAI-RAN (perteneciente a la plataforma Open Air Interface).
- **Transmisión y recepción de señal:** Utilizando SDR, el cual, conectado y bien sincronizado con el servicio RAN, pueda transmitir y recibir señales electromagnéticas y enviar y recibir señales digitales del ordenador. Existe una gran variedad de *software* y *hardware* disponible para lograr este objetivo.

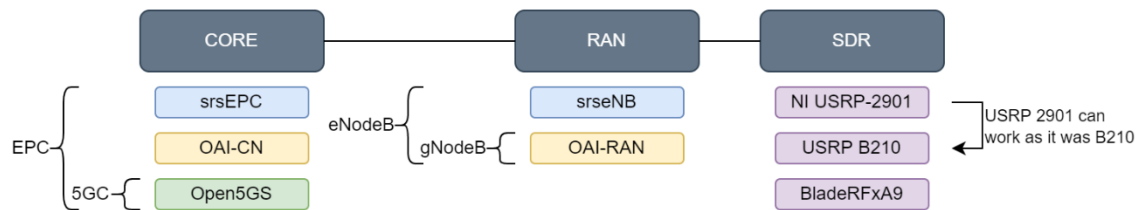


Figura 13. Software y dispositivos SDR analizados.

2.2.1.1. SrsRAN

La plataforma srsRAN es un conjunto de *software* de código abierto que implementa, de forma virtualizada, las funciones del UE, eNodeB/gNodeB y núcleo de la red 4G. Incluye tres *softwares* bien diferenciados para ejecutar cada una de estas funciones [24]:

- **srsUE**: Aplicación que incorpora la pila de protocolos completa de 4G del lado de usuario. Consiste en un módem *software* 4G LTE / 5G NSA que se ejecuta como una aplicación estándar del sistema operativo basado en Linux. Permite conectar cualquier dispositivo a cualquier red LTE (con o sin 5G NSA) proporcionando una interfaz de red estándar. Para la transmisión y recepción de señales, es necesario de un equipo SDR, el cual es controlado por dicho *software*.
- **srsENB**: Aplicación que incorpora la pila de protocolos completa de 4G del lado eNodeB y 5GNSA gNodeB. Implementa, mediante *software*, todo el procesamiento y los servicios que implementan los eNodeB y gNodeB 5G NSA. El servicio eNodeB se encuentra alineado con la versión 10 de LTE y permite implementar celdas de hasta 20MHz con 4 modos de transmisión: SISO, MIMO TM2 transmitiendo diversidad, MIMO TM3 CCD y configuración MIMO TM4 (multiplexado espacial de lazo cerrado). Con configuración SISO permite tasas máximas de 75Mbps de bajada y 50Mbps de subida, y con configuración MIMO TM3 o MIMO TM4 eleva la tasa de bajada hasta 150Mbps. Implementa las interfaces S1 estándar con sus protocolos correspondientes. Permite *handover* intra-ENB (entre diferentes subsectores de la celda) e inter-ENB (entre diferentes eNBs a través de la interfaz S1, gestionado por el núcleo). Para la transmisión y recepción de señales, es necesario de un equipo SDR, el cual es controlado por dicho *software*. Permite implementar también un gNodeB pero no por separado, sino funcionando en tiempo real a la vez que un eNodeB gestionado por la misma aplicación. Además, este caso de uso implica la utilización del mismo SDR para ambas estaciones base 4G y 5G. Esta posibilidad está pensada exclusivamente para implementar una red 5G NSA.
- **srsEPC**: Implementación *software* ligera de una red central LTE completa (EPC) virtualizada. Esta aplicación se ejecuta como un único binario, pero proporciona los componentes clave del EPC: servicio de abonado doméstico (HSS), entidad de gestión de la movilidad (MME), pasarela de servicios (S-GW) y pasarela de red de paquetes de datos (P-GW).

2.2.1.2. Open5GS

Open5GS [29] es una implementación de código abierto en lenguaje C que implementa la mayoría de los servicios, de forma virtualizada, del **núcleo** de la red 4G (EPC) compatible con 5G NSA, es decir, el núcleo de la red LTE/NR (*Release-16*), y el núcleo de la red 5G (5G NG, parte de una red 5G SA). Implementa un Core 4G / 5G NSA o un Core 5G SA. En este Trabajo Final de Máster nos centraremos en el núcleo 4G.

El núcleo 4G / 5G NSA de esta plataforma incluye los siguientes servicios (notar que tanto el servicio SGW como el servicio PGW de las redes 4G son divididos en dos servicios cada uno, uno para el plano de control y otro para el plano de datos):

- MME - *Mobility Management Entity*
- HSS - *Home Subscriber Server*
- PCRF - *Policy and Charging Rules Function*
- SGWC - *Serving Gateway Control Plane*
- SGWU - *Serving Gateway User Plane*
- PGWC/SMF - *Packet Gateway Control Plane*
- PGWU/UPF - *Packet Gateway User Plane*

Cada servicio es configurable por separado, pudiendo configurar su comportamiento y las IPs de sus interfaces de forma independiente. Esto permite poder implementar el núcleo de la red de forma distribuida, es decir, en varios ordenadores donde cada uno se encargue de ciertas funciones concretas de la red. Esta es una ventaja importante comparándolo con el núcleo ofrecido por srsEPC, ya que este consiste en un único bloque binario que implementa la mayoría de los servicios, los cuales no son configurables independientemente y obliga a ejecutar todos ellos en la misma máquina.

2.2.1.3. Open Air Interface

La plataforma Open Air Interface [30] es un conjunto de *software* de código abierto programado en C para varias variantes de Linux que implementa, de forma virtualizada, todos los servicios de las redes 4G LTE. Permite implementar el equipo de usuario (OAI-UE), la estación base (OAI-eNB o OAI-RAN) y la red central (OAI-EPC o OAI-CN), todo ello a través de *software* y, en los casos en los que se necesite conectividad radio, conectado a un dispositivo SDR (Figura 15):

- OAI-UE: Implementa los servicios del equipo de usuario 4G y 5G, permitiendo ofrecer conectividad 4G a cualquier ordenador mediante un dispositivo SDR. Permite configuraciones FDD y TDD con 5, 10 y 20 MHz de ancho de banda, con modos de transmisión 1 (SISO) y 2, 4, 5 y 6 (MIMO 2x2).
- OAI-CN: Implementa los servicios del núcleo de la red LTE, incluyendo el MME, HSS, SGW y PGW. Así mismo, permite ejecutar algunos de los servicios del núcleo de la red 5G (Figura 14). De la misma forma que Open5GS, permite configurar cada servicio y ejecutarlo de forma independiente por lo que se puede crear un *núcleo* distribuido en una red de varios ordenadores.

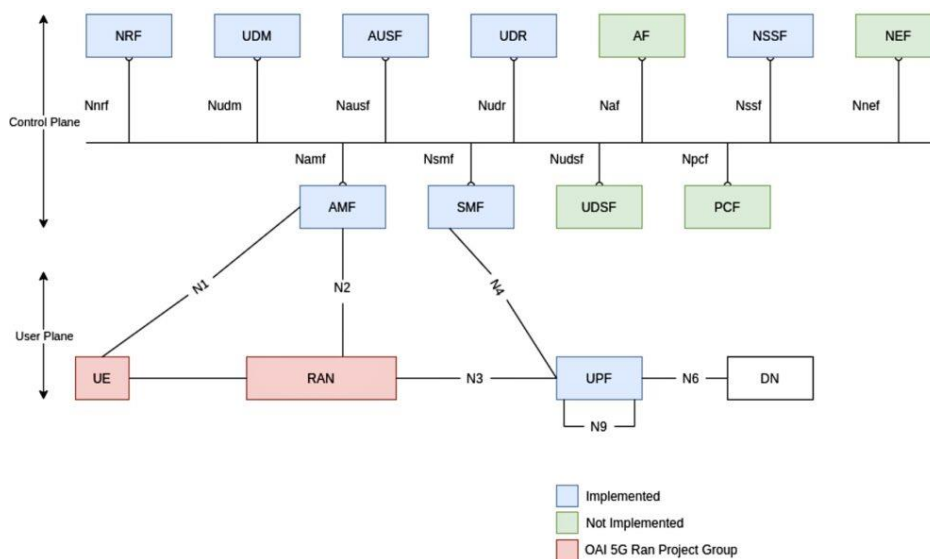


Figura 14. Servicios del núcleo de la red 5G implementados por OAI [31].

- OAI-RAN: Implementa los servicios del eNodeB y gNodeB, siendo compatible con 5G NSA, interconectando un eNodeB con un gNodeB mediante la interfaz X2.

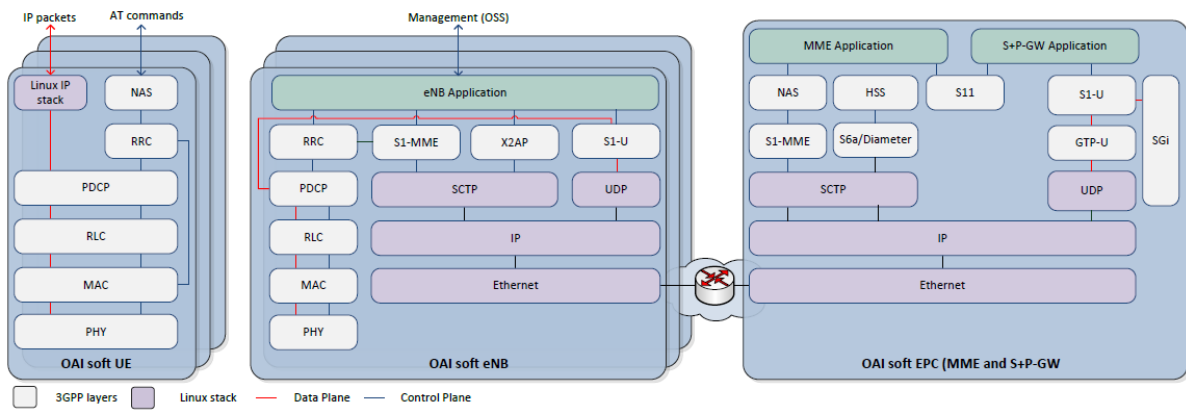


Figura 15. Conjunto de capacidades LTE de la plataforma Open Air Interface [32].

Además, proporciona un entorno de desarrollo con una serie de herramientas incorporadas como modos de emulación realistas, herramientas de monitorización y depuración, analizador de protocolos, etc.

2.2.1.4. Comparativa de opciones para la implementación de RAN

Tal y como se muestra anteriormente, las dos opciones analizadas para la implementación de un RAN a través de SDR en este Trabajo Fin de Máster son srsRAN y OAI-RAN. A continuación se comparan ambos *softwares*.

Para las comparaciones de rendimiento se han utilizado los estudios realizados por los autores de [33]. Para ello, implementaron una red 4G-LTE donde la parte RAN estaba montada con OAI y srsRAN. Utilizaron la banda 7, con sistema de multiplexado por frecuencia (FDD), anchos de banda de 5 y 10 MHz y una distancia entre el UE y el eNB de medio metro (Tabla 1). La ganancia de recepción la configuraron a 90 dB en el caso de OAI y en control automático de ganancia (AGC) en el caso de srsRAN. Esta configuración se debe que, si bien srsRAN implementa AGC, OAI no.

Tabla 1. Parámetros configurados para comparación entre srsRAN y OAI-RAN [33].

Parameter	OAI	srsLTE
<i>Duplex mode</i>	FDD	FDD
<i>System bandwidth</i>	5 MHz / 10 MHz	5 MHz / 10 MHz
<i>Transmission mode</i>	TM1	TM1
<i>DL carrier frequency</i>	2.68 GHz (band7)	2.68 GHz (band7)
<i>UE RX gain</i>	90 dB	AGC
<i>Distance between UE and eNB</i>	0.5 m	0.5 m

El *throughput* de bajada obtenido, en función del MCS (*Modulation and Coding Scheme*), utilizando una u otra herramienta se puede ver en la Figura 16.

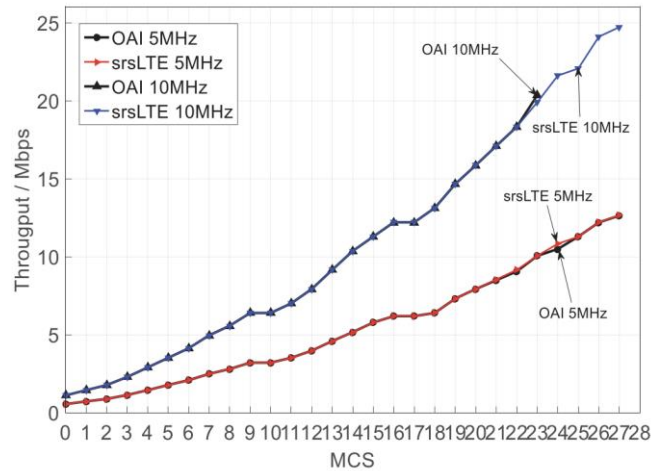


Figura 16. *Throughput* en el enlace de descarga para OAI-RAN y srsRAN [33].

En la Tabla 2 se observan las tasas de transmisión máximas obtenidas.

Tabla 2. Máxima tasa de descarga y máximo MCS logrado para OAI-RAN y srsRAN [33].

System Bandwidth	OAI		srsLTE	
	MCS	Peak rate (Mbps)	MCS	Peak rate (Mbps)
5 MHz	27	12.6368	27	12.6603
10 MHz	23	20.3427	27	24.7209

Ambos cumplen correctamente con las tasas de transmisión teóricas para cada combinación de ancho de banda y MCS. Sin embargo, en el caso de 10MHz, se comprueba que con srsRAN se puede alcanzar el valor 27, y por lo tanto mayores tasas de transmisión, mientras que con OAI el valor máximo alcanzable es 23.

En la Tabla 3 se muestran los tiempos de procesamiento de ambos *softwares* en un mismo ordenador transmitiendo en las mismas condiciones. Comprobamos como srsRAN está mejor optimizado, teniendo tiempos de procesamiento más pequeños.

Tabla 3. Tiempo de ejecución de la CPU para OAI-RAN y srsRAN [33].

CPU execution time		OAI	srsLTE
TX	Encoding	13 us	9 us
	Scrambling	11 us	8 us
	Modulate	56 us	22 us
	OFDM modulate	23 us	14 us
RX	OFDM demodulate	23 us	15 us
	Demodulate	11 us	10 us
	Descrambling	9 us	3 us
	Decoding	12 us	13 us

En la Tabla 4 podemos ver la memoria utilizada por OAI y srsRAN. Vemos cómo srsRAN utiliza mucha menos memoria tanto para la implementación de una eNB como un UE.

Tabla 4. Memoria utilizada por OAI-RAN y srsRAN [33].

Memory Utilization	OAI	srsLTE
eNB	1038728 KiB / 13.7%	55293 KiB / 0.7%
UE	786058 KiB / 9.7%	129659 KiB / 1.6%

Comprobamos, por tanto, que tanto a nivel de *throughput*, como a nivel de tiempo de ejecución de CPU y memoria RAM utilizada srsRAN es más recomendable que OAI. Sin embargo, Open Air Interface permite la implementación de un gNodeB 5G completo independiente, mientras que srsRAN, si bien también permite la implementación de un gNodeB, este únicamente servirá para establecer una red 5G NSA. Además, este deberá de implementarse en conjunción con un eNodeB donde ambas estaciones base deberán ejecutarse desde el mismo dispositivo SDR y el mismo proceso *software*. Esto quiere decir que, en el caso de querer implementar una red 5G NSA, srsRAN nos obliga a utilizar un mismo ordenador y un mismo SDR tanto para el eNodeB como el gNodeB. De hecho, para poder implementar dos estaciones base en un mismo SDR es necesario que este disponga de 2 canales Rx y 2 canales Tx completamente independizados cuando, generalmente, los SDR más comúnmente utilizados por su bajo coste con varios canales sólo son capaces de implementar sistemas MIMO en los que ambos canales trabajan a la misma frecuencia (no son independientes).

2.2.1.5. Comparativa de opciones para implementación de EPC

Tal y como hemos visto anteriormente, las tres opciones analizadas para la implementación de los servicios del núcleo de las redes 4G-LTE son: srsEPC, Open5GS y Open Air Interface – Core Network. A continuación, se muestra una tabla comparativa (Tabla 5) que muestra las mayores diferencias entre los 3 *softwares* analizados, realizada a partir de [34] [31] [35].

Tabla 5. Comparativa entre srsEPC, OAI-CN y Open5GS [34] [31] [35]

Tema	Soporte	srsEPC	OAI CN	Open5GS
Núcleo 4G LTE EPC	Soporte servicios LTE básicos: HSS, MME, S-GW y P-GW	Sí	Sí	Sí
	Ejecución de servicios independizados	No	Sí	Sí
	Soporte servicio PCRF	No	No	Sí
	Compatibilidad con Handover SI	No	No	Sí
Núcleo 5G NG	Soporte servicios AMF, SMF, UPF, AUSF, NRF, UDM, UDR, NSSF	No	Sí	Sí
	Soporte servicios anteriores + BSF y PCF	No	No	Sí
	Soporte servicios anteriores + NEF, UDSF y AF	No	No	No
Capacidades	Configuración de suscriptores mediante interfaz gráfica intuitiva	No	No	Sí
	Configuración de IPs estáticas por clientes	No	Sí	Sí
	Configuración de varios APNs con diferentes colecciones de IPs y parámetros de calidad de servicio	No	Sí	Sí

Como podemos ver en la tabla, srsEPC implementa un núcleo muy ligero, con los 4 servicios imprescindibles, todo en un mismo *software* (servicios no independizados) y sin soporte para la realización de *handover SI* (*handover* gestionado por el MME). OAI-CN, sí que permite la ejecución de servicios independizados, implementa una serie de servicios del núcleo 5G y permite la configuración de varios APNs con diferentes parámetros de calidad de servicio e incluso IPs estáticas por clientes. Finalmente, Open5GS es el núcleo más completo aportando, además de lo comentado anteriormente, implementación del servicio PCRF de LTE y más servicios del núcleo 5G, compatibilidad con el *handover SI* y una interfaz gráfica web intuitiva para la configuración de suscriptores.

2.2.2. Opciones para la implementación de un MEC sobre redes 4G-LTE/5G-NSA

A continuación, se va a mostrar una alternativa para la implementación de un MEC sobre las redes 4G/5G NSA analizadas anteriormente. Es importante recalcar que existe muy poca variedad de implementaciones *software* del MEC de código abierto en la actualidad. Varios trabajos han demostrado que es posible introducir MEC y soluciones *edge* en redes 4G como [36] o [37]. Sin embargo, ninguno de ellos toma el reto de implementar estas soluciones de forma práctica.

2.2.2.1. LightEdge

LightEdge consiste en una solución MEC para redes 4G desarrollada por Estefanía Coronado, Zarrar Yousaf y Roberto Riggio [38] que cumple con el estándar MEC de la ETSIT. Como se ha comentado anteriormente, las redes 4G no incluyen de forma nativa ningún método para encaminar el tráfico a un MEC y, si bien existen una serie de recomendaciones y métodos desarrollados por la ETSIT, no existe ninguna especificación tecnológica cerrada. El objetivo principal de LightEdge es proporcionar a los operadores una plataforma MEC que aporte inmediatamente las ventajas de la computación *edge* a los usuarios finales de 4G. Su mayor punto fuerte es su transparencia respecto al resto de componentes de la red 4G, permitiendo no tener que realizar ninguna modificación en el entorno del operador de telefonía salvo las funciones de cobro. Sigue la arquitectura *bump in the wire* propuesta por la ETSIT [27] explicada en la sección 2.1.4. Sitúa el MEC entre el RAN y el EPC del sistema 4G (interfaz S1), interceptando todos los mensajes intercambiados entre ambos.

En la Figura 17 vemos el diseño de LightEdge. El host intercepta todos los paquetes del plano de control, encapsulados en paquetes SCTP (*Stream Control Transmission Protocol*); y todos los paquetes del plano de datos, encapsulados en paquetes GTP (*GRPS Tunnelling Protocol*). El tráfico de todas las aplicaciones, por defecto, se dirigen hacia internet y, por lo tanto, el host MEC debe de ser capaz de reencaminar dicho tráfico hacia las aplicaciones que pueden ejecutarse en local. Para ello tiene dos métodos:

- Utilizando un **servidor DNS local** que resuelva las consultas a dichas aplicaciones con direcciones IP locales
- Utilizando una **tabla de reenvío** de manera que se modifique el puerto y la IP del tráfico dirigido a una cierta dirección de internet, reenviándolo a aplicaciones dentro del MEC.

A continuación, se explica un poco más en detalle en qué consisten los diferentes elementos en los que se divide LightEdge:

La **plataforma MEC** es configurada por el LightEdge *MEC Platform Manager* a través de la interfaz Mm5 (interfaz no definida todavía por la ETSIT). Incluye un catálogo de los servicios y aplicaciones que pueden ser lanzadas en el MEC (*Service Registry*), un servicio que permite obtener información y controlar los recursos radio de las estaciones base a través de una interfaz propietaria (*Radio Network Information Service – RNIS*. Servicio que sí que existe de forma nativa en el MEC de 5G), un servicio MQTT que habilita la comunicación entre varios componentes de LightEdge, un gestor de reglas de tráfico que permite configurar el switch L3 para enrutar el tráfico que pasa por el host MEC y un servicio DNS que permite mapear las consultas de los UEs a direcciones IP locales dentro del dominio MEC.

El **servicio 4G-UPF** incluye un *switch* de nivel 3 encargado de guiar el tráfico entre el eNB, el EPC y los servicios MEC de manera que el tráfico de control SCTP lo envía al monitor SIAP y el tráfico de usuario GTP lo envía al servicio vGTP, un monitor SIAP que identifica la correspondencia de cada

UE con su TEID³ de subida y bajada correspondiente, y un servicio vGTP que se encarga de encapsular y desencapsular los mensajes GTP entre el eNB y el SGW. El servicio vGTP incluye una tabla en la que, en función del protocolo, la dirección IP y el número de puerto, puede sustituirlo por una dirección IP y puerto virtual para encaminar cierto tráfico a aplicaciones que se ejecutan dentro del MEC.

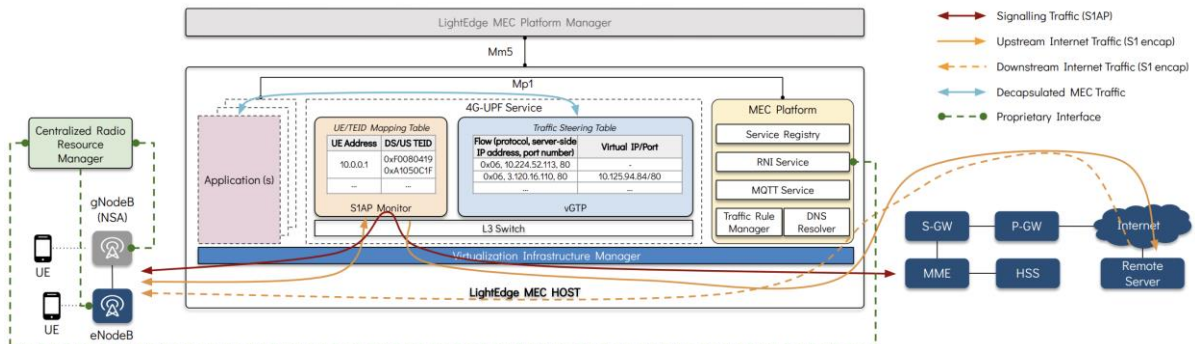


Figura 17. Arquitectura de referencia de LightEdge [38].

La aplicación ha sido diseñada para ser nativamente compatible con kubernetes. Cada uno de sus componentes se puede desplegar utilizando tecnologías de contenedores. Todas las aplicaciones MEC también pueden desplegarse como contenedores.

Los autores de esta herramienta realizaron una implementación práctica de ella para comprobar y demostrar sus posibilidades [6]. Se centraron en CCAM (*Connected, Cooperative, and Automated Mobility*) como caso de uso, principalmente en las funciones de vehículo autónomo que pueden ser descargadas en la red. Probaron dos casos de uso desarrollando las aplicaciones correspondientes (Figura 18):

- Seguimiento de líneas: Procesando de imágenes en tiempo real desde el servidor MEC provenientes de coches teledirigidos para controlarlos remotamente. Para realizarlo utilizaron la librería OpenCV [39]. Aplicaban un filtro gaussiano para reducir granularidad en la imagen, la transformaban al espacio de colores HSL (*Hue, Saturation and Lightness*) para extraer los tonos de las líneas y determinaban la curvatura de las líneas realizando una transformación de perspectiva a la imagen. Finalmente, se realizaba una interpolación de los puntos que conformaban la línea para estimar las curvas.
- Reconocimiento de objetos en la carretera: Detección de entidades con inteligencia artificial desde el servidor MEC.

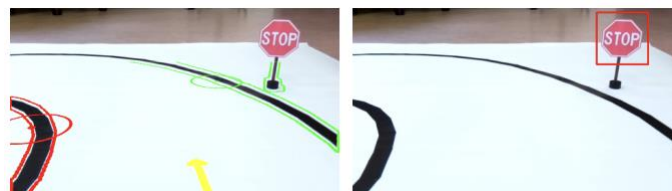


Figura 18. Aplicación de conducción autónoma desarrollada para testear LightEdge.

Implementaron una red 4G-LTE con srsRAN para la parte de la red RAN y nextEPC para la parte núcleo. Después, instalaron el *software* LightEdge, siguiendo la filosofía *bump in the wire* comentada

³ TEID (*Tunnal Endpoint ID*) es el identificador del túnel GTP que se crea entre el eNB y SGW. Existen dos TEIDs por UE, uno de subida y uno de bajada. Conociendo estos valores se puede identificar más fácilmente qué flujos de datos corresponden a qué UE.

anteriormente, en dicha red. Utilizaron el *software* controlador 5G-EmPOWER [40] que implementa las tareas de *Central Radio Resource Manager*, conectado a las estaciones base srsRAN y a LightEdge al servicio RNI (ver Figura 17). Este *software* permite controlar los recursos radio utilizados en función del servicio ofrecido, UE conectado u otros parámetros de calidad de servicio. Como núcleo utilizaron nextEPC [41], un *software* que funciona de forma similar a OAI-CN, srsEPC u Open5GS.

Compararon la latencia ofrecida por el servicio en función de si este se ejecutaba lejos del cliente o en el MEC y en función de cuantas instancias del servicio se ejecutaban simultáneamente. El *software* LightEdge estaba instalado en una máquina equipada con un procesador Intel Xeon E5-2666 con 4 vCPUs y 7,5GB de memoria RAM, mientras que para la ejecución en la nube utilizaron una configuración de 20 CPUs virtuales y 30 GB de memoria RAM alojadas en Irlanda (lejos del usuario final).

En la Figura 19 vemos algunos de los resultados obtenidos. Existe un balanceador de carga que lanza una segunda instancia de aplicación cuando la latencia excede un límite. Comprobamos como la latencia se ve aproximadamente dividida por dos gracias al uso del MEC con LightEdge. También vemos la relevancia que tiene el lanzamiento de varias instancias de la aplicación cuando se tienen múltiples clientes para mejorar la latencia del usuario final. Vemos que son capaces de ofrecer una latencia mínima de 50ms.

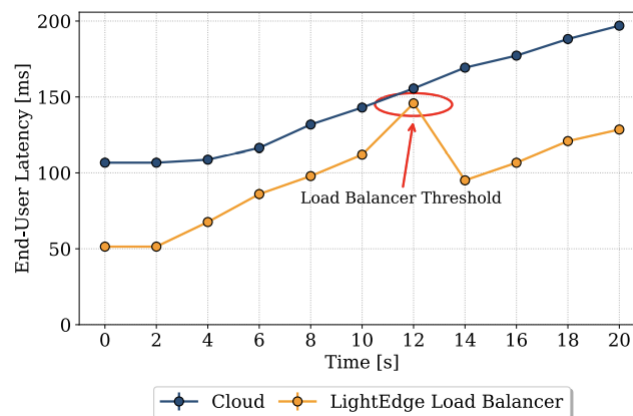


Figura 19. Latencia con el tiempo para un creciente número de consultas (uno cada 2seg).

3. Maqueta, vehículo y *software* de control

En este capítulo se explica todo el desarrollo, no relacionado con la red, realizado para la implementación del banco de pruebas. Este incluye: el diseño de la maqueta que permita evaluar multitud de casos de uso de vehículo conectado, es decir, la pista con carreteras por las que circulará nuestro vehículo (3.1); el análisis del vehículo disponible junto con las modificaciones físicas para adaptarlo a la maqueta y darle conectividad 4G y explicación del nuevo *software* desarrollado para que pueda ser controlable desde el exterior y pueda ejecutar el algoritmo de guiado (3.2); el desarrollo de la aplicación del servidor desarrollada para la ejecución del algoritmo de guiado en la red (3.3), y el desarrollo del algoritmo de conducción autónoma, que permite al vehículo circular autónomamente a través de la maqueta, pudiendo ejecutarse tanto dentro del vehículo como en un servidor externo (3.4).

3.1. Maqueta

La maqueta es una parte fundamental del banco de pruebas. Consiste en la pista a través de la cual se mueven todos los vehículos. Un correcto diseño es esencial para ofrecer al banco de pruebas flexibilidad, de forma que se puedan analizar una gran cantidad de casos de uso sin la necesidad de nuevos diseños. Para su diseño, hemos impuesto una serie de objetivos que esta debe cumplir:

- La maqueta debe de ser **suficientemente grande** para poder simular escenarios realísticos, como *handover* entre celdas de la red.
- La maqueta debe de ser **compleja**, incluyendo ambientes metropolitanos, es decir, **cruces, desvíos, incorporaciones y vías de varios carriles**, para que permita la evaluación de múltiples casos de uso. Por ejemplo, los desvíos e incorporaciones permitirán probar casos de uso relacionados con la capacidad de decisión entre vehículos, los cruces permitirán evaluar casos de uso de prevención de choques y semáforos inteligentes, y las vías de varios carriles permitirán el análisis de casos de uso relacionados con el platooning y adelantamientos.
- La maqueta debe de ser **modular**, de forma que pueda montarse y desmontarse con facilidad y puedan crearse nuevos circuitos con el material ya creado, sin la necesidad, en la medida de lo posible, de la creación de nuevas pistas.
- La maqueta debe **facilitar la implementación de un algoritmo de guiado** para convertir un vehículo en autónomo. Sin embargo, también debe de ofrecer la posibilidad de mejorar dicho algoritmo de conducción autónoma permitiendo optimizarlo y hacer pruebas para condiciones más realistas.

Siguiendo estos objetivos, hemos diseñado nueve piezas de diferentes tamaños, incluyendo curvas, carreteras rectas con uno o varios carriles, cruces y desvíos (Figura 20) a escala 1:16 aproximadamente. De esta manera, se puede crear una gran variedad de circuitos combinando las piezas de diferentes formas. La mayoría de las piezas no sobrepasan las medidas de 1,6x1,6m de manera que sea fácil su traslado y almacenaje. Para implementar varios carriles, se utilizan dos piezas especiales, una de 90cm de ancho que puede ser tan largo como sea necesario, que incluye 3 carriles en línea recta y otra de 360x160cm, dividida para mejor manejo, que permite empalmar dichos carriles con el resto del circuito. Todas las carreteras incluyen una línea central de diferentes colores que permite implementar el algoritmo de guiado explicado en la sección 3. A su vez, también incluyen líneas blancas laterales que permiten la implementación de algoritmos de guiado más realistas.

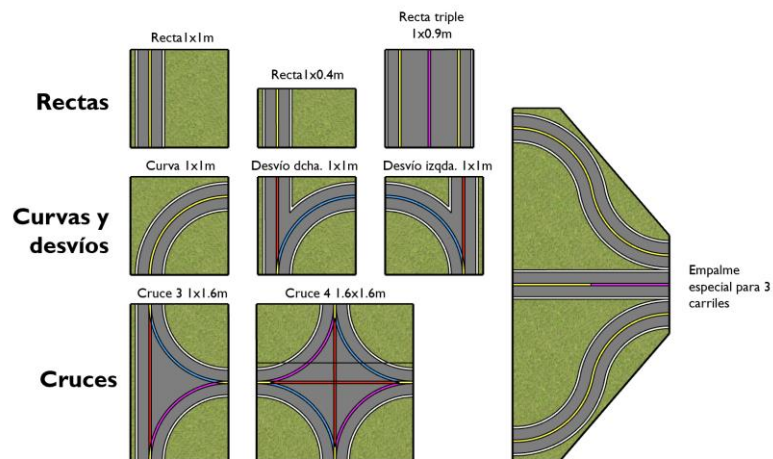


Figura 20. Conjunto de piezas diseñadas para la maqueta.

En la Figura 21 se muestran los diferentes materiales utilizados para la elaboración de la maqueta. En su primera versión se utilizó un material basado en goma negra con una línea pintada en su interior para conformar las carreteras. Se utilizó césped artificial para rellenar el resto de la pieza, evitando que el vehículo tuviera visión directa del suelo y, por lo tanto, permitiendo tener el mismo entorno alrededor de la carretera independientemente de la ubicación de la maqueta. Para darle robustez y portabilidad a las piezas, se montó tanto la goma como el césped artificial sobre planchas de cartón que, como vemos en la foto, también servía para marcar las líneas laterales de la carretera.

Al probar esta implementación, si bien en algunas condiciones funcionaba correctamente, existían muchos problemas relacionados con los reflejos que ofrecía la goma negra. Estos reflejos impedían la correcta visión de la carretera por el vehículo y, por lo tanto, impedía el correcto guiado. Se probó a lijar la goma y a añadir láminas polarizadas en la cámara de los vehículos, métodos que, si bien mejoraban el funcionamiento, no eran lo suficientemente eficaces. Por otro lado, el cartón, utilizado como base de las piezas, era demasiado frágil y se acababa deformando. Por estos motivos se decidió analizar el uso de otros materiales para crear una segunda versión de la maqueta.

En la segunda versión de la maqueta, se sustituyó la goma negra pulida de la primera versión por fieltro negro, material que ofrece muchos menos reflejos, es menos pesado y más manejable. Se decidió utilizar un único color en la línea central, lo que reduce significativamente los problemas en el algoritmo de visión del vehículo. A su vez, también se sustituyeron las piezas de cartón por plástico blanco, ofreciendo mayor robustez a las piezas y permitiendo que las líneas laterales de la carretera fueran blancas, tal y como lo son en la realidad.

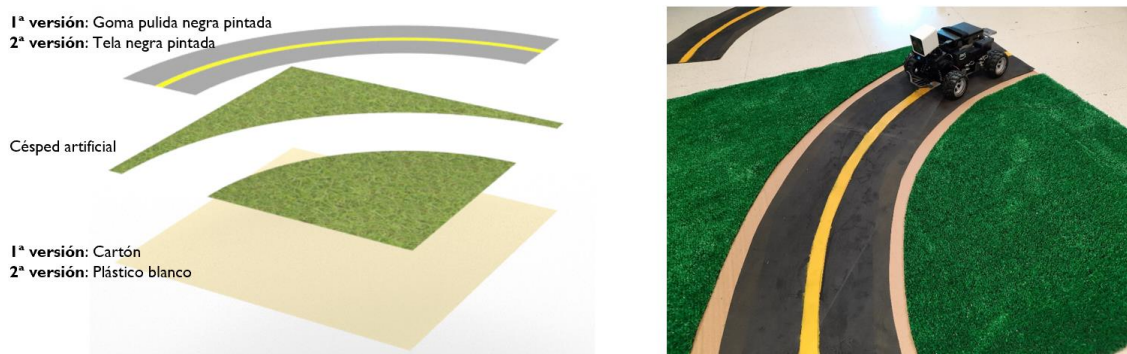


Figura 21. Materiales utilizados para la elaboración de la maqueta.

En la Figura 22 se pueden observar las medidas principales de dichas piezas para la versión 1. Los carriles comprenden siempre una anchura de 30cm, incluyendo líneas blancas laterales de 2,5cm y una línea central de diferentes colores de 2,5cm. El radio de giro de todas las curvas es de 90cm, debido principalmente a las limitaciones que impone el vehículo, como se explica en la sección 3.2.

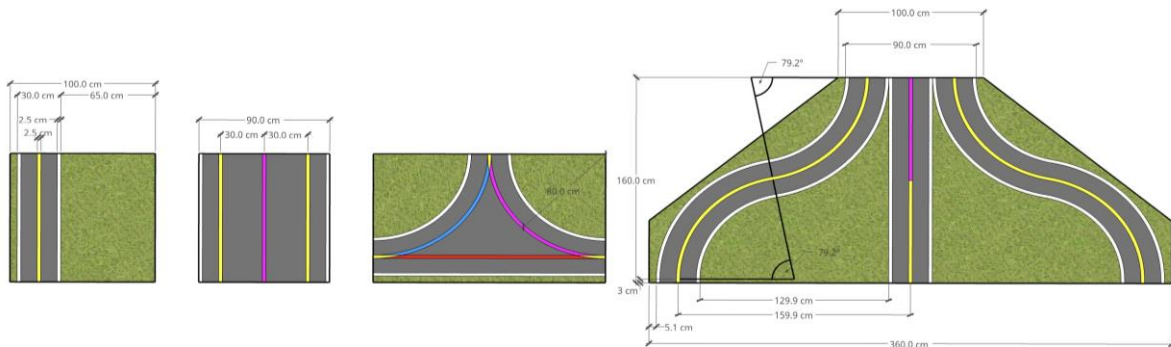


Figura 22. Medidas de acotadas de algunas de las piezas diseñadas para la versión 1 de la maqueta.

Por otro lado, en la Figura 23 se observan las medidas principales para la versión 2 de la maqueta. Vemos que son muy parecidas, variando únicamente la anchura de las líneas blancas laterales, las cuales pasan de 2,5cm a 1,5cm para ser más fieles a las marcas viales reales, a escala 1:16 (escala del vehículo).

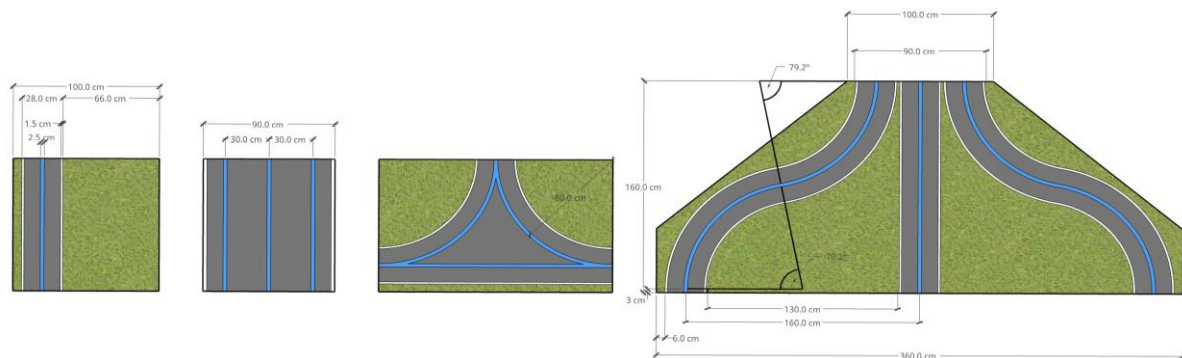


Figura 23. Medidas de acotadas de algunas de las piezas diseñadas para la versión 2 de la maqueta.

En la Figura 24 se muestran dos ejemplos de montaje de la maqueta, uno de la primera versión y otro de la segunda versión. Observamos a simple vista como las piezas de la segunda versión no están deformadas gracias al uso de plástico en vez de cartón, como sus carreteras ofrecen menores reflejos y como se utiliza un único color como línea central de las carreteras.

En el Anexo I se puede encontrar la colección de piezas completas acotadas de la versión final de la maqueta.



Figura 24. Ejemplos de montaje de la primera (izquierda) y segunda (derecha) versión de la maqueta.

3.2. Vehículo

Los vehículos de este banco de pruebas deberán poseer una serie de características que permitan probar la mayor cantidad de casos de uso experimentales posibles, asemejándose a los vehículos reales. Por lo tanto, deben de cumplir los siguientes objetivos:

- Deben ser vehículos a escala que puedan emular los sistemas de **sensórica** de los vehículos autónomos en la actualidad.
- Deben ser capaces de incorporar **múltiples tecnologías de comunicaciones**.
- Deben integrar un **ordenador programable** capaz de controlar el vehículo longitudinal y lateralmente y recibir información de todos los sensores.

El vehículo elegido ha sido el Amazon DeepRacer Evo [11]. Este vehículo se vende con el objetivo de realizar carreras de coches autónomos mediante inteligencia artificial. Incluye un LiDAR 2D con un radio de escaneo de 360 grados de 12 metros, un par de cámaras con gran angular de 4MP que permiten visión en estéreo, conexión Wi-Fi 802.11ac, 3 puertos USB, una IMU (Unidad de Medida Inercial) Bosh BMI 160 [42] con giroscopio y acelerómetro y un ordenador que ejecuta Ubuntu 16.04.3 LTS con 4GB de RAM y un procesador Intel Atom a 1,6GHz de velocidad. Alcanza velocidades de 4 m/s y su radio de giro es de 70cm.

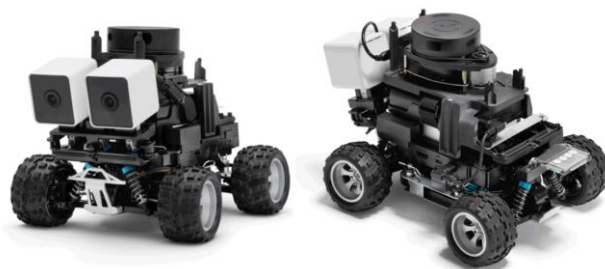


Figura 25. Vista frontal y trasera del vehículo Amazon DeepRacer Evo [11].

Las ventajas de este vehículo son múltiples ya que obtenemos en un mismo paquete un vehículo motorizado a escala con sensor LiDAR, cámaras y un ordenador embebido que ejecuta Linux. Sin embargo, el uso de este vehículo presenta varios inconvenientes. No incorpora ningún sistema de medición de la velocidad (aunque dispone de un parámetro que puede utilizarse para mantener una velocidad desconocida constante), ni está equipado con un sistema de geoposicionamiento. Además, el radio de giro de 70 cm a escala 1:16 equivale a un radio de 11,2 metros en escala real, que es

mucho mayor que el radio de giro de la mayoría de vehículos utilitarios (unos 5 metros). Este factor impone la condición del radio de giro de las curvas de la maqueta.

En cuanto a las tecnologías de comunicación, el vehículo sólo dispone de tecnología Wi-Fi. Sin embargo, posee varios puertos USB, permitiendo así la conexión de módems 4G y 5G. La única tecnología de comunicación que aún no hemos podido acoplar con éxito es la de Comunicaciones Dedicadas de Corto Alcance (DSRC), ya que, hasta donde sabemos, no existe ninguna versión de dispositivos USB para 802.11p que implemente el modo OCB (*Outside the Context of a Basic service set*). La tecnología DSRC es una de las opciones para las comunicaciones vehículo a vehículo y, por tanto, es crucial para algunas aplicaciones de sistemas de transporte inteligentes (ITS). El modo OCB permite transmitir mensajes en redes inalámbricas evitando los procesos de autenticación y asociación, que son las principales fuentes de retardo en las redes Wi-Fi. Así, aunque existen varios dispositivos USB Wi-Fi que permiten trabajar en frecuencias 802.11p, siguen necesitando realizar los procesos de autenticación y asociación.

3.2.1. Modificaciones físicas del vehículo

El vehículo incluye dos baterías de serie, una para alimentar su ordenador de 13600 mAh y otra de 1100 mAh (que llamaremos batería de impulso) para alimentar el motor eléctrico que impulsa el vehículo y los servos que giran la dirección. La batería de impulso, a parte de estar conectada a los motores, también está conectada al ordenador para que este pueda monitorizar su voltaje. Si bien la batería del ordenador permite usar el vehículo de continuo durante aproximadamente 5 horas, la batería de impulso se agota rápidamente y la potencia que ofrece es muy poco estable mientras se va descargando. Es por esto que se decidió sustituirla por otra de 6200 mAh que, si bien aporta mucha más autonomía, también es mucho más grande y pesada. Para montarla, fue necesario fabricar un nuevo cable para conectar el conjunto batería-motor-ordenador debido a que la nueva batería contaba con otro conector (Figura 26 y Figura 27).

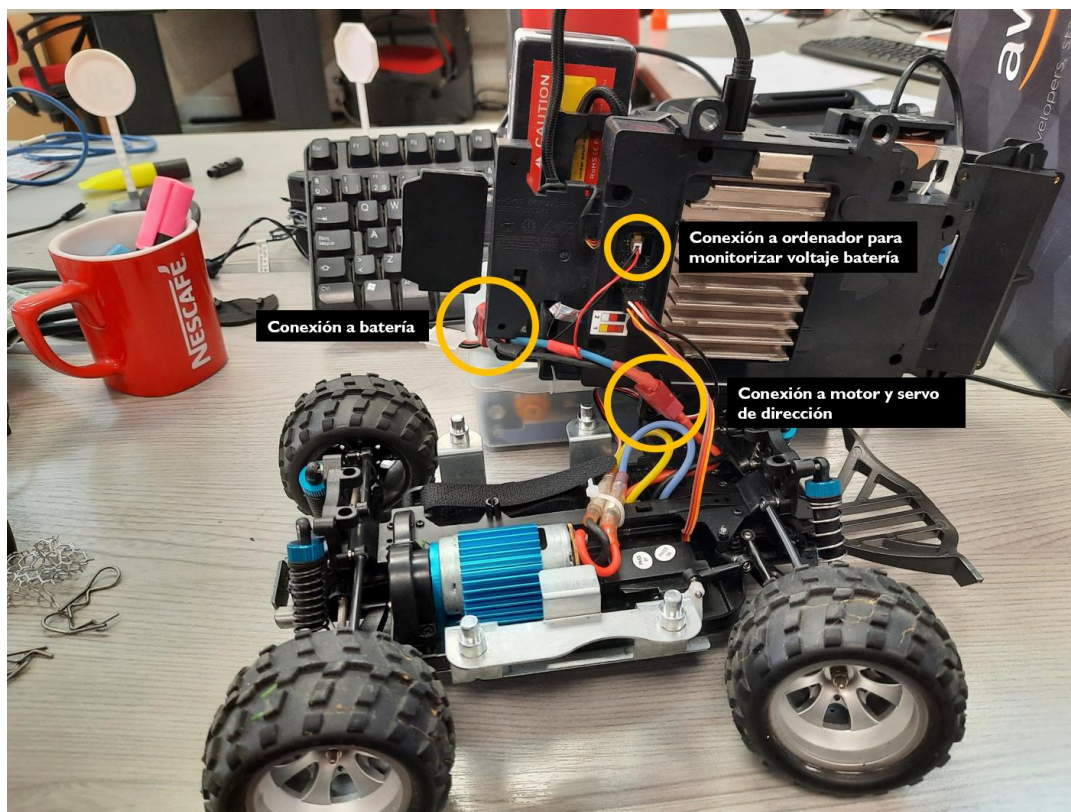


Figura 26. Vista interior del vehículo con conexiones de la batería de impulso.

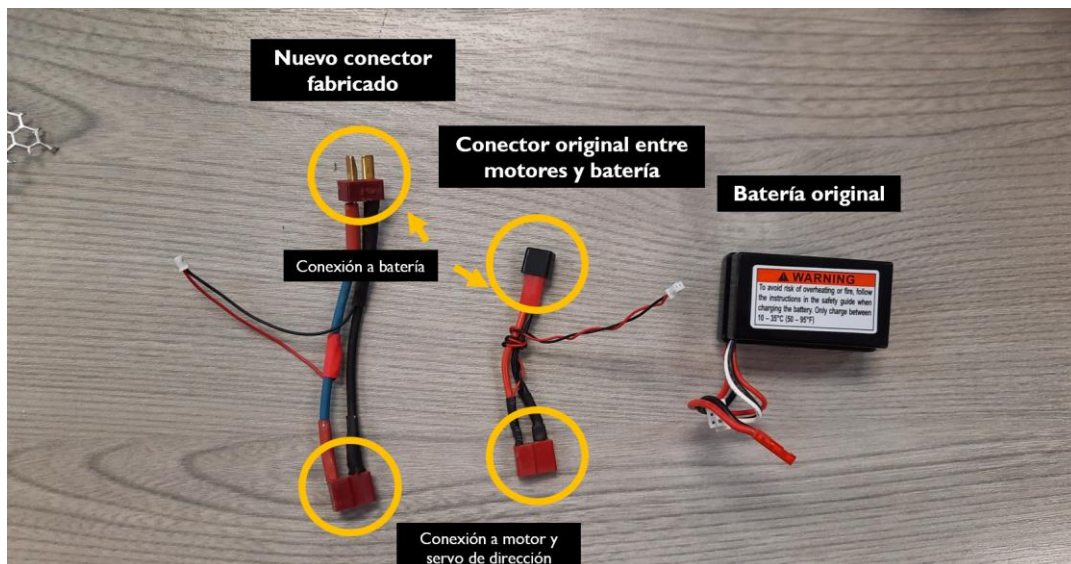


Figura 27. Nuevo cable conector conjunto batería-motor-ordenador, cable original y batería original.

Debido al peso de dicha batería, fue necesaria la sustitución de los muelles para la suspensión trasera del vehículo por unos más fuertes, como se puede ver en la Figura 28.

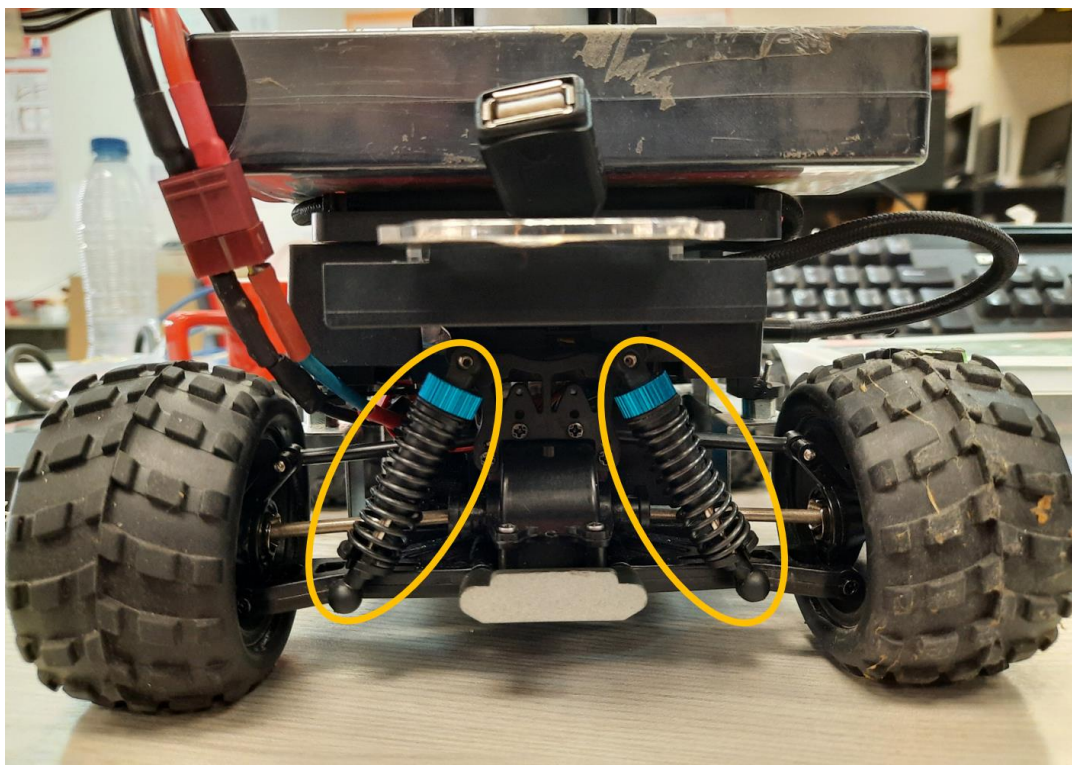


Figura 28. Nuevos muelles montados en el vehículo.

Finalmente, se aprovechó el puerto USB que dispone en la parte trasera (junto con un alargador también incluido) para conectar un módem 4G Huawei E3372 [43] (Figura 29).



Figura 29. Nueva batería y módem 4G instalado.

3.2.2. *Software original*

Tal y como se ha comentado anteriormente el ordenador del vehículo viene con el sistema operativo Ubuntu 16.04.3 LTS preinstalado. Además, viene con el sistema ROS (Robot Operating System) en su versión Kinetic [44].

ROS es un conjunto de librerías y herramientas *software* pensadas para ayudar a los desarrolladores a crear aplicaciones de robots. Ofrece los servicios estándar de un sistema operativo, tales como abstracción de *hardware*, controladores de dispositivos, visualizadores, gestión de mensajes entre procesos, y gestión de paquetes entre otros. Si bien el objetivo primario de ROS es soportar la reutilización de código en la investigación y el desarrollo dentro de la robótica, este aporta muchas otras ventajas.

Los procesos programados para ser ejecutados en ROS son denominados nodos. Estos nodos pueden comunicarse entre sí a través de servicios, comunicaciones del estilo RPC (Remote Procedure Call) síncronas; o a través de tópicos siguiendo la filosofía publicador-suscriptor, donde un nodo publica en un tópico y varios nodos pueden recibir la información publicada suscribiéndose a dicho tópico. De esta manera, varios procesos pueden acceder a la información publicada por uno en tiempo real simultáneamente, facilitando enormemente la programación de los nodos al no necesitar utilizar colas de trabajo, tuberías o variables en memoria compartida.

En el vehículo, para poder acceder a los comandos ROS que permiten obtener información de los nodos, servicios o tópicos del vehículo (entre otras posibilidades), es necesario importar una serie de variables de entorno en el *shell* desde el que trabajemos. Para ello, existe un *script* predefinido en el vehículo llamado *setup.bash*, dentro de la carpeta */opt/aws/deepracer/*. Será necesario, por tanto, escribir el siguiente comando:

```
source /opt/aws/deepracer/setup.bash
```

Tras ejecutarlo, podemos controlar el entorno ROS mediante la colección de comandos ROS [45].

En la Tabla 6 se pueden observar los nodos preinstalados en el vehículo junto con los tópicos en los que publica cada uno de ellos, los tópicos a los que están suscritos y el nombre de los servicios que ofrecen. Hay que recalcar que esta tabla está obtenida en un estado concreto del vehículo, que corresponde al control manual del vehículo a través de su interfaz web. En otros estados las suscripciones de los diferentes nodos pueden variar, según la necesidad de cada nodo en cada estado.

Tabla 6. Nodos ROS preinstalados en el vehículo junto a los tópicos en los que publican y a los que se suscriben, y los servicios que ofrecen.

Nodo	Publicaciones	Suscripciones	Servicios
/battery_node	/rosout		/battery_level /battery_node/get_loggers /battery_node/set_logger_level
/control_node	/rosout	/auto_drive /calibration_drive /manual_drive	/autonomous_throttle /car_state /control_node/get_loggers /control_node/set_logger_level /enable_state /get_car_cal /get_car_led /model_state /set_car_cal /set_car_led
/inference_engine	/rosout		/inference_engine/get_loggers /inference_engine/set_logger_level /inference_sate /load_model
/media_engine	/display_mjpeg /rosout /video_mjpeg		/media_engine/get_loggers /media_engine/set_logger_level /media_state
/model_optimizer	/rosout		/model_optimizer/get_loggers /model_optimizer/set_logger_level /model_optimizer_server
/navigation_node	/auto_drive /rosout	/rl_results	/load_action_space /navigation_node/get_loggers /navigation_node/set_logger_level /navigation_throttle
/rosout	/rosout_agg	/rosout	/rosout/get_loggers /rosout/set_logger_level
/rplidarNode	/rosout /scan		/start_motor /stop_motor /rplidarNode/get_loggers /rplidarNode/set_logger_level
/sensor_fusion_node	/overlay_msg /rosout /sensor_msg	/scan /video_mjpeg	/configure_lidar /sensor_data_status /sensor_fusion_node/get_loggers /sensor_fusion_node/set_logger_level
/servo_node	/rosout		/get_cal /get_led_state /servo_cal /servo_gpio /servo_node/get_loggers /servo_node/set_logger_level /servo_state /set_led_state /set_raw_pwm
/software_update	/rosout		/begin_update /console_upload_model /get_device_info /get_otg_link_state /software_update/get_loggers /software_update/set_logger_level /software_update_get_state /software_update_status /verify_model_ready
/web_video_server	/rosout	/display_mjpeg	/web_video_server/get_loggers /web_video_server/set_logger_level
/webserver	/manual_drive /rosout		/webserver/get_loggers /webserver/set_logger_level

En la Figura 30 se puede ver de manera gráfica los diferentes nodos que se ejecutan en el vehículo y cómo se comunican entre ellos a través de tópicos. Es importante recordar que también se comunican a través de servicios, método no mostrado en la figura.

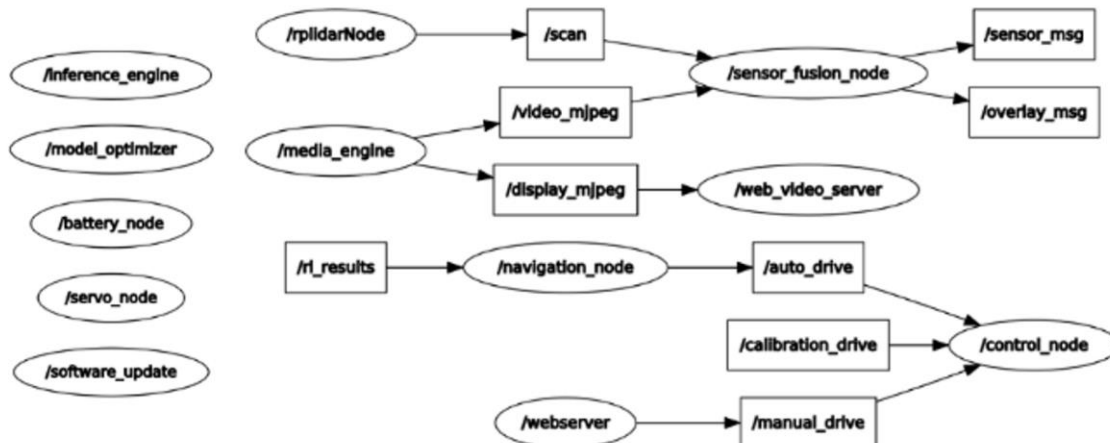


Figura 30. Nodos y tópicos utilizados en el *software* original del vehículo (exceptuando el nodo */rosout*).

Si bien el *software* tiene muchos nodos, a continuación, se van a explicar los más relevantes.

- ***/rplidar***: Controlador del LiDAR incluido en el vehículo (RPLIDAR AI de Slamtec [12]). Envía todos los datos que genera a través del tópico ***/scan***. Incluye los servicios ***/stop_motor*** y ***/start_motor*** para detener o arrancar el LiDAR.
- ***/media_engine***: Gestiona las imágenes recibidas por las cámaras conectadas al vehículo. Envía dichas imágenes en tiempo real a través del tópico ***/video_mjpeg*** y ***/display_mjpeg***, este último con mayor compresión utilizado para ser enviado a través del servidor web con el nodo ***/web_video_server***. Se puede configurar o comprobar su estado a través del servicio ***/media_state***.
- ***/sensor_fusion_node***: Fusiona la información proveniente del LiDAR con la información de la cámara. Ofrece sus resultados en tiempo real a través de los tópicos ***/sensor_msg*** y ***/overlay_msg***. A través de los servicios ***/configure_lidar*** y ***/sensor_data_status*** se puede configurar y recibir su estado.
- ***/webservice***: Gestiona el servidor web del vehículo y publica en ***/manual_drive*** los mensajes de control del vehículo (aceleración y ángulo de la dirección).
- ***/control_node***: Controla el motor y la dirección del vehículo. Recibe órdenes a través del mensaje ***/manual_drive*** o ***/auto_drive*** en función del modo. Permite modificar diversos parámetros del nodo a través de sus servicios.
- ***/servo_node***: Gestiona tanto el estado de las luces LED del vehículo como la calibración del motor y la dirección. Permite modificar dicha calibración y el estado de las luces a través de los servicios ***/servo_cal*** y ***/set_led_state***. Más información acerca de la calibración del vehículo se puede encontrar en el Anexo II.

3.2.3. *Software* para la gestión de aplicaciones de conducción.

Antes de analizar el nuevo *software* desarrollado en el vehículo, es necesario **deshabilitar el *software* de fábrica** del vehículo que no vamos a utilizar. Su deshabilitación es necesaria por dos motivos principales. Por un lado, algunos de los nodos preinstalados pueden ocasionar interferencias para el control del vehículo; por otro lado, se prefiere liberar al ordenador del vehículo de cualquier carga de procesamiento innecesaria para que ofrezca menores latencias y su velocidad de procesamiento sea menos relevante a la hora de analizar parámetros de funcionamiento en los casos de uso.

Para ello se deshabilitan los siguientes nodos ***/interference_engine***, ***/model_optimizer***, ***/navigation_node*** y ***/sensor_fusion_node*** debido a que no se utilizará ninguna herramienta de aprendizaje automático ofrecida por Amazon para el guiado del vehículo. También se deshabilitan los

nodos `/webserver` y `/web_video_server` al no necesitar la utilización de la interfaz web preinstalada en el vehículo y el nodo `/software_update` al no querer instalar actualizaciones que pudieran dar problemas con la compatibilidad del nuevo *software* desarrollado para el vehículo. Con este fin, se comentan las líneas de lanzamiento de dichos nodos en el fichero `/opt/aws/deepracer/share/launch/deepracer.launch`. De esta forma, el vehículo, al arrancar el servicio `deepracer-núcleo.service`, servicio que lanza todos los nodos preinstalados en el vehículo, sólo se lanzarán el resto de nodos que no hemos deshabilitado (los relacionados con el control del vehículo y recepción de imágenes y sensórica).

El nuevo *software* instalado en el vehículo se ha desarrollado con 3 objetivos en mente:

1. Capacidad de **comunicarse con el vehículo** desde un servidor externo, permitiendo enviarle órdenes y recibiendo información de él. Esta capacidad debe permitir ordenar al vehículo que solicite cualquier servicio que se programe en la red o que ejecute cualquier proceso programado en local.
2. Ejecución de un **algoritmo de guiado autónomo del vehículo** con procesamiento en local.
3. Capacidad para ejecutar el algoritmo de guiado autónomo externamente. Esto implica la capacidad de que **el vehículo sea guiado desde el exterior en tiempo-real**, pudiendo el vehículo enviar la información de sus sensores y cámaras en tiempo real a un servidor de aplicaciones externo y permitiendo la recepción de mensajes de control para operar la aceleración y la dirección de este.

Estos tres objetivos se han cumplido mediante la programación de tres nuevos nodos ROS, uno por objetivo, que se integran con el resto de los nodos instalados de fábrica. Es importante recalcar 3 aspectos fundamentales que engloban a los 3 nodos:

- Están programados en Python 3.7 [46], importando las librerías de ROS que permiten trabajar en dicho ecosistema.
- Utilizan un fichero de configuración denominado `vehicle.conf` donde se encuentran diversos parámetros de configuración relevantes para todos los nodos (ejemplo de dicho fichero comentado en el Anexo III)
- Una vez instalados, se lanzan a la vez a través del servicio `Artemis.service`, de manera que, siempre que este servicio esté en ejecución, los 3 nodos lo estarán.

A continuación, se explica en profundidad cada nodo desarrollado. En la página GitHub del Grupo de Comunicaciones Ópticas de la Universidad de Valladolid se puede encontrar un repositorio llamado Artemis [47] con más información del *software* instalado en el vehículo, scripts e información de instalación, y otros datos interesantes.

3.2.3.1. Nodo de comunicaciones con el administrador

Este nodo se encarga de gestionar el **envío y recepción de mensajes con el administrador** que permitan, por un lado, enviar órdenes al vehículo (mantenerse a la espera de nuevas órdenes, entrar en modo autónomo, ordenar la transmisión de imágenes y sensórica a un servidor, etc.) y, por otro lado, la transmisión de mensajes informativos desde el vehículo (estado de la batería, información de la IMU, calidad de la conexión, etc.). Es el nodo principal ya **que gestiona en todo momento el estado del vehículo y coordina el resto de nodos** para que realicen las tareas correspondientes a dicho estado.

Para establecer este canal de comunicación, se ha decidido utilizar el protocolo de mensajería MQTT [48], protocolo pensado para IoT que utiliza una arquitectura cliente-servidor basada en publicaciones y suscripciones en tópicos. En esta arquitectura, el servidor MQTT, de aquí en adelante conocido como bróker MQTT, es el nodo central que se encarga de conmutar todos los paquetes entre

diferentes clientes. Los clientes transmiten información publicando mensajes ligados a un tópico y reciben información suscribiéndose a un tópico. En este proyecto, dicho protocolo nos facilita la comunicación con los vehículos, permitiendo la comunicación bidireccional con varios vehículos simultáneamente a través de una única interfaz de control.

Se han definido una serie de tópicos vinculados a mensajes para efectuar dicha comunicación. Los mensajes que puede enviar el vehículo al administrador son los siguientes Tabla 7:

Tabla 7. Mensajes MQTT enviados por el vehículo.

Tópico	Mensaje	Tipo
{ID vehículo}/type_of_conection	4G/5G o Wifi	Espontáneos. Se envían cuando se establece con éxito la comunicación MQTT entre el vehículo y el bróker.
{ID vehículo}/public_ip	Dirección IP externa ("pública") del vehículo	
{ID vehículo}/mqtt_server_port	Puerto del bróker	
{ID vehículo}/cloud_server_ip	Dirección IP del servidor que ofrece la aplicación de control del vehículo	
{ID vehículo}/cloud_server_port	Puerto del servidor que ofrece la aplicación de control del vehículo	Espontáneo. Se envía cuando se establece con éxito la comunicación MQTT entre el vehículo y el bróker o tras enviar comando de calibración (calibrate) al vehículo.
{ID vehículo}/steering_calibration	Parámetros de calibración del sistema de dirección	
{ID vehículo}/IMU	Datos del giroscopio del vehículo en tiempo real medido en m ² /s (subtópicos x, y, z)	Periódicos. Se envían cada segundo. Sólo enviados si se recibe el comando Get-Data . Se dejan de enviar si se recibe el comando Stop-Data .
{ID vehículo}/battery_level	Nivel de la batería de propulsión (valores del 0 al 10, -1 en caso de batería desconectada del ordenador).	
{ID vehículo}/signal	Nivel de señal recibida en dBm (sólo válido en conexiones WiFi)	

Es importante recalcar como todos los tópicos utilizados siempre se incluyen dentro de un tópico que define el ID del vehículo (siendo este un número entero: 1, 2, 3, ...). Esto está realizado de esta manera para poder acceder a la información de cada vehículo en paralelo de forma asequible.

Tal y como hemos dicho antes, este nodo es el encargado de gestionar el estado del vehículo y coordinar al resto de nodos en función de este. Los cambios de estado se deben a 3 factores principales: Arranque del servicio *Artemis.service*, envío de un comando por parte del administrador, o fallo de configuración o conexión del vehículo. Cada estado está ligado a una respuesta visual que ofrece el vehículo a través de los LEDs que incorpora en la parte trasera. Todos los comandos que puede enviar el administrador se envían a través del tópico {ID vehículo}/command. En la siguiente tabla (Tabla 8) se muestran los estados posibles del vehículo con su explicación; el comando MQTT, acción o causa que lanza dicho estado; y la reacción de las luces del vehículo en cada caso.

Tabla 8. Estados del vehículo.

Estado	Explicación	Comando, acción o causa para alcanzar el estado	Reacción de las luces del vehículo
Start-Up	El servicio <code>Artemis.service</code> comienza a ejecutarse. Todos los parámetros del fichero de configuración <code>vehicle.conf</code> se recogen, se calibra la dirección del vehículo según los parámetros guardados, y se intenta establecer la conexión con el bróker MQTT. Tras 2 segundos pasa al estado AM-OFF	Iniciar el vehículo presionando el botón y esperar durante 1 minuto.	Blanco 2 segundos
AM-Cloud	El vehículo transmite las imágenes y la información del LiDAR en tiempo real al servidor configurado en el fichero <code>vehicle.conf</code> , y espera a recibir mensajes de control de guiado.	AM-Cloud	Azul
AM-Off	El vehículo espera nuevas órdenes a través de MQTT. Mantiene su posición.	AM-Off	Amarillo
AM-Local	El vehículo comienza a procesar las imágenes y el control del vehículo de acuerdo con el algoritmo de guiado autónomo instalado en él.	AM-Local	Verde
Calibration	El vehículo calibra el sistema de dirección con los nuevos parámetros recibidos a través de MQTT. Estos nuevos parámetros se guardan en el fichero de configuración <code>vehicle.conf</code> . Después vuelve al estado anterior.	Calibrate maxvalue midvalue minvalue	Azul claro 1 segundo
Unknown command	El vehículo ha recibido un comando desconocido a través de MQTT. Después vuelve al estado anterior	Cualquier comando no especificado.	Rojo parpadeando dos veces
No connection	El vehículo no logra establecer conexión el bróker MQTT e intenta reiteradamente su establecimiento.	Vehículo desconectado de la red o fichero <code>vehicle.conf</code> mal configurado.	Rojo

Aparte de los comandos listados en la tabla anterior, existen otros dos comandos **Get-Data** y **Stop-Data** utilizados para ordenar al vehículo que envíen información periódica de la IMU, estado de carga de la batería, y nivel de señal en dBm recibida en caso de que el vehículo se comunique a través de WiFi.

El nodo de comunicaciones con el administrador se llama `admin_communications_node`. Este nodo implementa lo siguientes métodos para poder recoger toda la información necesaria para enviársela al administrador y para gestionar el estado de los otros nodos desarrollados (explicados en las siguientes subsecciones 3.2.3.2 y 3.2.3.3):

- A partir del fichero de configuración `vehicle.conf` conoce el id del vehículo, la dirección IP y puerto del bróker MQTT, la dirección IP y puerto del servidor que ofrece la aplicación de control del vehículo en red y los parámetros de la última calibración realizada.
- A partir de la clase auxiliar `net_artemis` recupera la IP pública del vehículo, el tipo de conexión (WiFi o 4G/5G), y los dBm de la señal en caso de que la conexión sea por WiFi (Ver Anexo IV).
- A partir de la clase auxiliar `bmi160` recupera la información de la IMU instalada en el vehículo (Ver Anexo IV).
- Se comunica con el nodo `/battery_node` para obtener el estado de carga de la batería a través del servicio `/battery_level`.
- Se comunica con el nodo `/servo_node` para cambiar el estado de los LEDs (a través del servicio `/set_led_state`) y calibrar la dirección (a través del servicio `/servo_cal`).

- Se comunica con los nodos de control autónomo del vehículo (`/autonomous_control_node`) y control externo del vehículo (`/cloud_control_node`) a través de los servicios `/enable_local_autonomous_control` y `/enable_cloud_control`. Estos servicios están asociados al tipo de servicio ROS estándar `SetBool`, el cual permite pasar como argumento una variable booleana, facilitando la orden de activar o desactivar el control del vehículo en local o externo.

En la Figura 31 se muestra la comunicación que realiza el nodo con el resto de nodos mediante servicios.

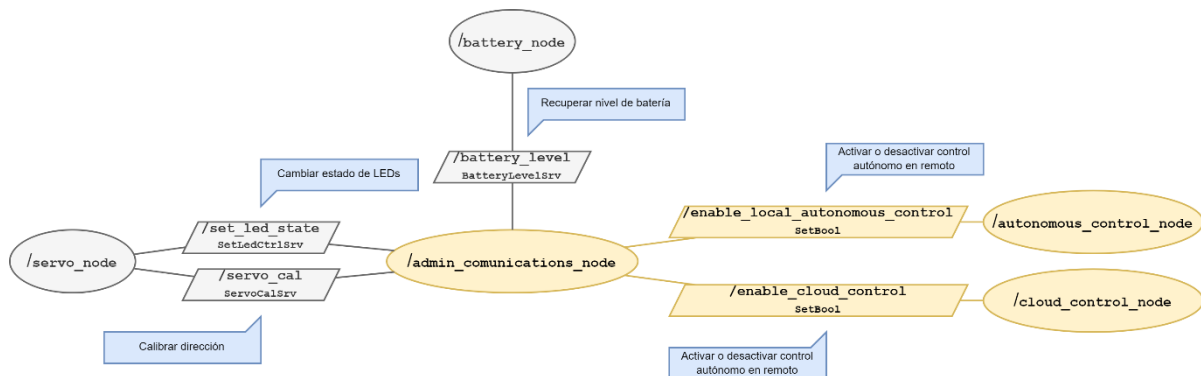


Figura 31. Comunicación del nodo `admin_comunications_node` con el resto de nodos mediante servicios. En gris nodos y servicios preinstalados en el vehículo. En naranja, nuevos nodos y servicios implementados.

En la Figura 32 se muestra un diagrama de flujo del funcionamiento del nodo, en el que se especifican las acciones que realiza desde su arranque. Tener en cuenta que el cliente MQTT instalado en el vehículo es capaz de detectar la pérdida de conexión con el servidor en cualquier momento gracias a unos mensajes ping enviados cada 5 segundos. Cuando el cliente MQTT detecta pérdida de conexión, salta un evento que permite, tal y como vemos en el diagrama de flujo, volver al estado inicial donde intenta establecer conexión.

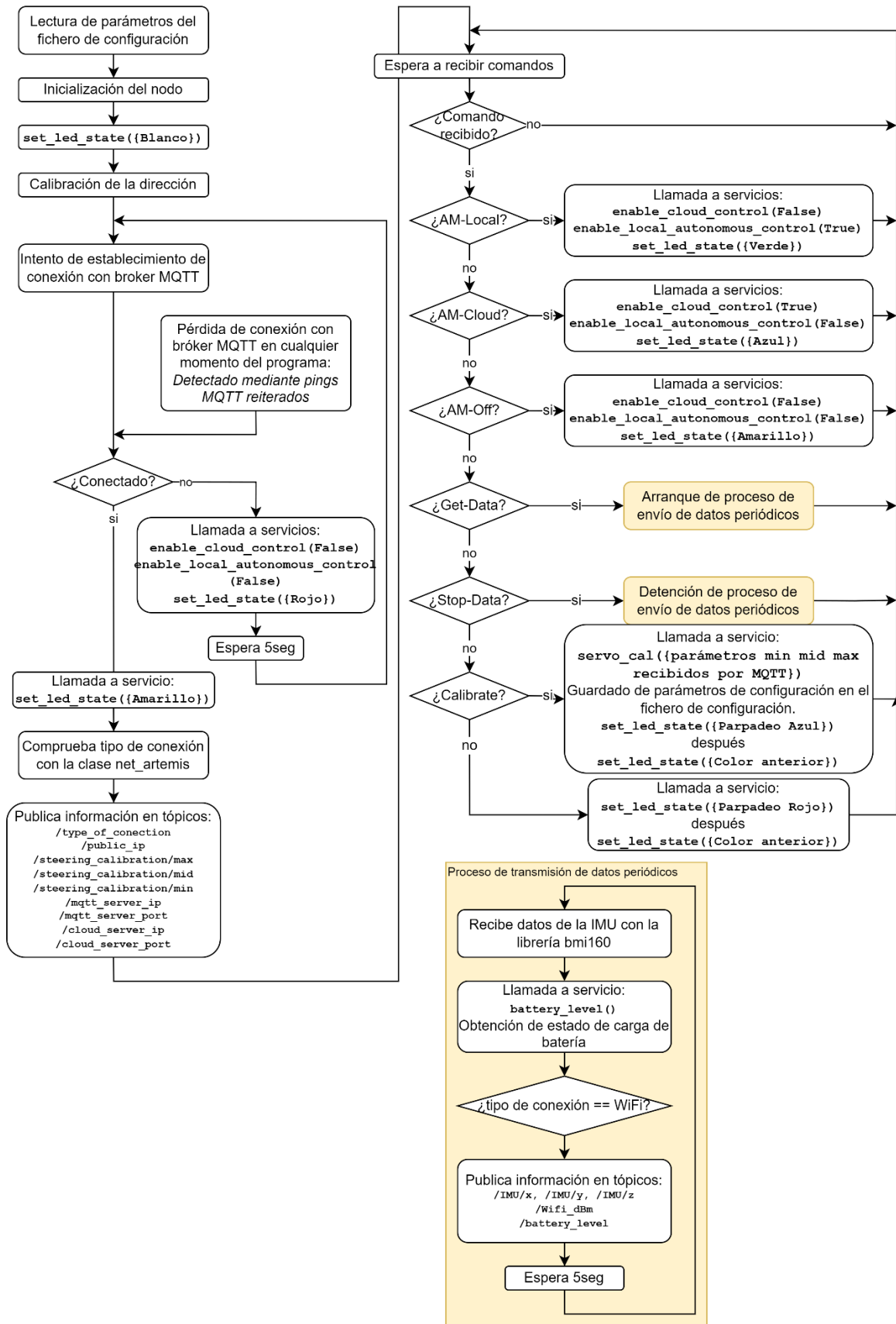


Figura 32. Diagrama de flujo simplificado del funcionamiento del nodo admin_communications_node.

3.2.3.2. Nodo de control autónomo del vehículo

Este nodo se encarga de **controlar el vehículo de forma autónoma con procesamiento en local** a través de la maqueta. Para ello sigue los siguientes pasos de forma reiterada:

1. Recibe imágenes de la cámara a través del tópico `/video_mjpeg` (tipo de mensaje `media_pkg/cameraMSG`) ofrecido por el nodo `/media_engine` y recibe datos del LiDAR a través del tópico `/scan` (tipo de mensaje `sensor_msgs/LaserScan`) ofrecido por el nodo `/rplidarNode`.
2. Utiliza la función `procesa_fotograma` de la clase `artemis_autonomous_car` para realizar el procesado de la imagen y de los datos del LiDAR y obtener los parámetros de control del vehículo en ese instante. Esta clase es la que implementa el algoritmo de guiado autónomo desarrollado y se explica en la sección 3. Utiliza la librería OpenCV [39] para el procesamiento de la imagen. Es por esto que, antes de enviar dicha imagen a la función de la clase, el nodo modifica el formato de esta para que sea compatible con el posterior procesado. Por otro lado, también utiliza la función `proceso_LIDAR` para enviar a la clase `artemis_autonomous_car` los datos del LiDAR.
3. Una vez recibidos los parámetros de control ofrecidos como variables de retorno de la función, los publica a través del tópico `/manual_drive` (tipo de mensaje `media_pkg/cameraMSG`), donde se especifica tanto el giro del vehículo (con una variable de -1 a 1), como la aceleración del vehículo (con una variable de 0 a 1). En caso de que el algoritmo de guiado autónomo no haya sido capaz de encontrar una trayectoria a seguir (informado mediante la variable booleana `trajectory_not_found` que también devuelve la función), el nodo envía parámetros nulos a través del tópico de control para detener el vehículo. Además, pone las luces del vehículo de color rosa para indicarlo.

Este nodo se llama en su implementación `autonomous_control_node`. Para controlar el inicio o apagado de este nodo, ofrece el servicio `/enable_local_autonomous_control` el cual incluye un parámetro booleano de entrada. De esta forma, cuando se llama al servicio con la variable `True` el nodo comienza a ejecutar los pasos explicados anteriormente y cuando se llama al servicio con la variable `False` el nodo se detiene y se mantiene a la espera.

A continuación, se muestra el diagrama de flujo de funcionamiento del nodo (Figura 33)⁴. Observamos como al llamar a la función `procesa_fotograma` se obtienen los parámetros de control `control_giro`, `control_aceelerador` y `trajectory_not_found`, y, en función del valor de `trajectory_not_found`, se utilizan los parámetros de control recibidos o, por el contrario, se detiene el vehículo. Si bien la recepción de los parámetros de control va condicionada a la recepción de una imagen y no al recibir datos del LiDAR, dichos datos también son utilizados por la clase `artemis_autonomous_car` para modificar el comportamiento del vehículo.

⁴ Tener en cuenta que, una vez que el nodo se suscribe a un tópico, la acción que desencadena la recepción de un mensaje por dicho tópico se ejecuta en paralelo al hilo principal del programa.

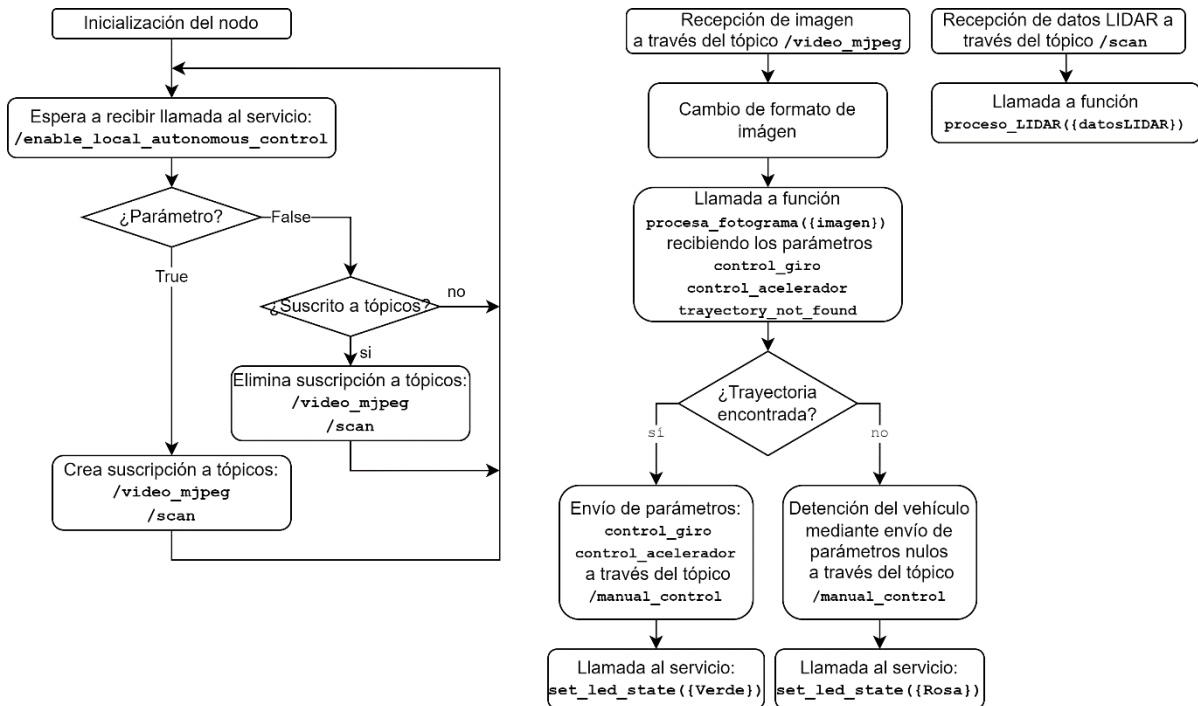


Figura 33. Diagrama de flujo simplificado del funcionamiento del nodo `autonomous_control1_node`,

3.2.3.3. Nodo de control externo del vehículo

Este nodo permite al vehículo ser controlado por un servidor externo a través de una conexión de red. Se encarga, por un lado, de **enviar las imágenes de la cámara, los datos del LiDAR y el estado de la batería en tiempo real a un servidor externo** y, por otro lado, de **recibir los parámetros de control del vehículo** y redirigirlos al nodo correspondiente para que el vehículo se comporte según dichas órdenes. Todo ello, persiguiendo lograr la menor latencia posible.

Para la transmisión y recepción de datos se ha decidido utilizar UDP sobre IP, permitiendo funcionar bajo cualquier tipo de red convencional, como redes móviles 3G/4G/5G o WiFi. Se decidió utilizar UDP frente a TCP ya que, para este proyecto, aporta una serie de ventajas:

- Es un protocolo **no orientado a conexión**, lo que facilita y agiliza la realización de pruebas, permitiendo que, en caso de pérdidas momentáneas de conexión entre vehículo y servidor, no sea necesario el restablecimiento de la conexión, sino que simplemente, al recuperar la conectividad, los mensajes UDP se vuelven a recibir directamente.
- Es un protocolo **sin retransmisiones**. Al no disponer de asentimientos, los paquetes perdidos no se reenvían. Si bien esto implica una comunicación no fiable, permite transmisión de datos en tiempo-real, permitiendo un *jitter* mucho más bajo.
- Es **más rápido y simple** que TCP, lo que permite una transmisión más rápida.

Para la transmisión y recepción de datos, estos se envían directamente serializados a través de UDP en el campo de datos de este protocolo. Sin embargo, debido a que se envían diferentes tipos de datos, es necesario implementar una capa de aplicación que permita al servidor identificar qué tipo de datos está recibiendo.

La capa implementada consiste en añadir un campo de tamaño de 1 byte al inicio del campo de datos UDP. De esta manera se puede enviar un **carácter de identificación** que defina el tipo de dato que se envía en la trama concreta. En la Figura 34 podemos ver el formato de las tramas utilizadas desde la cabecera IP. El byte 28 de la trama (correspondiente al primer byte de datos UDP) es el que define el tipo de trama en cuestión que se va a enviar.

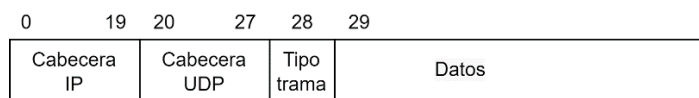


Figura 34. Formato de tramas enviadas UDP/IP.

En la Tabla 9 se muestran los tipos de tramas que se utilizan junto con el carácter de identificación que se añade al inicio del campo de datos UDP. Observamos como están definidos los mensajes para la transmisión de imágenes de dos cámaras (derecha e izquierda)⁵, del LiDAR, de la carga de la batería y de los parámetros de control.

Tabla 9. Tipos de tramas definidos con su carácter de identificación.

Tipo de trama	Carácter de identificación	Emisor	Contenido
Imagen cámara izquierda	'I' (Image)	Vehículo	Longitud de datos variable: Una imagen de la cámara del vehículo (la izquierda en caso de dos cámaras conectadas).
Imagen cámara derecha	'R' (Right)	Vehículo	Longitud de datos variable: Una imagen de la cámara derecha del vehículo (sólo si hay dos cámaras conectadas).
Datos LIDAR	'L' (LIDAR)	Vehículo	X bytes: Información del LiDAR de una vuelta 360° completa.
Carga de la batería	'B' (Battery)	Vehículo	0-3 bytes: Variable de tipo float del 0 al 10 que indica carga de la batería (-1 en caso de batería no conectada).
Parámetros de control	'C' (Control)	Servidor	0-7 byte: Variable double de control de giro (de -1 a 1) 8-15 byte: Variable double de control de acelerador (de 0 a 1)

Los pasos de funcionamiento de este nodo son similares a los del nodo anterior.

1. Recibe imágenes de la cámara a través del tópico `/video_mjpeg` (tipo de mensaje `media_pkg/cameraMSG`) ofrecido por el nodo `/media_engine` con una tasa de 30 fotogramas por segundo, y recibe datos del LiDAR a través del tópico `/scan` (tipo de mensaje `sensor_msgs/LaserScan`) ofrecido por el nodo `/rplidarNode`. Además, recibe el nivel de la batería en tiempo real gracias a llamadas periódicas al servicio `battery_level()`.
2. Compone las tramas UDP correspondientes a cada tipo de dato recibido y las envía a través de un socket UDP a la dirección IP y puerto configurado en el fichero `vehicle.conf`. En el caso de las imágenes de la cámara, tal y como el nodo anterior, modifica su formato para que sea compatible con las librerías OpenCV [39] y, además, las comprime en JPEG mediante herramientas de dicha librería, de manera que gasten un menor ancho de banda sin añadir un retraso considerable en la retransmisión.
3. Espera a recibir parámetros de control a través del socket establecido inicialmente. Una vez recibidos, los publica a través del tópico `/manual_drive` (tipo de mensaje `media_pkg/cameraMSG`), donde se especifica tanto el giro del vehículo (con una variable de -1 a 1), como la aceleración del vehículo (con una variable de 0 a 1). En condiciones normales, con un servidor bien configurado, debería recibir un mensaje de control por cada imagen enviada, es decir 30 mensajes de control por segundo.

Este nodo se llama en su implementación `c1oud_control_node`. Para controlar el inicio o apagado de este nodo, ofrece el servicio `/enable_c1oud_control` el cual incluye un parámetro booleano de entrada. De esta forma, cuando se llama al servicio con la variable `True` el nodo comienza a ejecutar los pasos explicados anteriormente y cuando se llama al servicio con la variable `False` el nodo se detiene y se mantiene a la espera.

⁵ En este Trabajo Final de Máster no se ha implementado la transmisión de datos de dos cámaras simultáneamente. Por lo tanto, el tipo de tramas R (imagen cámara derecha) no se ha utilizado.

En la Figura 35 se muestra el diagrama de flujo de funcionamiento del nodo. Es necesario tener en cuenta varios aspectos:

- Tanto con los mensajes ROS como con las peticiones de servicio recibidas, la acción que desencadenan se ejecuta en hilos paralelos al hilo principal del programa.
- Se utilizan dos temporizadores en este programa:
 - `timer_frame_not_received`: Temporizador de 0,2 segundos utilizado para detectar la no recepción de tramas de control por parte del servidor. Se reinicia cada vez que se recibe dicha trama, de manera que, si se deja de recibir, se detiene el vehículo automáticamente. Como en condiciones normales se deben recibir 30 mensajes de control por segundo, la no recepción de 6 mensajes de control consecutivos implicaría la detención del vehículo.
 - `timer_battery_level`: Temporizador de 2 segundos utilizado para llamar reiteradamente al servicio `get_battery()`.
- La variable `encendido` define cuando el nodo está activo y cuando no.
- El servicio `enable_cloud_control` devuelve una variable booleana que indica si el cambio ha sido realizado con éxito y un texto explicativo. No se realiza el cambio con éxito cuando se da una orden que ya había sido dada. Por ejemplo: ordenar la habilitación del control externo cuando ya estaba habilitado.

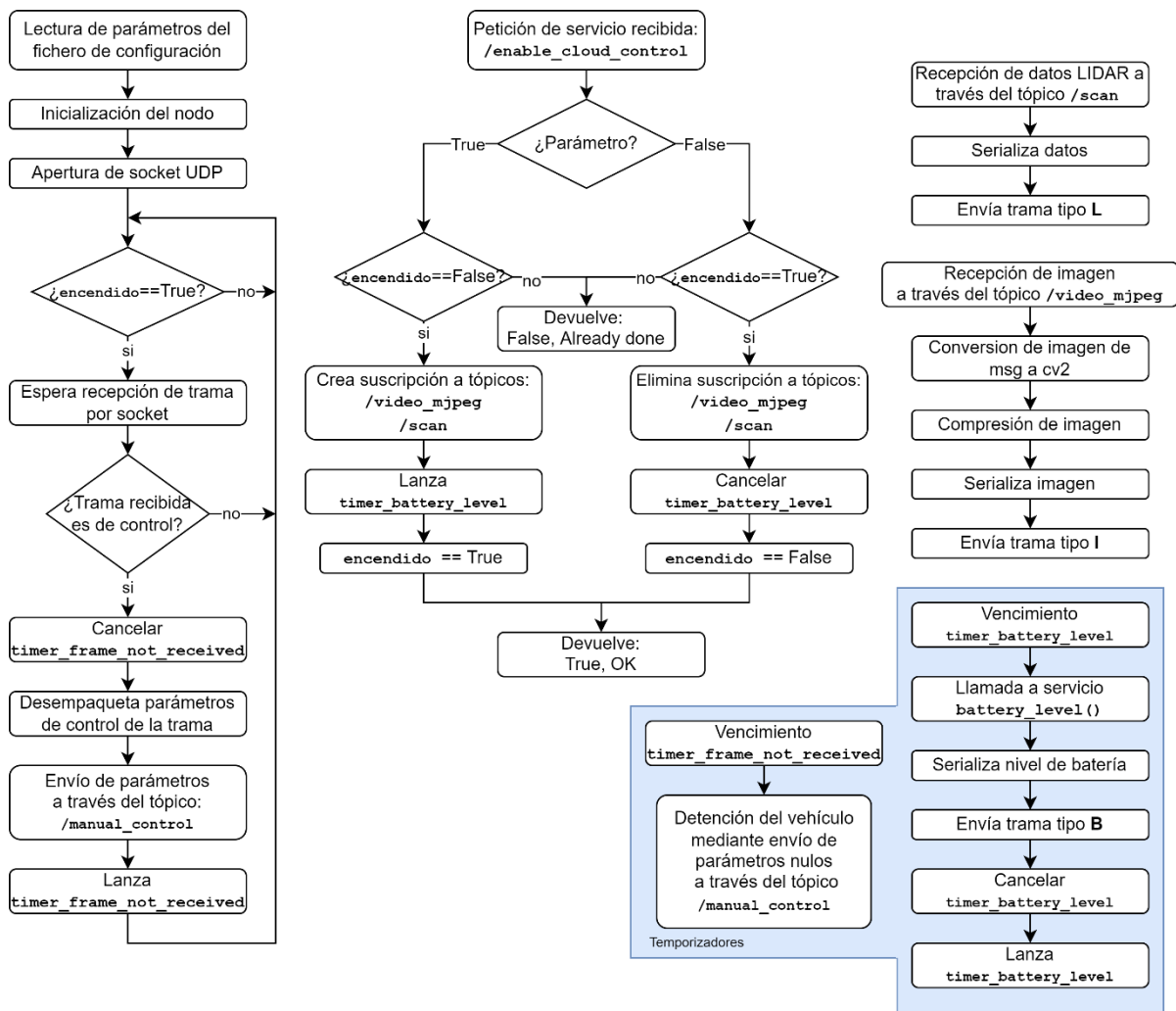


Figura 35. Diagrama de flujo simplificado del funcionamiento del nodo `cloud_control_node`.

En la Figura 36 se observa un resumen esquemático de las comunicaciones que tienen los vehículos con la red. Comprobamos como, por un lado, los vehículos se comunican a través de MQTT con el administrador y, por otro lado, se pueden comunicar simultáneamente con el servidor de aplicaciones, quien ejecuta el servicio de guiado autónomo.

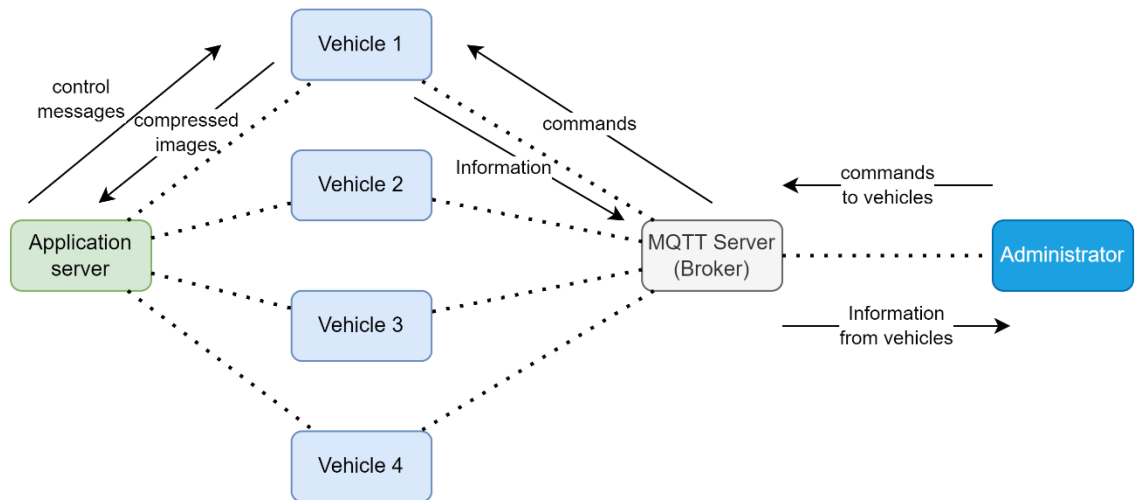


Figura 36. Diagrama de comunicación entre el administrador, los vehículos y el servidor de control.

3.3. Servicio de conducción autónoma en red

En esta sección se explica la aplicación desarrollada para ejecutar el algoritmo de guiado del vehículo en un servidor. Esta aplicación permite la realización del *off-loading* del procesado del control autónomo del vehículo en la red

Tal y cómo se ha explicado en la subsección 3.2.3.3, el vehículo es capaz de enviar la información del LIDAR y las imágenes en tiempo real a un servidor y al mismo tiempo recibir mensajes de control y actuar en función ellos, todo en tiempo real a través de unos mensajes UDP definidos y mostrados en la Tabla 9. El servicio de conducción autónoma en red realiza los siguientes pasos:

1. **Escucha** en una IP y puerto preconfigurado y **recibe y desencapsula** las tramas UDP de tipo imagen ("I"), datos LiDAR ("L") o carga de la batería ("B") (ver Tabla 9) de un vehículo emisor.
2. De la misma forma que lo hace el nodo de control autónomo del vehículo en local (explicado en la sección 3.2.3.2), utiliza la función `procesa_fotograma` de la clase `artemis_autonomous_car` para realizar el **procesado de la imagen** y de los **datos del LiDAR** y obtener los parámetros de control del vehículo. Esta clase es la que implementa el algoritmo de guiado autónomo desarrollado y se explica en la sección 3.
3. Una vez recibidos los parámetros de control ofrecidos como variables de retorno de la función, **encapsula** dichos parámetros en la trama UDP de tipo control ("C") (ver Tabla 9) y los **envía** a la dirección IP y puerto del vehículo que ha enviado el mensaje.

3.4. Algoritmo de control autónomo del vehículo

En esta sección se explica el algoritmo de control autónomo del vehículo desarrollado en este Trabajo Final de Máster. Este se divide en tres capas principales: **percepción** (subsección 3.4.1), donde se interpreta el entorno del vehículo, reconociendo las posibles trayectorias que puede seguir el vehículo; **planificación** (subsección 3.4.2), donde se decide qué trayectoria se debe tomar en cada momento (en el caso de bifurcaciones o cruces) en función de la ruta programada; y **control** (subsección 0), donde, una vez reconocida la trayectoria debe de seguir el vehículo, se ajustan los parámetros de control para que la siga correctamente, sin oscilaciones ni salidas. Todas estas capas están programadas en la clase `artemis_autonomous_car` y se apoyan en gran medida en la librería de

procesado de imagen OpenCV [39]. Las variables y funciones de dicha clase se explican en la subsección 3.4.4.

3.4.1. Percepción

La capa de percepción en este trabajo es la encargada de interpretar el entorno del vehículo y reconocer las posibles trayectorias que este puede seguir, identificando las carreteras que hay alrededor del vehículo y su ubicación respecto de este. Para que esta detección sea robusta y la programación del algoritmo no sea muy compleja, se decidió pintar unas líneas centrales en las carreteras de la maqueta por las que puede circular el vehículo.

A partir de las imágenes de la cámara, esta capa es capaz de devolver una matriz de puntos donde se encuentran posibles rutas a seguir delante del vehículo. Cada elemento de la matriz son puntos descritos mediante coordenadas respecto del vehículo de posiciones donde se han detectado tramos de carretera (línea de color). Los pasos son los siguientes:

1. La capa parte de imágenes RGB de 640x480px ofrecidas por la cámara. A estas imágenes se les aplica una **transformación de perspectiva**, lo que permite obtener una imagen a vista de pájaro de la parte frontal del vehículo, facilitando la obtención de parámetros de la trayectoria a seguir al eliminar la deformación de vista horizontal. La transformación está calibrada de forma que cada pixel de la imagen resultante, de 640x480px, equivalga aproximadamente a 2 milímetros cuadrados. En la Figura 37 se muestra un ejemplo de la transformación. Si bien en las partes más alejadas del vehículo se aprecian ciertas deformaciones, lo que es debido a la distorsión de ojo de pez que provoca la cámara gran angular del vehículo. Se desestimó corregir dicha deformación ya que, aunque la librería *OpenCV* ofrece funciones para realizar estas correcciones con una sencilla configuración, implicaba tiempos de procesamiento más grandes.

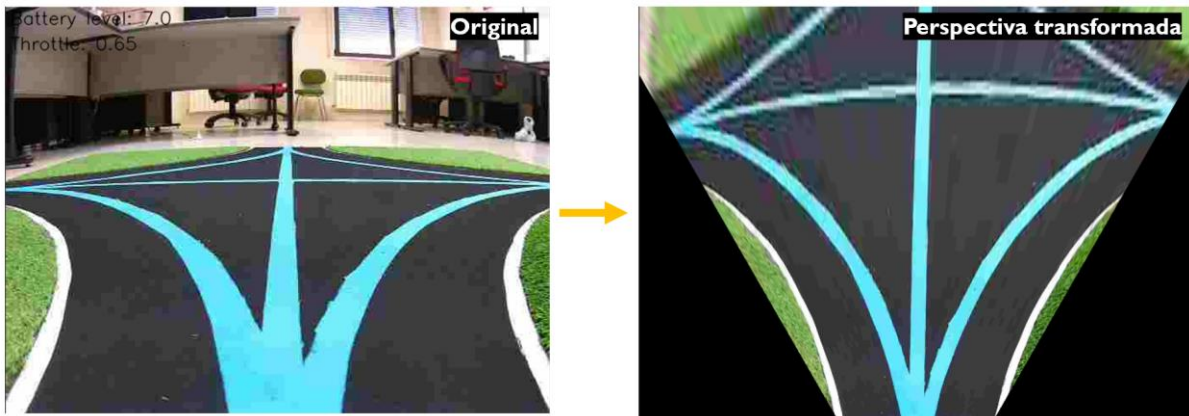


Figura 37. Ejemplo: Imagen original obtenida con la cámara del vehículo e imagen con perspectiva transformada.

2. Después se realiza una conversión de la imagen de RGB a HSV para después, realizar un **filtrado de color** con el que se obtiene únicamente las líneas centrales de las carreteras delante del vehículo.

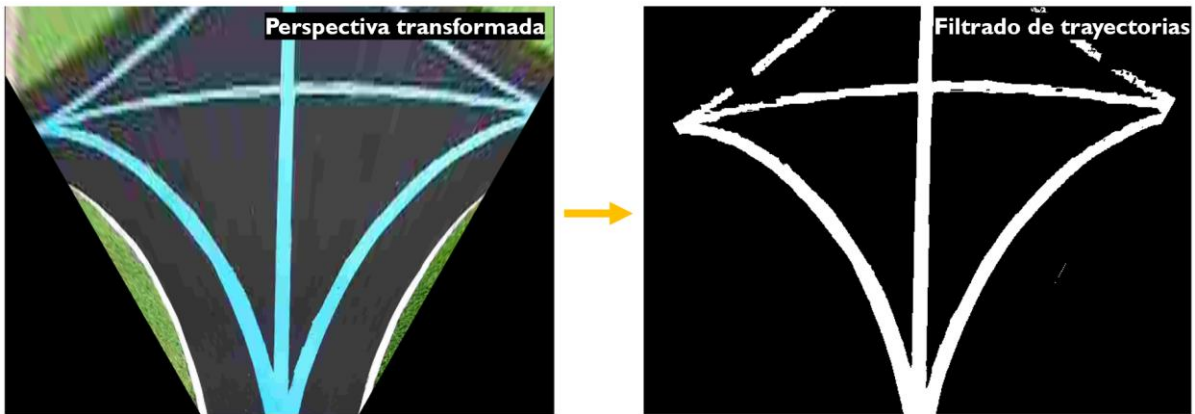


Figura 38. Ejemplo: Imagen con perspectiva transformada e imagen filtrada.

La conversión de RGB a HSV se realiza porque mientras RGB define los colores como una mezcla de cantidades de rojo, verde y azul; HSV los define como una combinación de tono, saturación y brillo (Figura 39). La definición de filtros de color en la base RGB es mucho más complicada ya que es necesario dictar de cuánto de cada color primario se compone el color que se busca. Además, en nuestro caso existen cambios de brillo y saturación constantes. Definir un rango de colores junto con un rango de saturación y un rango de brillo en RGB es muy complicado, por ejemplo, sería necesario definir una diagonal en la cara que se ve en la Figura 39 para coger las variaciones de brillo de un rojo, mientras que en HSV, consistiría en coger el plano completo del ángulo deseado dentro del cilindro definido por este sistema de color.

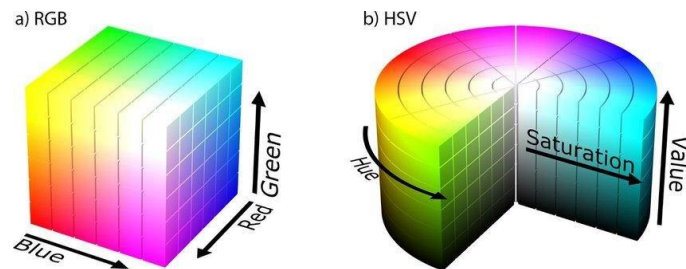


Figura 39. Bases de color RGB y HSV.

3. Una vez realizado el filtrado de color es necesario extraer la información de los caminos detectados para convertirlos en una serie de parámetros manejables para las posteriores capas del algoritmo. En nuestro caso, se ha decidido **generar una matriz de puntos** de tamaño 3×3 , donde cada punto referencia a una parte de la trayectoria que podría seguir el vehículo. Para ello se realizan los siguientes pasos (Figura 40):
 - a. Se obtienen 3 recortes horizontales de la imagen de 100px de alto (equivalente a 20 cm).
 - b. Se realiza un pequeño filtrado a los 3 recortes realizados que une los píxeles blancos cercanos para eliminar ruido.
 - c. Se obtienen los contornos de la imagen, es decir, los conjuntos de píxeles blancos pegados entre sí.
 - d. Se obtiene el punto medio de dichos contornos, obteniendo unas coordenadas en píxeles convertibles a medidas de distancia.

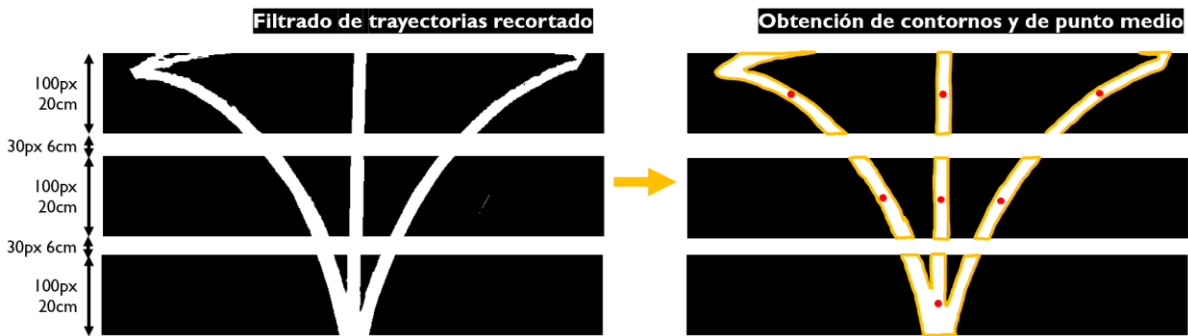


Figura 40. Ejemplo: Recorte de imagen filtrada y obtención de contornos y puntos.

Respecto a este apartado, es necesario tener en cuenta una serie de consideraciones:

- La altura de los cortes es de 20cm para minimizar las posibilidades de no ver la ruta. Debido a los cambios de iluminación durante el funcionamiento de la maqueta, el filtrado está sometido a mucho ruido y, con cortes más pequeños, el sistema es menos robusto.
- Si bien la matriz de puntos tiene un tamaño de 3×3 , existe la posibilidad de no encontrar 3 contornos diferentes en cada corte, como por ejemplo en la sección inferior de la Figura 40 o cuando no hay bifurcaciones. En estos casos la matriz tendrá elementos vacíos.
- En el caso en el que el tamaño de contorno obtenido supere un umbral definido, se divide la imagen realizando 3 cortes verticales y se obtienen de nuevo los contornos, garantizando que se obtendrán puntos aproximados de la ruta a seguir.

En la Figura 41 se observa el resultado final de la capa de percepción del algoritmo de guiado autónomo del vehículo. Esta capa obtiene una matriz de puntos que definen los posibles caminos observados. Dicha matriz, es enviada a la capa de planificación para que decida qué camino seguir.

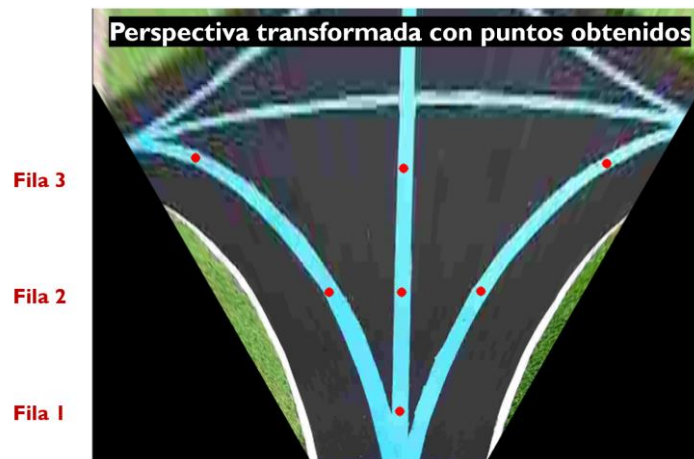


Figura 41. Ejemplo: Imagen perspectiva transformada con matriz de puntos representada.

En la Figura 42 se muestra otro ejemplo del procesado que realiza la capa de planificación para una pista con 3 carriles.

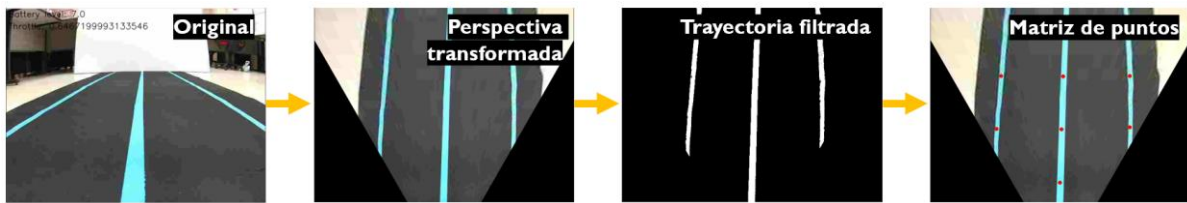


Figura 42. Ejemplo completo de capa de planificación con pista de 3 carriles

3.4.2. Planificación

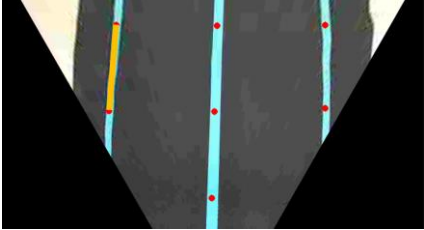
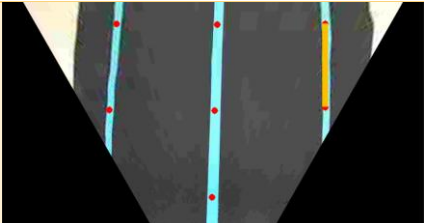
La capa de planificación se encarga de decidir qué camino se debe de tomar en cada momento (en el caso de bifurcaciones o cruces) en función de la ruta programada. Para ello, parte de la matriz de puntos obtenida de la capa de percepción y, selecciona cuáles de los puntos definirán la trayectoria que el vehículo debe de seguir.

En este trabajo, la capa de planificación debe seleccionar dos puntos de los recibidos en la matriz para ofrecer un segmento de la trayectoria que debe de seguir el vehículo en un instante determinado. Este segmento, expresado mediante las coordenadas su punto inicial y final, es el que ofrece a la capa de control.

La selección de los puntos se realiza en función del *comando de selección de ruta* activo en un instante determinado. En la siguiente tabla (Tabla 10) se muestran los comandos de comportamiento que se utilizan junto con los puntos utilizados para generar la línea que define la trayectoria (punto 1 y punto 2). Dependiendo de los puntos disponibles en la matriz de puntos, el sistema puede interpretar que no existe suficiente información para generar una trayectoria y, por lo tanto, lanzar un error de trayectoria no detectada.

Tabla 10. Selección de trayectoria en función de matriz de puntos y el comando de selección de ruta seleccionado.

Valor asignado	Explicación	Puntos seleccionados	Casos especiales	Ejemplo
1	Mantenerse a la izquierda	Puntos más a la izquierda de las filas 1 y 2		
2	Mantenerse en el centro	Puntos más centrados de las filas 1 y 2	Si no hay ningún punto en la fila 1: Se toma punto por defecto centrado en el corte Si no hay ningún punto en la fila 2: Trayectoria no encontrada	
3	Mantenerse a la derecha	Puntos más a la derecha de las filas 1 y 2		

4	Cambio de carril a la izquierda	Puntos más a la izquierda de las filas 2 y 3	Si no hay ningún punto en las filas 1 o 2: Trayectoria no encontrada	
6	Cambio de carril a la derecha	Punto más a la derecha de las filas 2 y 3		

Tal y como vemos en la tabla, los primeros 3 comandos de selección de ruta están pensados para la gestión de la trayectoria en bifurcaciones, cruces o incorporaciones. En caso de no encontrar un punto en la primera fila se asume un punto central por defecto. Este comportamiento está pensado para que, en caso de dejar el vehículo un poco alejado de la maqueta, este pueda llegar a incorporarse a la carretera.

Por otro lado, los comandos de cambio de carril están pensados única y exclusivamente para esta función. Se decidió, en este caso, utilizar los puntos de la tercera fila de la matriz debido a que, en estos casos, la primera fila de puntos no alcanza a mostrar la trayectoria de los carriles adyacentes. No se decidió utilizar un punto por defecto central en la primera fila en estos casos ya que la trayectoria calculada sería errónea, con ángulos muy pronunciados respecto del vehículo, provocando inestabilidades en el guiado del vehículo no asumibles por la capa de control.

Para poder seguir y programar rutas complejas, la capa de planificación puede seguir un vector de comandos de comportamiento, de manera que, tras cada intersección, actualice el comando de selección de ruta que sigue. En la Figura 43 se muestra un ejemplo de definición de ruta. En el ejemplo, el vehículo entra por carril central y sale por el carril de la derecha y el punto rojo indica la actualización automática del comando de selección de ruta. Es necesario tener en cuenta dos consideraciones:

- La capa lee el siguiente comando de selección de ruta únicamente tras haber pasado una intersección. Para detectarlo, la fila 2 de la matriz de puntos, que siempre incluye a más de un punto durante una intersección, bifurcación o incorporación, debe pasar a incluir un único punto. Para tener robustez al ruido, la fila 2 deberá de mantenerse con un único punto durante 15 fotogramas seguidos (parámetro configurable, explicado en la sección 3.4.4).
- En el caso de las incorporaciones, es necesario incluir un comando para ellas, siendo mantenerse a la derecha en caso de incorporaciones a la izquierda o mantenerse a la izquierda en caso de incorporaciones a la derecha. De esta forma se logra una incorporación suave. Como ejemplo se puede ver el último comando mostrado en la Figura 43.

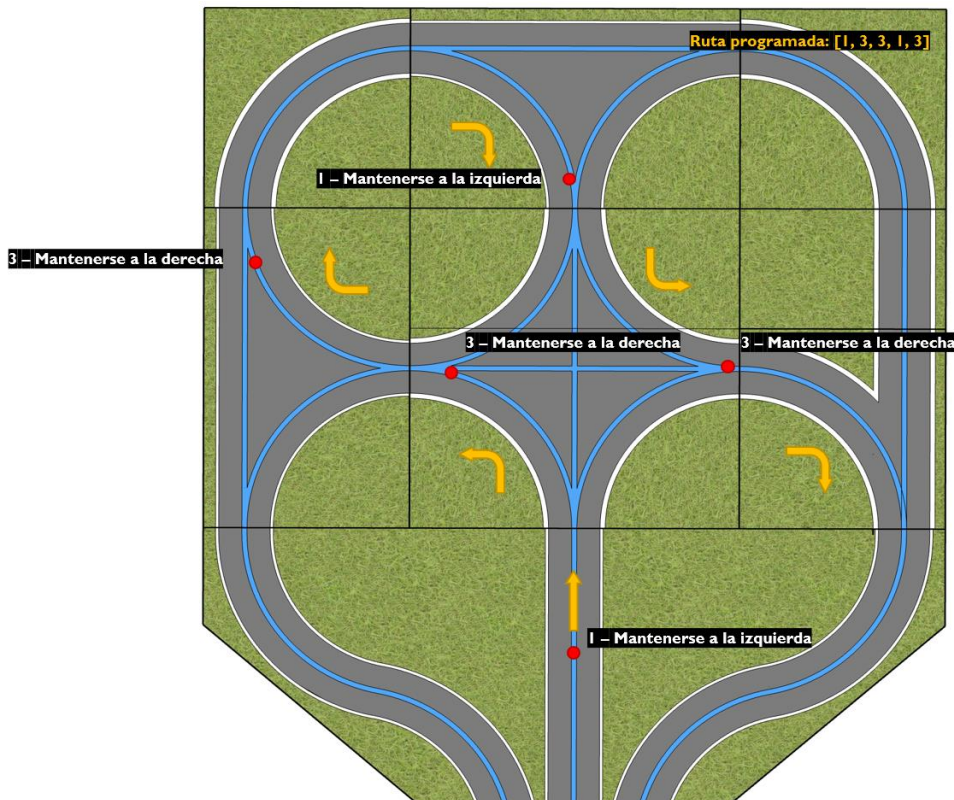


Figura 43. Ejemplo: Definición de ruta y elección de comando de selección de ruta por capa de planificación.

Por otro lado, esta capa también acepta el control del comando del comportamiento utilizado mediante otro programa externo, a través de un argumento de la función correspondiente. En la sección 3.4.4 se explica cómo funciona.

3.4.3. Control

La capa de control es la encargada de ajustar los parámetros del vehículo para que este siga el segmento de trayectoria ofrecido por la capa de planificación de forma correcta, sin oscilaciones ni salidas.

Para implementar esta capa, nos hemos basado en el algoritmo de control de dirección implementado en el vehículo Stanley [49] por la Universidad de Stanford. Este vehículo fue el primero en ganar el *DARPA Grand Challenge*, carrera de vehículos autónomos de Estados Unidos en la cual, los vehículos deben realizar un recorrido sin ninguna intervención humana en el menor tiempo posible. Incluía una multitud de sistemas y algoritmos para la percepción del entorno, la planificación de la ruta y el control del vehículo. Hemos reutilizado el algoritmo de control lateral que implementaron.

Este algoritmo acepta como entradas la trayectoria del vehículo y su velocidad (u), dando como salida el ángulo de giro que debe tomar la dirección del vehículo en un instante determinado (δ). En la Figura 44 se muestran todas las variables utilizadas en el algoritmo. A partir de la trayectoria del vehículo el sistema obtiene dos parámetros fundamentales:

- Error transversal de trayectoria (*cross-track error*): (x) mide la distancia lateral al centro de las ruedas delanteras del vehículo desde el punto más cercano de la trayectoria.
- Ángulo de la ruta respecto del vehículo: (ψ) define la orientación del segmento de trayectoria más cercano, medida de forma relativa a la propia orientación del vehículo.

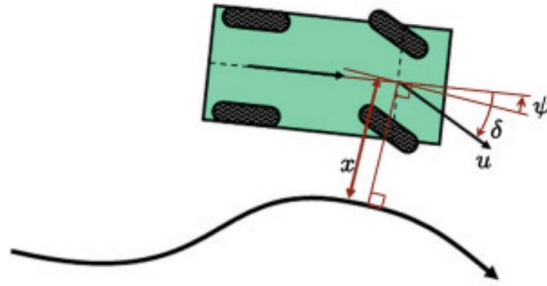


Figura 44. Variables utilizadas en el algoritmo de control de dirección de Stanley.

La expresión completa del algoritmo es la mostrada en la siguiente ecuación:

$$\delta(t) = \psi(t) + \arctan \frac{kx(t)}{u(t)}$$

Observamos como si el error transversal de trayectoria es nulo ($x(t) = 0$), es decir, si el vehículo está perfectamente centrado en la ruta, el algoritmo devuelve el ángulo de la ruta respecto del vehículo directamente ($\delta = \psi$), de manera que, si el sistema está correctamente calibrado, se seguiría la ruta sin desvíos. Si el error transversal de trayectoria no es nulo, el algoritmo intentará centrar el vehículo, añadiendo un sumando al ángulo de salida ($\arctan \frac{kx(t)}{u(t)}$) que tiene en cuenta la velocidad del vehículo y el error. Este sumando permite que el error converja a 0 en un tiempo determinado, independientemente de la velocidad que lleve el vehículo, de manera que se harán correcciones suaves cuando el vehículo vaya a mucha velocidad y correcciones bruscas a poca velocidad. La constante k es un parámetro de calibración que permite definir dicho tiempo de convergencia.

A la hora de aplicar este algoritmo en nuestro sistema nos encontramos con un problema. Si bien la capa de planificación ofrece un segmento de trayectoria del cual se pueden obtener los parámetros x (error transversal de trayectoria) y ψ (ángulo de la ruta respecto del vehículo), dicho segmento no se encuentra en el eje de las ruedas, sino 20cm más adelante (donde la cámara comienza a ver). Si introducimos dichos parámetros en el algoritmo, el vehículo, si bien en rectas se mantiene estable, en curvas se desestabiliza, girando antes de tiempo. En la Figura 45 se muestra de forma intuitiva el motivo de este comportamiento en curvas.

Este problema se evitaría si la capa de planificación fuera capaz de ir reconstruyendo la ruta de forma que sepa cómo es el segmento de trayectoria que se encuentra justo encima del vehículo. Sin embargo, para realizar esto son necesarios datos de odometría y posición del vehículo, información no disponible en este vehículo. Ya que esta solución no es viable, se plantearon dos soluciones alternativas mostradas de forma intuitiva en la Figura 46:

1. Extender el vector de trayectoria hasta el eje de las ruedas. Mientras el ángulo ψ tras no se modifica, sí que varía error transversal de trayectoria x . La ruta equivalente que seguiría el vehículo sería más verosímil a la ruta real. Trasladamos el segmento de trayectoria hacia la derecha o la izquierda para que así, partiendo del eje de las ruedas del vehículo, su extensión coincida con el segmento obtenido por la capa de planificación. De esta forma, teóricamente el vehículo debería de girar de forma coherente para seguir correctamente la trayectoria vista por la cámara. Este método se probó de forma experimental y, si bien mejoraba el comportamiento del vehículo en curvas, evitando que girara con antelación, este seguía siendo bastante inestable.
2. Extender el vector multiplicándolo por un factor k_d . El método es similar al anterior pero, en este caso, no se impone la condición estricta de que este tenga que alargarse hasta el eje

de las ruedas siempre. Empíricamente se ha comprobado que este método es muy estable y es el que se utiliza.

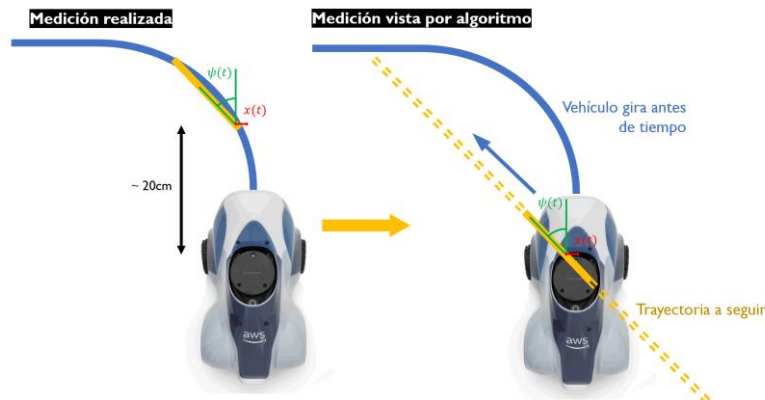


Figura 45. Problema por medición de segmento de ruta adelantado.

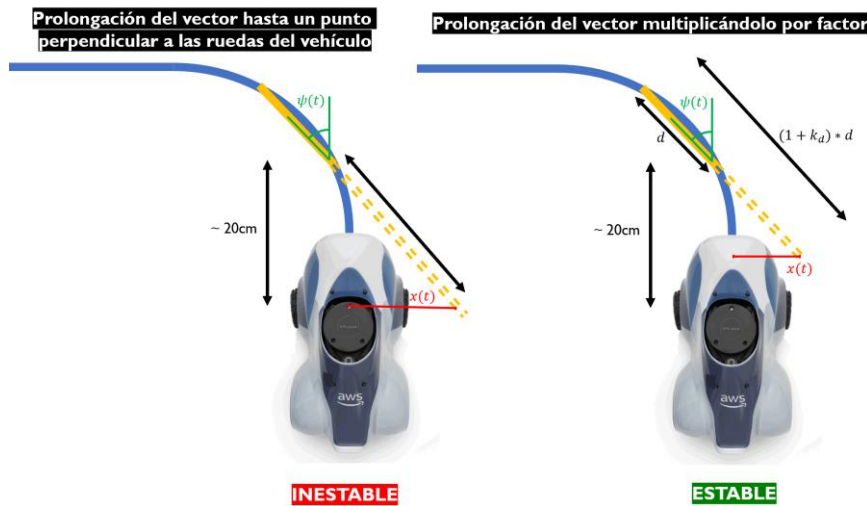


Figura 46. Métodos analizados para dar estabilidad al algoritmo.

Para calcular los parámetros de entrada del algoritmo Stanley partiendo de las coordenadas de los dos puntos que definen la trayectoria realiza los siguientes cálculos:

$$\psi = \arctan\left(\frac{x_1 - x_2}{y_1 - y_2}\right) [rad]$$

$$x = -\frac{x_2 - 320 + (x_1 - x_2) * k_d}{n \frac{[px]}{[m]}} [m]$$

Donde x_1 e y_1 son las coordenadas del primer punto, x_2 e y_2 son las coordenadas del segundo punto, k_d es el factor utilizado para prolongar el vector de trayectoria y n son los píxeles de la imagen equivalentes a un metro (aproximadamente 500 píxeles en nuestro caso).

3.4.4. Clase `artemis_autonomous_car`.

La clase `artemis_autonomous_car` incorpora todos los atributos de configuración y métodos necesarios para ejecutar el algoritmo de guiado autónomo explicado en esta sección. En la Figura 47 se observa un esquema de funcionamiento del algoritmo de conducción autónoma, separada por funciones. Los dos métodos principales de la clase son `proceso_fotograma` y `proceso_lidar`.

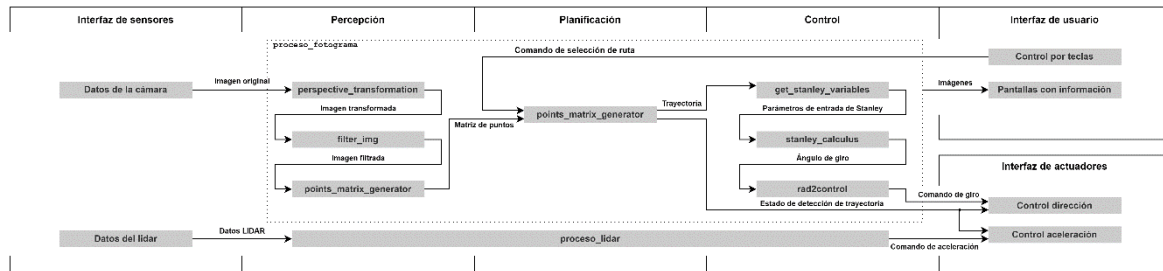


Figura 47. Esquema de funcionamiento de las funciones principales del algoritmo de conducción autónoma.

El método principal `proceso_fotograma` se encarga de implementar todas las tareas de percepción, planificación y control lateral del vehículo. Para ello, toma como entrada la imagen original en formato de imagen cv2 RGB y, a partir de esta, devuelve los parámetros de control lateral y longitudinal del vehículo, junto con una variable que indica si se ha detectado o no trayectoria a seguir. Internamente, llama a múltiples funciones las cuales se encargan de realizar las diferentes tareas de percepción, planificación y control explicadas en las subsecciones anteriores.

Por otro lado, el método principal `proceso_lidar` implementa un procesamiento básico de los datos del lidar a partir de los cuales devuelve el parámetro de aceleración del vehículo. Utilizando únicamente un rango de puntos del LiDAR (15° de la parte delantera del vehículo), obtiene el punto más cercano y, a través de una función lineal, devuelve el parámetro de aceleración. Esta función es calibrable con dos parámetros: `offset` y `sensibilidad`. Por otro lado, si la distancia devuelta por el rango de puntos del LiDAR es menor que un mínimo, el parámetro de aceleración que devuelve es nulo. Es importante recalcar que la salida de velocidad que ofrece este proceso se guarda en un atributo de la clase, y se devuelve al vehículo cuando se llama a la función `proceso_fotograma`.

La clase incorpora una serie de atributos que permiten calibrar y configurar el algoritmo del vehículo:

Tabla II. Atributos de la clase `proceso_fotograma`.

Imagen de entrada	<code>img_width</code>	Anchura de la imagen en px
	<code>img_height</code>	Altura de la imagen en px
Definición de matriz de transformación	<code>source</code>	Rango de píxeles de la imagen que se utilizan para la transformación.
	<code>dest</code>	Rango de píxeles de salida de la transformación
Control longitudinal	<code>stop</code>	l = orden de detención del vehículo
	<code>throttle_s</code>	Sensibilidad del acelerador en función de los datos del LiDAR
	<code>throttle_o</code>	Offset del acelerador en función de los datos del LiDAR
	<code>max_throttle</code>	Máximo valor de aceleración
	<code>min_longitude_obstacle</code>	Mínima distancia a un obstáculo a partir de la cual el vehículo no se detiene
Control lateral	<code>k_cross_track_error</code>	Constante k_d
	<code>k_stanley</code>	Constante del algoritmo de control lateral de Stanley
	<code>mínimum_contour_area</code>	Mínimo tamaño de contorno en píxeles para obtener un punto a partir de él

	<code>contour_junction_param</code>	Parámetro de unión de contornos cercanos utilizado en el filtrado de imagen filtrada para evitar ruido
	<code>pixels_in_meter</code>	Píxeles por metro equivalente en la imagen con perspectiva transformada
	<code>max_larger_width_contour</code>	Máximo tamaño de contorno, si se supera se dividen la imagen
	<code>contour_cut_x_position_1</code>	Coordenada X de la división 1 de la imagen en caso de superar el tamaño de contorno máximo
	<code>contour_cut_x_position2</code>	Coordenada X de la división 2 de la imagen en caso de superar el tamaño de contorno máximo
	<code>path</code>	Vector de comandos de selección de ruta.
	<code>steering_calibration_param</code>	Parámetro de calibración de la dirección (no utilizado)
Definición de color	<code>lower_color</code>	Definición de color HSV mínimo para filtrado
	<code>upper_color</code>	Definición de color HSV máximo para filtrado

El código, disponible en [47], incluye todas las funciones y variables comentadas con explicaciones.

Respecto a las tareas de planificación, la clase ofrece dos posibilidades para seguir rutas, definible a través del argumento de dicha función `real_time_control`:

- Si este argumento se mantiene a 0, la capa de planificación seguirá la ruta definida por el vector de comandos de selección de ruta `path`.
- Si este argumento toma otro valor, la capa de planificación implementará el comando de selección de ruta equivalente al valor introducido como argumento. De esta forma, se permite que un *software* externo sea capaz de controlar las rutas que sigue el vehículo en tiempo real. (Por ejemplo, usuario en servidor controle la ruta que sigue el vehículo a través de las teclas, decidiendo cuando tiene que tomar un desvío u otro en el momento).

4. Red móvil 4G mediante SDR

En este capítulo se explica la implementación de una red 4G LTE con SDR. Tal y cómo se muestra en el estado del arte (sección 2.2.1 y Figura 13), existen tres partes bien diferenciadas a la hora de implementar una red móvil basada en SDR. En este Trabajo Fin de Máster se han implementado dichas tres partes con diferentes *softwares* y dispositivos. Mientras en el estado del arte se muestra una introducción a las alternativas más relevantes encontradas para la implementación de la red, en este capítulo se explica la instalación y configuración de varias de estas alternativas, las cuales han sido probadas y utilizadas para el estudio de los casos de uso explicados en el capítulo 5.

En la sección 4.1 se explican los métodos utilizados para ofrecer acceso a la red a los dispositivos de usuario, es decir, los vehículos en nuestro caso. Para la transmisión y recepción de la señal mediante SDR se han probado los dispositivos BladeRFxA9 y USRP NI-B2901, cuya instalación y configuración se explica en la sección 4.2. Para la implementación de los servicios de la RAN de la red móvil se ha utilizado el *software* srsENB (perteneciente a la familia de *software* srsRAN), instalación y configuración explicada en la sección 4.3. En cuanto a la implementación de los servicios del núcleo de la red móvil se ha utilizado srsEPC y Open5GS, cuya instalación y configuración se explica en la sección 4.4.

En la página GitHub del Grupo de Comunicaciones Ópticas de la Universidad de Valladolid se puede encontrar un repositorio llamado Artemis [47] que incluye los ficheros de configuración utilizados en el proyecto, junto con scripts de instalación para poner en marcha el sistema de forma rápida y sencilla.

4.1. Configuración del dispositivo de usuario

Para ofrecer a los usuarios, ya sean teléfonos móviles o módems 4G, acceso a la red, es necesario disponer de unas SIMs propietarias. Una SIM incluye una serie de claves utilizadas para establecer la conexión con el núcleo de la red. Dichas claves únicamente las conocen el núcleo y la propia SIM, de manera de que, si no conocemos las claves de la SIM que vamos a utilizar para conectar un dispositivo a nuestra red, nunca podríamos configurar dichas claves en el núcleo y, por lo tanto, no lograríamos establecer la conexión. Es por esto por lo que es necesaria la compra de unas SIMs propietarias sobre las cuales se conozcan sus claves. En nuestro caso hemos utilizado las SIMs programables sysmolSIM-SJA2 de la empresa Sysmocom [10], sobre las cuales sí que conocemos todos sus parámetros.



Figura 48. Tarjetas SIM sysmolSIM-SJA2 junto con dispositivo USB que permite reprogramarlas.

Cada SIM incluye un identificador denominado IMSI (*International Mobile Subscriber Identity*) el cual permite identificar de forma inequívoca a cualquier usuario. Los 6 primeros dígitos de este identificador se corresponden con el MCC (*Mobile Country Code*) y el MNC (*Mobile Network Code*). La

combinación de ambos parámetros permite definir el operador que ofrece el servicio. El primer parámetro define el país del operador telefónico y segundo parámetro define el operador como tal. La ITU define y asigna dichos códigos a cada operador, en [50] se puede encontrar el estado actual de dichas asignaciones a nivel internacional.

Las SIMs de Sysmocom son programables y, por lo tanto, permiten modificar su IMSI para asignarle el identificador y el MCC y MNC que nosotros queramos. Sin embargo, el MCC-MNC con el que vienen por defecto es el 901-70, combinación únicamente utilizada por una pequeña empresa de telecomunicaciones de Estonia. Debido a que en España esta empresa no ofrece servicio⁶ y que la reprogramación del IMSI es arriesgada (en caso de fallo las tarjetas pueden quedar inutilizadas), se decidió mantener el IMSI original de las tarjetas.

A parte de disponer de una SIM propietaria para que el dispositivo de usuario se pueda conectar correctamente a la red es necesario configurar un Nombre de Punto de Acceso (APN). Cualquier red de telefonía tiene un APN asignado y, sin este, el usuario no logra conexión a internet. En nuestro caso, hemos decidido utilizar el nombre artemis como APN. Este parámetro se configura en el propio dispositivo. En el caso de un teléfono móvil se puede configurar en las opciones de red móvil. En el caso del módem 4G existe una interfaz web que permite configurarlo también.

A parte del nombre del APN, las redes 4G convencionales suelen requerir un nombre de usuario y contraseña. Sin embargo, estos parámetros no son obligatorios por lo que, en nuestro caso, hemos prescindido de ellos.

4.2. Configuración del SDR

El dispositivo SDR es el utilizado para tara transmitir y recibir las señales. Este dispositivo se encarga de hacer de frontera entre el ordenador que implementa el servicio RAN (Intel NUC en este Trabajo Fin de Máster) de la red y el tratamiento analógico de la señal. Tal y como se ha comentado anteriormente, se han probado dos dispositivos, BladeRFxA9 y USRP NI-B2901.

4.2.1. BladeRFxA9

La BladeRF 2.0 micro xA9 de Nuand [9] (Figura 49) incorpora dos canales RX y TX MIMO, un rango de frecuencias de 47MHz a 6 GHz y 30 MHz de ancho de banda.



Figura 49. BladeRFxA9 utilizada en el proyecto.

⁶ En caso de utilizar una combinación MCC-MNC de una operadora que ofrece cobertura en la ubicación de la SIM, el dispositivo de usuario intentaría conectarse con sus estaciones base.

Para utilizar el dispositivo con un ordenador con Ubuntu 20.04 LTS, conectado a través de USB, es necesario instalar los drivers correspondientes. Para ello, es necesario instalar una serie de dependencias; a continuación, descargar, compilar e instalar el *software* bladeRF; y, por último, crear los enlaces necesarios a las librerías recién instaladas. Esto se puede hacer a través de los siguientes comandos:

```
sudo apt install libbladerf2 libbladerf-dev libusb-1.0-0 libusb-1.0-0-dev
git clone https://github.com/Nuand/bladeRF
cd bladeRF/host
mkdir build
cd build
cmake ..
make -j4
sudo make install
sudo ldconfig
```

Después, debemos descargar el *firmware* de nuestra tarjeta bladeRF con el comando:

```
sudo bladeRF-install-firmware
```

Podemos comprobar si el controlador instalado en el ordenador detecta el dispositivo conectado correctamente con el siguiente comando comando:

```
bladeRF-cli -p.
```

Existe más información acerca de la instalación y uso del *software* bladeRF en su github [51] y la wiki bladeRF [52].

4.2.2. USRP NI-B2901

La USRP NI-2901 de National Instruments [8] (Figura 50) incorpora dos canales RX y TX MIMO, un rango de frecuencias de 70 MHz a 6 GHz y 56 MHz de ancho de banda para utilizarlos como *hardware* SDR.



Figura 50. USRP NI-B2901 utilizada en el proyecto.

Para utilizar el NI USRP-2901 en Ubuntu 20.04 LTS es necesario instalar los drivers UHD (USRP *Hardware Driver*) que ofrece la empresa ETTUS. Para realizar una correcta instalación de estos drivers, es necesario añadir el repositorio correspondiente a la lista de Ubuntu e instalar una serie de dependencias y el driver UHD. Todo esto se puede realizar introduciendo los siguientes comandos:

```
sudo add-apt-repository ppa:ettusresearch/uhd
sudo apt-get update
sudo apt-get install -y libuhd-dev libuhd3.15.0 uhd-host
sudo uhd_images_downloader
```

Una vez seguidos estos pasos, se puede comprobar si el *software* instalado reconoce correctamente la USRP tecleando el siguiente comando:

```
sudo uhd_usrp_probe
```

Existe más información acerca de la instalación y uso de los drivers UHD en el manual de usuario que ofrece ETTUS [53].

4.3. RAN de la red con srsRAN

Tal y como se explicó en el estado del arte (subsección 2.2.1.1), el conjunto de *software* que ofrece srsRAN incluye srsENB, aplicación que incorpora la pila de protocolos completa de 4G de un eNodeB y la de 5GNSA de un gNodeB. Esta aplicación incluye la mayoría de los servicios de un eNodeB y está programada para ser utilizada con un dispositivo SDR, como bladeRF de Nuand o una USRP de ETTUS.

4.3.1. Instalación

Para instalar el *software* en Ubuntu 20.04 LTS, es necesario seguir los pasos indicados en el manual de instalación de srsRAN [54]. Los pasos consisten en la instalación de una serie de dependencias, clonar el repositorio de srsRAN, compilar el programa y crear los enlaces necesarios a las nuevas librerías instaladas. A parte de las dependencias indicadas en el manual de instalación, se recomienda instalar otra serie de dependencias para garantizar que el *software* funcione correctamente y sea compatible tanto con la BladeRF como con la USRP. A continuación, se explican los pasos detalladamente que hay que realizar, con las comprobaciones necesarias para garantizar que todo funciona correctamente.

1. Instalar todas las dependencias necesarias:

```
sudo apt-get install libboost-system-dev libboost-test-dev libboost-thread-dev
libqwt-dev libqt4-dev
sudo apt-get install boost libboost-all-dev libpolarssl-dev
sudo apt-get install -y build-essential cmake libfftw3-dev libmbedtls-dev
libboost-program-options-dev libconfig++-dev libsctp-dev
```

2. Clonar el repositorio srsRAN de github:

```
git clone https://github.com/srsRAN/srsRAN.git
cd srsRAN
```

3. Abrir el fichero CMakeLists.txt y comprobar que están habilitadas las opciones SRSENB, UHD y BLADERF:

```
option(ENABLE_SRSENB      "Build srsENB application"      ON)
option(ENABLE_UHD        "Enable UHD"                    ON)
option(ENABLE_BLADERF    "Enable BladeRF"                 ON)
```

4. Crear carpeta de compilación y ejecutar cmake:

```
mkdir build
cd build
cmake ../
```


5. Se crearán los ficheros de compilación necesarios en la carpeta build. Abrir el fichero CMakeCache.txt dentro de dicha carpeta y comprobar que se ha detectado correctamente la ubicación de las librerías de los controladores UHD y bladeRF. Este paso es muy importante ya que si no detecta correctamente dichos controladores, tras compilar el programa este no funcionará con el dispositivo SDR.

```
//Enable UHD
ENABLE_UHD:BOOL=ON
//Path to a file.
UHD_INCLUDE_DIRS:PATH=/usr/include
//Path to a library.
UHD_LIBRARIES:FILEPATH=/usr/lib/x86_64-linux-gnu/libuhd.so
//Enable BladeRF
ENABLE_BLADERF:BOOL=ON
//Path to a file.
BLADERF_INCLUDE_DIRS:PATH=/usr/local/include
//Path to a library.
BLADERF_LIBRARIES:FILEPATH=/usr/local/lib/libbladeRF.so
```

6. Compilar e instalar el programa desde la carpeta build:

```
make
sudo make install
```

7. Copiar los ficheros de configuración por defecto y actualizar enlaces de librerías:

```
sudo srsran_install_configs.sh user
sudo ldconfig
```

Una vez seguidos estos pasos tendremos instalado el *software* con los parámetros de configuración por defecto. En la siguiente subsección se explican los diferentes parámetros de configuración que se pueden modificar.

4.3.2. Configuración

El *software* srsENB incluye un fichero de configuración principal denominado `enb.conf`, y 3 ficheros de configuración secundarios: `sib.conf` para configurar los SIB (*System Information Block* [55]), `rr.conf` para configurar los recursos radio, y `rb.conf`. Todos estos ficheros se encuentran en la siguiente ruta:

```
/root/.config/srsran
```

El fichero de configuración `enb.conf` incluye la configuración principal del programa. Está escrito en formato INI, un tipo de fichero de configuración en el que cada parámetro se define como un conjunto sección-propiedad-valor. Una sección del fichero de configuración incluye una serie de propiedades, cada una con un valor configurado. Cada parámetro de configuración viene comentado en el fichero por defecto, de forma que se puede comprender fácilmente su significado. A continuación, se comentan los parámetros principales que son necesarios modificar para montar una red a nuestra medida.

- `enb.enb_id`: **Identificador** en hexadecimal del eNodeB. Se puede elegir cualquiera, únicamente relevante cuando se realiza *handover*, en cuyo caso el identificador de cada eNodeB debe de ser diferente.
- `enb.mcc` y `enb.mnc`: Permiten configurar el **MCC** (*Mobile Country Code*) y **MNC** (*Mobile Network Code*) del eNodeB. La combinación de ambos parámetros permite definir el operador que ofrece el servicio. Tal y como se ha explicado en la sección 4.1, el MCC y MNC que utilizamos en nuestro caso es 901-70 para que se corresponda con el IMSI de las SIM.
- `enb.mme_addr`: **IP** de la interfaz S1-C en el servicio MME del núcleo (Figura 51).

- `enb.gtp_bind_addr`: **IP** de la interfaz de datos de usuario S1-U o interfaz GTP en el eNodeB (Figura 51).
- `enb.s1c_bind_addr`: **IP** de la interfaz de control S1-U o interfaz S1-MME en el eNodeB (Figura 51).
- `enb.n_prb`: Número de Bloques de Recursos Físicos (PRB, *Physical Resource Blocks*). Proporcional al **ancho de banda utilizado**, siendo un PRB equivalente a 0,2MHz. Los valores permitidos son 6=1.4MHz, 15=3MHz, 25=5MHz, 50=10MHz, 75=15MHz y 100=20MHz. Más información acerca de este parámetro y su significado en [56].
- `enb.tm`: **Modo de transmisión (SISO, MIMO)** del eNodeB con cuatro posibles valores: una única antena, transmitir diversidad, CDD (Cyclic Delay Diversity) o multiplexado espacial de lazo cerrado.
- `enb.nof_ports`: Número de puertos de transmisión (1 por defecto y 2 para los modos de transmisión 2, 3 y 4).
- `rf.dl_earfcn`: Código EARFCN (E-UTRA *Absolute Radio Frequency Channel Number*) para el enlace de descarga. Especifica la **banda en la que se va a transmitir**. El código define tanto la frecuencia del enlace de descarga como el de subida.
- `rf.tx_gain`: **Ganancia de la transmisión**. Este parámetro define los dBm a los que transmitirá el dispositivo SDR.
- `rf.rx_gain`: **Ganancia en la recepción**. Este parámetro define cuánto se amplifica la señal recibida antes de ser procesada. Si se comenta, se habilita AGC (*Automatic Gain Control*), de manera que se ajusta automáticamente en tiempo real en función de la potencia recibida.

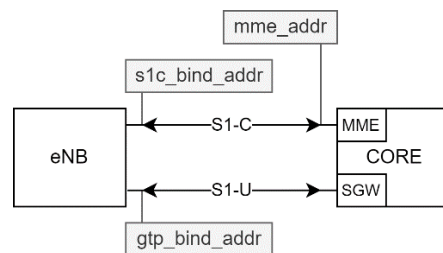


Figura 51. Interfaces de red configurables en srsENB.

El fichero `rr.conf` se utiliza para configurar los recursos radio del sistema. Permite configurar múltiples parámetros de bajo nivel correspondientes con el estándar LTE. Es decir, los parámetros configurables en este fichero son parámetros que se configuran en las estaciones eNodeB utilizadas por los operadores. Por otro lado, este fichero también permite configurar varios sectores de celdas, *handover* y habilitar 5GNSA (deshabilitado por defecto). En este Trabajo Final de Máster el contenido de este fichero se ha mantenido por defecto salvo para la implementación de algunos de los casos de uso explicados en el capítulo 5. Sus modificaciones se explican en las secciones de los casos de uso que lo necesitan.

Por otro lado, los ficheros `rb.conf` y `sib.conf` se han mantenido por defecto en este Trabajo Final de Máster y, por lo tanto, no se analizan.

4.3.3. Recomendaciones para mejorar el rendimiento del *software srsenb*.

Este *software* consume muchos recursos. A parte de los servicios correspondientes al eNodeB, es preciso recordar que estamos trabajando con dispositivos SDR y por lo tanto, prácticamente la totalidad del procesamiento de la señal es digital y llevado a cabo en el ordenador. Es el *software srsenb* el que se encarga de producir un flujo de muestras de señal para la transmisión, dirigido hacia el SDR, y de procesar el flujo de muestras proveniente de este. La tasa de muestreo (configurable manualmente en el fichero `enb.conf`, aunque por defecto la calcula automáticamente el *software* al arranque en función del parámetro que define el ancho de banda) impone una velocidad de generación

y procesado de muestras que el ordenador debe poder soportar. En los casos en los que el ordenador se retrase, se mostrarán por consola las letras U (Underflow = las muestras enviadas por el ordenador al SDR para la transmisión no alcanzan la velocidad de muestreo) y O (Overflow = el ordenador no consume las muestras provenientes del SDR lo suficientemente rápido). Cuando sucede esto, la red experimenta caídas y comportamientos no deseables.

Aún disponiendo de un ordenador con gran capacidad de cómputo, estas situaciones se pueden dar debido a la gestión de procesos del sistema operativo. Este realiza un multiplexado en tiempo para organizar la ejecución de varios procesos en una misma CPU. La decisión de qué proceso se ejecuta antes que el resto se toma en función de unos valores de prioridades. Para garantizar que el *software* *srsenb* se ejecute siempre antes que el resto se debe añadir la prioridad más alta en este *software*. Sin embargo, el *kernel* de Linux no permite asignar a *softwares* de terceros el nivel máximo de prioridad.

Para lograr el máximo nivel de prioridad, es necesario instalar el módulo del *kernel* de Linux de baja latencia mediante el siguiente comando.

```
sudo apt-get install -y linux-lowlatency
```

Una vez instalado, es necesario reiniciar el ordenador y, en el menú de arranque, seleccionar la nueva versión de Linux de baja latencia. Después, podemos ejecutar el *software* *srsenb*. Para que se ejecute con la máxima prioridad (en tiempo real) en el sistema operativo, será necesario escribir el siguiente comando.

```
sudo chrt -r -p 99 $(pidof srsenb)
```

4.4. Núcleo de la red

Para la implementación del núcleo de la red, en este Trabajo Final de Máster se han utilizado dos *softwares*, *srsEPC* y *Open5GS*. En el estado del arte (subsección 2.2.1), se da una introducción a ambos *softwares*. A continuación se explican los pasos para su instalación y correcta configuración.

4.4.1. *srsEPC*

El *software* *srsEPC*, parte de la familia del conjunto *srsRAN*, consiste en una implementación *software* ligera de una red central LTE completa (EPC) virtualizada. Esta aplicación se ejecuta como un único binario, pero proporciona los componentes clave del EPC: servicio de abonado doméstico (HSS), entidad de gestión de la movilidad (MME), pasarela de servicios (S-GW) y pasarela de red de paquetes de datos (P-GW).

Para la instalación de dicho *software* en Ubuntu 20.04 LTS, se deben de seguir los mismos pasos explicados en la sección 4.3.1 para la instalación de *srsENB*. Dicha instalación instala todo el *software* que ofrece el conjunto *srsRAN*.

Una vez instalado, encontraremos los ficheros de configuración por defecto del EPC en siguiente directorio:

```
/root/.config/srsran
```

Estos ficheros son 2: Por un lado, el fichero de configuración principal *epc.conf*, escrito en formato INI, permite configurar los diferentes parámetros de los 4 servicios núcleo LTE que ofrece el programa (MME, HSS, S-GW y P-GW). Por otro lado, el fichero *user_db.csv* consiste en una tabla que hace de base de datos y almacena la información de todos los suscriptores (IMSI, claves, etc.).

Los parámetros más relevantes del fichero *epc.conf* son los siguientes:

- *mme.mme_code* y *mme.mme_group*: **Identificación del MME** (código y grupo), mantenido por defecto.

- `mme.mcc`: **Mobile Country Code**, explicado en las secciones anteriores. Configurado en 901 para coincidir con el IMSI de nuestras SIMs.
- `mme.mnc`: **Mobile Network Code**, explicado en las secciones anteriores. Configurado en 70 para coincidir con el IMSI de nuestras SIMs.
- `mme.tac`: **Tracking Area Code (TAC)** del MME. La red ofrecida por el operador (definida con el MCC y el MNC) se divide en diferentes áreas. Cada área incluye su propio identificador que se configura con este parámetro. Debe de coincidir con el TAC de los eNodeB, siendo este 0x7 por defecto en `srsENB` (configurable en el fichero `rr.conf`).
- `mme.mme_bind_addr`: Dirección **IP** de la interfaz S1-C o S1-MME del MME (*Core*). Ver Figura 52.
- `mme.apn`: **Nombre del punto de acceso (APN)**. Tal y como explicamos en la [sección](#)
- `mme.dns_addr`: Dirección **IP** del servicio DNS para los UEs.
- `hss.db_file`: Ruta al fichero csv que incluye la base de datos de los suscriptores.
- `spgw.gtpu_bind_addr`: Dirección **IP** de la interfaz S1-U del S-GW (*Core*). Ver Figura 52.
- `spgw.sgi_if_addr`: Dirección **IP** de la interfaz virtual SGi. Las direcciones IPs ofrecidas a los diferentes UEs formarán parte de la subred de dicha dirección IP. Ver Figura 52.
- `spgw.sgi_if_name`: **Nombre de la interfaz virtual SGi** que genera el programa.
- `spgw.max_paging_queue`: **Cola de paquetes** máxima por UE.

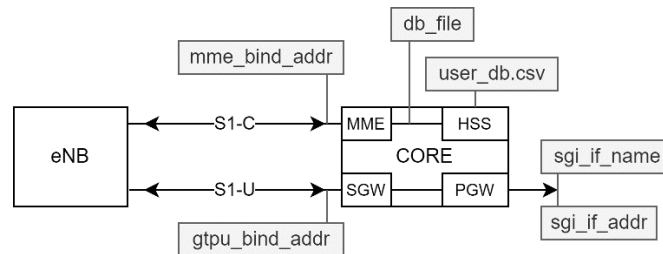


Figura 52. Interfaces configurables en srsEPC.

El fichero `user_db.csv` almacena toda la información de los suscriptores. Este fichero es el que hace de servicio HSS del núcleo. Por cada línea incluye la información de un suscriptor. El fichero está por defecto comentado e incluye dos líneas ya configuradas. Los parámetros de este se separan por comas y son los siguientes:

- Nombre: Utilizado únicamente para facilitar la lectura de logs del EPC, no utilizado realmente por el HSS.
- Algoritmo de autenticación utilizado por el UE: Puede ser XOR o MILENAGE. En nuestro caso hemos utilizado siempre MILENAGE.
- IMSI: Identificador del suscriptor.
- Clave: Clave principal de la SIM. Esta clave es una de las facilitadas por `sysmocom` para cada SIM.
- Código de Operador (OP) o Código de Operador Cifrado (OPc) configurado en la SIM del UE: Otra clave que debe de coincidir y que ofrece `sysmocom`. Este código privado se utiliza para evitar la suplantación de identidad del usuario.
- Tipo de código de operador: Define si se utiliza OP o OPC. En nuestro caso utilizamos OPC.
- AMF, SQN, QCI: Mantener por defecto en 9000, 000000000000, 9. No analizados en este Trabajo Final de Grado.
- Asignación de IP: Permite definir entre una asignación de IP dinámica o una IP estática para un usuario concreto.

Una vez ejecutado este *software*, genera una interfaz virtual llamada por defecto `srs_spgw_sgi` en el núcleo. En esta interfaz se encuentra todo el intercambio de paquetes de usuario entre los servicios

SPGW y los diferentes UEs. De esta forma, tendremos conectividad completa entre los UEs y el ordenador que ejecuta el núcleo. Sin embargo, si queremos que estos tengan conectividad hacia el exterior, es necesario configurar el ordenador que ejecuta el núcleo para que reencamine los paquetes a través de la interfaz de red. Además, debe de implementar un servicio NAT facilitar el encaminamiento de los paquetes una vez que salen del ordenador hacia internet. Así, nuestra red 4G privada se verá al exterior como una única IP, la del ordenador que implementa el núcleo. En la Figura 53 vemos un ejemplo de la asignación de IPs y de la interfaz virtual.

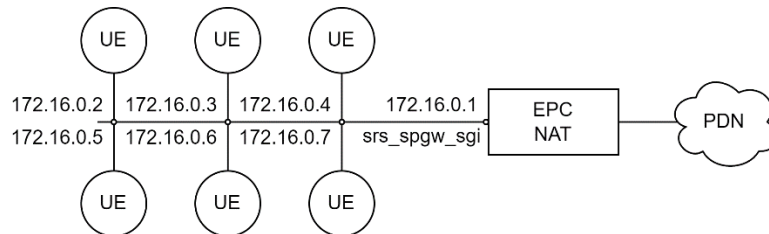


Figura 53. Ejemplo de asignación de IPs a UEs.

Estas tareas de configuración se pueden realizar manualmente. Sin embargo, el *software* srsEPC ofrece un método rápido de configuración mediante la ejecución del siguiente comando donde únicamente es necesario introducir el nombre de la interfaz de red que conecta al ordenador a internet.

```
srsepc_if_masq.sh <nombre de interfaz de red>
```

4.4.2. Open5GS

Tal y como se explicó en el estado del arte (subsección 2.2.1.3), el conjunto de *software* Open5GS [29] implementa la mayoría de los servicios del **núcleo** de la red 4G (EPC) y el núcleo de la red 5G (5G NG, parte de una red 5G SA). Implementa un *Core* 4G / 5G NSA o un *Core* 5G SA. En este Trabajo Final de Máster nos centramos en el núcleo 4G.

Este *software* incluye un tutorial de inicio rápido [30] en el que se explica detalladamente como realizar la instalación y configuración del sistema. A continuación, se van a explicar los pasos principales que hay que realizar para la instalación, y las configuraciones clave para disponer de un EPC 4G con Open5GS funcional.

4.4.2.1. Instalación

Para instalar este *software* en Ubuntu 20.04 LTS es necesario agregar el repositorio de Open5GS e instalar el *software* mediante el gestor de paquetes apt-get. Se puede realizar escribiendo los siguientes comandos.

```
sudo apt update
sudo apt install software-properties-common
sudo add-apt-repository ppa:open5gs/latest
sudo apt update
sudo apt install open5gs
```

Por otro lado, es recomendable instalar la interfaz web del sistema para poder editar la base de datos de usuarios (servicio HSS) de forma fácil e intuitiva. Se puede realizar ejecutando los siguientes comandos.

```
sudo apt update
sudo apt install curl
curl -fsSL https://deb.nodesource.com/setup_14.x | sudo -E bash -
sudo apt install nodejs
curl -fsSL https://open5gs.org/open5gs/assets/webui/install | sudo -E bash -
```

4.4.2.2. Servicios y configuración

Una vez finalizada la instalación completa, se habrán habilitado automáticamente los siguientes servicios: open5gs-mmed, open5gs-sgwcd, open5gs-smfd, open5gs-amfd, open5gs-sgwud, open5gs-upfd, open5gs-hssd, open5gs-pcrfd, open5gs-nrfd, open5gs-ausfd, open5gs-udmd, open5gs-pcfd, open5gs-nssfd, open5gs-bsfd, open5gs-udrd, open5gs-webui. Toda esta colección de servicios incluye servicios de 4G y de 5G SA. En este Trabajo Final de Máster utilizamos únicamente los relacionados con el núcleo 4G LTE (EPC) y, por lo tanto, son los que tendremos que configurar y arrancar. Estos son los siguientes:

- MME: Implementado por el servicio open5gs-mmed.
- HSS: Implementado por el servicio open5gs-hssd.
- PCRF: Implementado por el servicio open5gs-pcrfd.
- SGW: Implementado por los servicios open5gs-sgwcd (SGWC, plano de control) y open5gs-sgwud (SGWU, plano de usuario).
- PGW: Implementado por los servicios open5gs-smfd (PGWC, plano de control) y open5gs-upfd (PGWU, plano de usuario).

Tener en cuenta que cada uno de estos servicios se puede ejecutar independientemente en diferentes máquinas pudiendo comunicarse entre sí al utilizar interfaces de red. En caso de estar ejecutándose en la misma máquina se utilizaría IPs de loopback del ordenador.

Cada servicio incluye su propio fichero de configuración en el que se pueden configurar sus parámetros. Por defecto, todos los servicios están configurados para ser ejecutados directamente en un mismo ordenador, con interfaces de red configuradas con IPs de loopback. En la Figura 54 se muestra la interconexión de los servicios del núcleo de la red 4G implementado con Open5GS, así como las direcciones IPs y puertos por defecto configurados, y nombre de los parámetros de configuración de las interfaces de cada servicio.

A continuación, se explican los parámetros de configuración principales de cada servicio.

Los parámetros relacionados con el servicio **MME** se pueden modificar en el fichero de configuración /etc/open5gs/mme.yaml. Sus parámetros principales son los siguientes;

- mme.s1ap.addr/port: IP y puerto de la interfaz S1-C / S1-MME desde el MME.
- mme.gtpc.addr/port: IP y puerto de la interfaz S11 desde el MME.
- sgwc.gtpc.addr/port: IP y puerto de la interfaz S11 desde el SGW-c.
- mme.gummei.plmn_id.mcc y mme.tai.plmn_id.mcc: **Mobile Country Code**, explicado en las secciones anteriores. Configurado en 901 para coincidir con el IMSI de nuestras SIMs.
- mme.gummei.plmn_id.mnc y mme.tac.plmn_id.mnc: **Mobile Network Code**, explicado en las secciones anteriores. Configurado en 70 para coincidir con el IMSI de nuestras SIMs.
- mme.gummei.mme_code y mme.gummei.mme_gid: **Identificación del MME** (código y grupo), mantenido por defecto.
- mme.gummei.tai.tac: **Tracking Area Code** (TAC) del MME. La red ofrecida por el operador (definida con el MCC y el MNC) se divide en diferentes áreas. Cada área incluye su propio identificador que se configura con este parámetro. Debe coincidir con el TAC de los eNodesB, siendo este 0x7 por defecto en srsENB (configurable en el fichero rr.conf).
- mme.network_name.full: **Nombre de la operadora**. Nombre que se muestra en los dispositivos móviles conectados. En nuestro caso Artemis-4G.

Los parámetros relacionados con el servicio **SGWC** se pueden modificar en el fichero de configuración /etc/open5gs/sgwc.yaml. Sus parámetros principales son los siguientes;

- sgwc.gtpc.addr/port: IP y puerto de la interfaz S11 desde el SGW-c.

- `sgwc.pfcp.addr/port`: IP y puerto de la interfaz Sxa desde el SGW-c.
- `sgwu.pfcp.addr/port`: IP y puerto de la interfaz Sxa desde el SGW-u.

Los parámetros relacionados con el servicio **SGWU** se pueden modificar en el fichero de configuración `/etc/open5gs/sgwu.yaml`. Sus parámetros principales son los siguientes;

- `sgwu.pfcp.addr/port`: IP y puerto de la interfaz Sxa desde el SGW-u.
- `sgwu.gtpu.addr/port`: IP y puerto de la interfaz S1-U y S5u desde el SGW-u.

Los parámetros relacionados con el servicio **PGWC** se pueden modificar en el fichero de configuración `/etc/open5gs/smf.yaml`. Sus parámetros principales son los siguientes;

- `smf.frDi.addr/port`: IP y puerto de la interfaz Gx desde el PGW-c.
- `smf.pfcp.addr/port`: IP y puerto de la interfaz Sxb desde el PGW-c.
- `smf.gtpc.addr/port`: IP y puerto de la interfaz S5c desde el PGW-c.
- `smf.gtpu.addr/port`: IP y puerto de la interfaz Sxu desde el PGW-c.
- `upf.pfcp.addr/port`: IP y puerto de la interfaz Sxu desde el PGW-u.
- `smf.subnet.addr`: **Subred** en la que se asignan las diferentes IPs de los UEs.
- `smf.dns`: Dirección **IP** del servicio DNS para los UEs.
- `smf.mtu`: **MTU** (*Maximum Transfer Unit*) utilizado en la red 4G.

Los parámetros relacionados con el servicio **PGWU** se pueden modificar en el fichero de configuración `/etc/open5gs/upf.yaml`. Sus parámetros principales son los siguientes;

- `upf.pfcp.addr/port`: IP y puerto de la interfaz Sxb desde el PGW-u.
- `upf.gtpu.addr/port`: IP y puerto de la interfaz Sxu y S5u desde el PGW-u.
- `upf.subnet.addr`: **Subred** en la que se asignan las diferentes IPs de los UEs.

Los parámetros relacionados con el servicio **HSS** y **PCRF** se pueden modificar en el fichero de configuración `/etc/open5gs/hss.yaml` y `/etc/open5gs/pcrf.yaml`. Sus parámetros principales son los siguientes:

- `db_uri`: Dirección URI desde la que se puede acceder a la base de datos de suscriptores, implementada en MongoDB por defecto en open5Gs.
- `hss.frDi.addr/port`: IP y puerto de la interfaz S6a desde el HSS.
- `pcrf.drDi.addr/port`: IP y puerto de la interfaz Gx desde el PCRF.

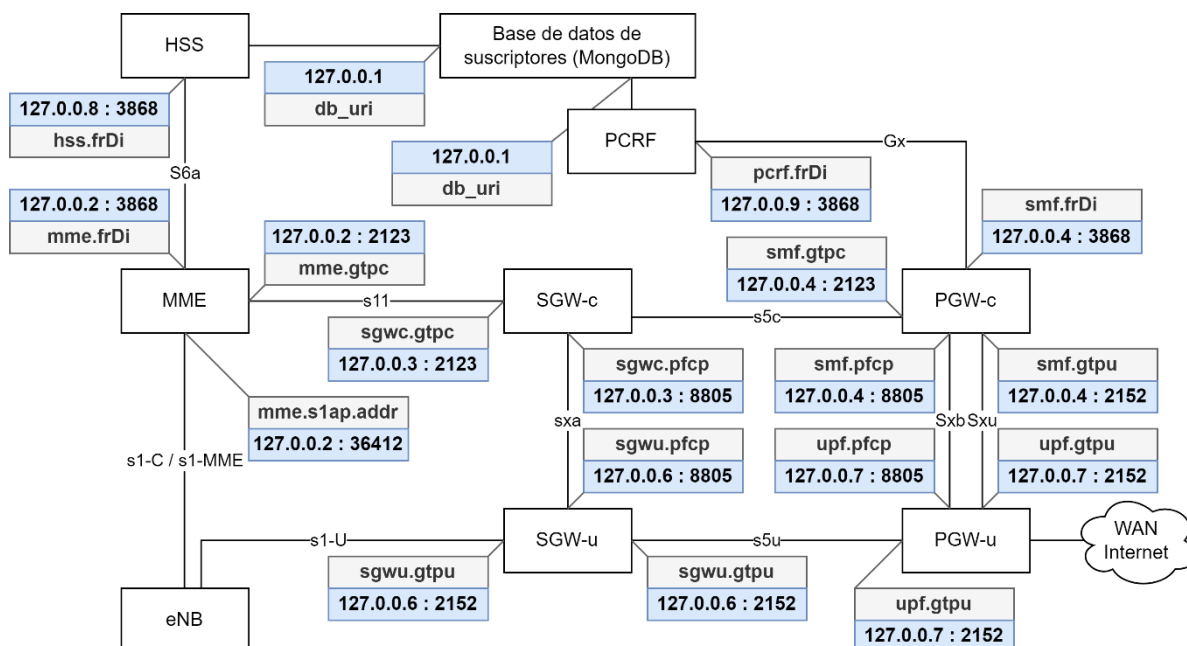


Figura 54. Servicios, direcciones IPs y puertos por defecto configurados, y nombre de los parámetros de configuración de las interfaces del núcleo de la red 4G implementada con Open5Gs.

Existen muchos más parámetros de configuración configurables para cada servicio concreto. Todos estos parámetros están comentados en ficheros de configuración por defecto y, por lo tanto, no se comentan en este Trabajo Final de Máster. Si bien existe una gran variedad de interfaces, tal y como se ha dicho antes, los servicios están preconfigurados con las IPs y puertos que se muestran en la Figura 54, de manera que todos los servicios se pueden ejecutar directamente en un ordenador sin modificar ningún parámetro (salvo los relacionados con el TAC, MCC y MNC).

En caso de querer ejecutar el eNodeB en otro ordenador diferente, sería necesario configurar las IPs y puertos de las interfaces S1-C y S1-U del núcleo para que tuvieran conexión desde el exterior. Cualquier otro servicio también sería transferible a otra máquina, siempre teniendo en cuenta qué interfaces se comunican con él para modificar sus parámetros en dicho servicio y en los adyacentes.

Una vez ejecutados todos los servicios, el servicio PGW-u crea una interfaz virtual llamada `ogstun` en la máquina donde se ejecuta (interfaz virtual equivalente a la interfaz virtual `srs_spgw_sgi` creada por `srsEPC`, sección 4.4.1). En esta interfaz se encuentra todo el intercambio de paquetes de usuario entre los servicios SPGW y los diferentes UEs. Al igual que sucedía con `srsEPC` si queremos que estos tengan conectividad hacia el exterior, es necesario configurar el ordenador que ejecuta el servicio PGW-u para que reencamine los paquetes a través de la interfaz de red e implemente NAT. En este caso, `open5Gs` recomienda realizar dicha configuración mediante la ejecución de los siguientes comandos:

```
### Enable IPv4/IPv6 Forwarding
sudo sysctl -w net.ipv4.ip_forward=1
sudo sysctl -w net.ipv6.conf.all.forwarding=1

### Add NAT Rule
sudo iptables -t nat -A POSTROUTING -s 10.45.0.0/16 ! -o ogstun -j MASQUERADE
sudo ip6tables -t nat -A POSTROUTING -s 2001:db8:cafe::/48 ! -o ogstun -j MASQUERADE
```

Finalmente, para configurar la base de datos de suscriptores se puede utilizar la interfaz web del *software* accediendo a la siguiente dirección URI <http://localhost:3000> con el nombre de usuario `admin` y la contraseña `1423`.

5. Casos de uso y resultados

En este capítulo se muestran las evaluaciones del sistema montado realizadas en este Trabajo Final de Máster. Se evaluaron una serie de casos de uso ligados a la conducción autónoma del vehículo a través de la maqueta en local (5.1), en la red con conexión wifi (5.2), en la red con conexión 4G (5.3) y en la red con conexión 4G y con *handover* (5.4). En cada sección se explica la topología de la red utilizada, la preparación de esta, y la configuración de varios sistemas de medida que se utilizaron para la obtención de diversos parámetros de la red. En todos los casos de uso se utilizaron secciones de la maqueta y el *software* que se explicó en el capítulo 3. Por otro lado, en los casos de uso en los que se utiliza la red 4G, se utilizó el *software* y *hardware* explicado en el capítulo 4.

5.1. Guiado del vehículo con procesamiento local

La primera evaluación del sistema que se realiza consiste en comprobar el funcionamiento del guiado del vehículo con procesado en su interior. El objetivo de esta evaluación es doble. Por una parte, conocer cual es el tiempo de procesamiento del algoritmo de guiado en el vehículo y comprobar la fiabilidad del algoritmo en función de la latencia de procesamiento.

5.1.1. Preparación de la red

Si bien este caso de uso no evalúa ningún parámetro de la red, es necesario que el vehículo esté conectado a una red para poder conectarse al servidor MQTT y, por lo tanto, poder recibir órdenes de encendido o apagado del algoritmo en local (gracias al *software* instalado explicado en la sección 3.2.3.). En la Figura 55 se muestra el esquema de red utilizado para poder enviar comandos MQTT al vehículo.

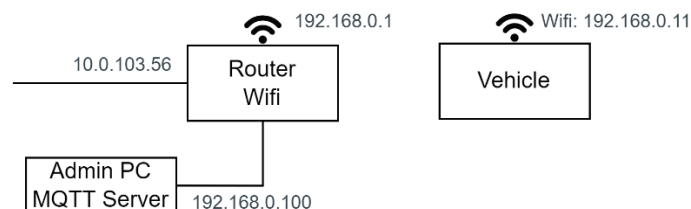


Figura 55. Topología de red para el guiado del vehículo con procesamiento en local.

5.1.2. Preparación de las medidas

Para medir la latencia por procesado del algoritmo se ha programado un cálculo de latencia media en el nodo ROS de ejecución de algoritmo autónomo en local. Tener en cuenta que el tiempo calculado es el tiempo medio desde que el nodo ROS de control autónomo del vehículo en local recibe una imagen hasta que publica en los tópicos de control del vehículo con los parámetros calculados. No se descarta que existan otros retardos relacionados con los nodos relacionados con la obtención de imágenes de la cámara o con el de control (nodos de fábrica programados por Amazon).

Por otro lado, para aumentar dicha latencia, se ha modificado el nodo ROS de ejecución del algoritmo autónomo en local, añadiendo una variable que permite aumentar la latencia de procesamiento del sistema.

5.1.3. Resultados

Tras ordenar al vehículo a seguir diferentes rutas a través de la maqueta con el algoritmo autónomo siendo ejecutado en local se obtuvieron los siguientes resultados:

- El tiempo de procesado medio es de **24ms**, es decir, se tardan aproximadamente 24ms desde que se recibe una nueva imagen de la cámara hasta que se envía orden a los actuadores del vehículo en función estas (Figura 56).
- La **estabilidad del sistema es mayor** cuanto **menor es la velocidad del vehículo** y cuanto **menor es la latencia del procesado**.
- Para una latencia de **24ms**, el vehículo sigue cualquier ruta de forma estable siempre que la velocidad no supere **2,5m/s** aproximadamente.
- Para una latencia de **100ms**, el vehículo sigue cualquier ruta de forma estable siempre que la velocidad no supere **1m/s** aproximadamente.
- Con latencias mayores de **100ms**, el comportamiento del vehículo se vuelve inestable en cualquier caso.

Comprobamos como existe una clara dependencia entre latencia y velocidad. Una mayor latencia implica una menor estabilidad que debe de ser compensada con una menor velocidad. Sin embargo, con latencias bajas se pueden alcanzar mayores velocidades. Por ejemplo, si la velocidad media del coche es de 1 m/s con una latencia del sistema de 100ms significa que el coche recorre 10 cm antes de reaccionar a cualquier evento en la carretera debido a ese retraso de ida y vuelta. Al duplicar la velocidad, la distancia se duplica, lo que dificulta la adaptación del vehículo a los cambios de trayectoria.

Con latencias mayores a 100ms resulta muy difícil lograr estabilidad ya que lograr una velocidad menor de 1m/s sin un sistema retroalimentado (únicamente permaneciendo constante la variable del acelerador sin percepción de la velocidad del vehículo) es muy complicado.

Por lo tanto, obtenemos un requisito fundamental para el sistema: en los casos de uso en los que el procesado se haga fuera del vehículo **es necesario garantizar que la latencia global del sistema sea menor de 100ms** para que nuestra aplicación de guiado funcione.

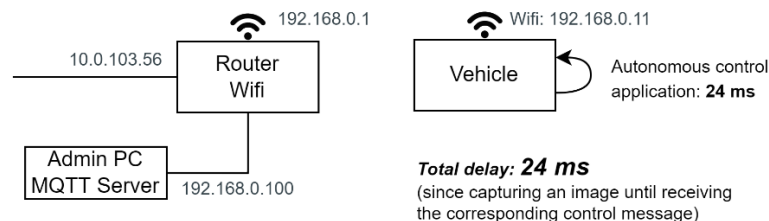


Figura 56. Retardos con guiado del vehículo con procesamiento en local.

5.2. Guiado del vehículo con procesamiento en servidor a través de wifi.

En este caso de uso se va a evaluar el funcionamiento del sistema cuando se ejecuta el algoritmo de conducción autónoma en un servidor de aplicaciones a través conexión wifi. Se comprueban los retardos implicados, el *jitter* y el ancho de banda utilizado.

5.2.1. Preparación de la red

La topología de red utilizada en este caso de uso es parecida a la utilizada en el caso de uso anterior (sección 5.1) pero agregando un servidor de aplicaciones (servidor *edge* o frontera) conectado al mismo *router*, es decir, muy cerca del vehículo. De esta forma, a través de la conexión wifi podremos enviar ordenes al vehículo con el servidor MQTT y el vehículo podrá ser guiado de forma autónoma a través del servidor de aplicaciones. El servidor de aplicaciones lo implementa un Intel NUC (especificaciones dadas en la sección 1.4) que ejecuta la aplicación desarrollada para la ejecución del

algoritmo autónomo en la red (explicado en la sección 3.3). En la Figura 57 se muestra un esquema de la topología de la red utilizada.

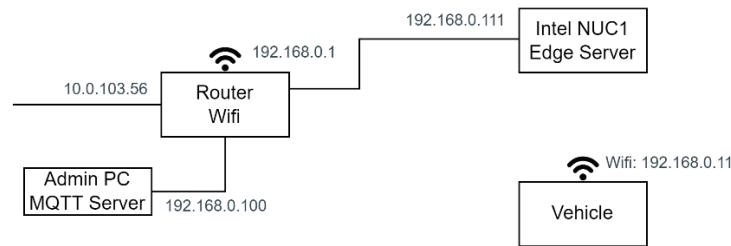


Figura 57. Topología de red para el guiado del vehículo con procesamiento en servidor de aplicaciones a través de wifi.

Se ha configurado en el vehículo la transmisión de las imágenes JPEG con un factor de calidad de 12 (cuanto mayor es el número, menor es la compresión). Con dicho factor de calidad garantizamos que las imágenes no superaban nunca el tamaño máximo del payload de un paquete UDP (65000 bytes).

5.2.2. Preparación de las medidas

Para efectuar las medidas de retardos de la red se han utilizado diferentes métodos en función del retardo a medir:

- Para medir los **retardos de procesamiento de la aplicación** de conducción autónoma en el servidor de la red se ha **modificado el software** de la aplicación para que genere una media de lo que tarda en responder la llamada a la función `proceso_fotograma`.
- Para medir los **retardos de procesamiento por descompresión** de la imagen JPEG recibida en el servidor se ha **modificado el software** de la aplicación para que genere media de lo que tarda en ejecutarse la función correspondiente.
- Para medir los **retardos de procesamiento por compresión** de la imagen JPEG en el vehículo se ha **modificado el software** del nodo ROS de control autónomo en la red para que genere una media de lo que tarda en ejecutarse la función correspondiente.
- Para medir los **retardos**, el **jitter**, y el **ancho de banda** utilizado en la conexión entre el vehículo y el servidor de aplicaciones se ha utilizado el **software Qosium** junto con sistema de sincronización de relojes, cuyo uso se explica a continuación.

Qosium [57] es un *software* desarrollado por la empresa finlandesa Kaitotec [58] que permite realizar multitud de mediciones en red y visualizarlas en tiempo real mediante una interfaz gráfica. Es una solución de medición distribuida, que se compone de tres tipos de componentes: agente de medición, controlador de medición y sistema de resultados. De estos componentes, el agente de medición y el controlador de medición son siempre necesarios y son los que se utilizan en este Trabajo Final de Máster. Su funcionamiento se muestra a continuación (Figura 58):

- Los agentes de medición son implementados a través del **software Qosium Probe**. Este *software* debe ser instalado en todos los dispositivos de la red a partir de los cuales deseemos obtener medidas. Es un *software* ligero y pasivo, transparente para la red, capaz de analizar todo el tráfico que circula por las diferentes interfaces de red del dispositivo.
- El controlador de medición es implementado a través del **software Qosium Scope**. Consiste en un analizador en tiempo real, ejecutable en cualquier ordenador. Este permite configurar las medidas que se desean realizar (definiendo filtros, parámetros a medir, etc.) y, una vez configuradas, se conecta a los dispositivos que tienen instalado Qosium Probe, lanza las mediciones correspondientes, registra la información proveniente de estos y las muestra y ordena en una interfaz gráfica.

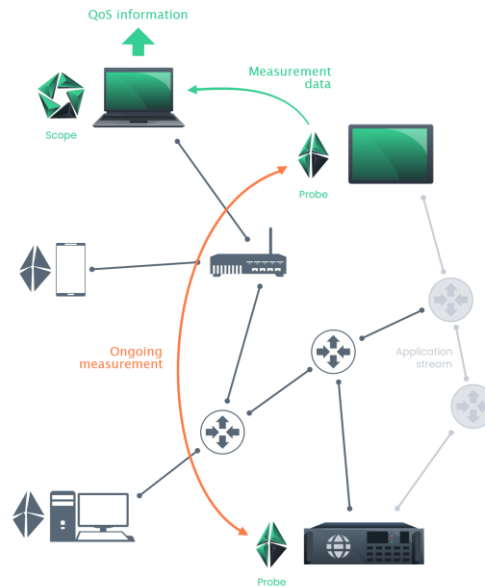


Figura 58. Ejemplo de funcionamiento del software Qosium para la medición del tráfico de una aplicación concreta.

Las medidas que realiza este *software* son mayormente pasivas, de forma que no necesita modificar los paquetes de la red para poder hacer, por ejemplo, mediciones de retardos de ida o de vuelta entre dos dispositivos.

Para una correcta medición de retardos unidireccionales, es necesario que los relojes de ambos dispositivos estén perfectamente sincronizados de forma que Qosium pueda calcular correctamente los retardos simplemente restando el *timestamp* en el que los paquetes se recibieron con el *timestamp* en el que los paquetes se enviaron. Si los relojes no están sincronizados, las mediciones de retardo entre dispositivos no serán fiables. Existen múltiples protocolos diseñados para lograr una correcta sincronización de relojes entre máquinas, como NTP (*Network Time Protocol*), SNTP (*Simple Network Time Protocol*) o PTP (*Precision Time Protocol*). En la siguiente referencia ofrecida por el Instituto Nacional de Ciberseguridad (INCIBE) [59] se puede ver un análisis del funcionamiento de cada protocolo y una comparativa. En la referencia se justifica que el protocolo PTP es el que ofrece mayor precisión y, por lo tanto, es el que utilizamos para sincronizar los relojes entre nuestras máquinas (en este caso de uso entre el vehículo y el servidor de aplicaciones).

Para utilizarlo, instalamos el servicio `ptpd` en el servidor de aplicaciones y en el vehículo mediante el siguiente comando.

```
sudo apt install ptpd
```

El protocolo PTP funciona definiendo un ordenador maestro y varios ordenadores esclavos, que se sincronizan con el reloj del maestro. Puede funcionar en dos modos diferentes: multicast o unicast.

En el modo multicast el ordenador maestro envía un mensaje que es recibido por todos los dispositivos de la misma subred. Esta opción es la recomendada por la documentación de Qosium [60] para sincronizar correctamente los relojes entre dispositivos y es la que utilizaremos para el análisis de este caso de uso. Para operar en este modo es necesario ejecutar el siguiente comando en el ordenador maestro, en nuestro caso el Intel NUC que hace de servidor de aplicaciones.

```
sudo ptpd --masteronly --interface eno1 --verbose
```

Por otro lado, en las máquinas clientes, en nuestro caso el vehículo, es necesario ejecutar el siguiente comando.

```
sudo ptpd --slaveonly --interface eno1 --verbose
```

Una vez realizado, podemos ejecutar el *software* Qosium Probe en ambas máquinas y realizar diferentes medidas entre ellas.

5.2.3. Resultados

Tras ordenar al vehículo a seguir diferentes rutas a través de la maqueta con el algoritmo autónomo siendo ejecutado en el servidor a través de una conexión wifi se obtuvieron los siguientes resultados:

- El **ancho de banda medio necesario de subida** es de **7,8Mbps**. Este ancho de banda varía en función de las imágenes a transmitir debido a la compresión JPEG. Imágenes monotonas, con pocos contrastes ocupan poco ancho de banda y viceversa.
- El **ancho de banda medio necesario de bajada** es de **5,58Kbps**. Es mucho más bajo debido a que los únicos paquetes que recibe el vehículo son los relacionados con el mantenimiento de la conexión MQTT y los mensajes UDP de control provenientes del servidor.
- El **retardo total del sistema** desde que se captura una imagen hasta que se recibe el mensaje de control correspondiente es de **34,6ms** (ver Figura 59). Este retardo se divide en los siguientes:
 - Retardo de procesamiento por **compresión** de la imagen en el vehículo: **12,6ms**.
 - Retardo de transmisión en **envío** de imagen a servidor a través de *router* wifi: **1ms**.
 - Retardo de procesamiento por **decompresión** de la imagen en servidor: **7ms**.
 - Retardo de procesamiento de la aplicación de **guiado** autónomo en servidor: **13ms**.
 - Retardo de transmisión en **envío** de mensaje de control a vehículo a través de *router* wifi: **1ms**.

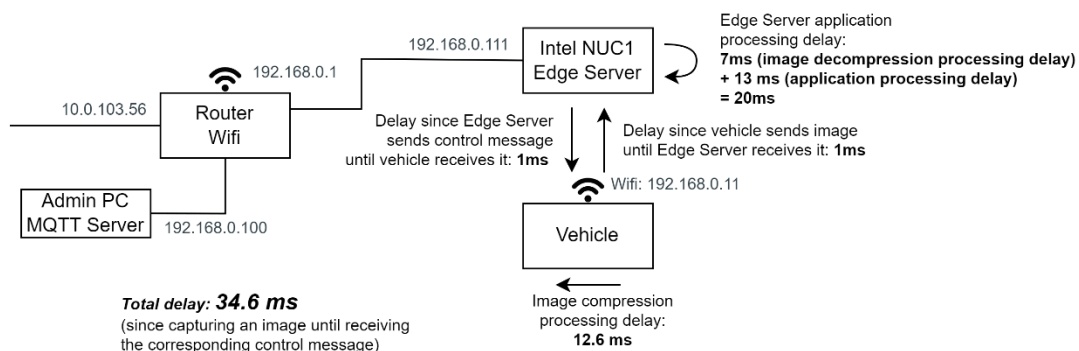


Figura 59. Retardos con guiado del vehículo con procesamiento en servidor de aplicaciones a través de wifi.

Comprobamos como en este caso obtenemos un retardo global mayor que en el caso anterior. El algoritmo autónomo se ejecuta de forma más rápida en el servidor que en el vehículo (13ms por imagen en vez de 24ms). Sin embargo, la transmisión de la imagen implica una compresión de las imágenes en el vehículo (que añade un retardo de 12.6ms) y una decompresión de las imágenes en el servidor (que añade otro retardo de 7ms). Llama la atención como el retardo que añade el *router* wifi para la transmisión y recepción de mensajes no es mayor de 2ms en total.

Concluimos con que, para esta aplicación concreta, **la realización de off-loading en la red no compensa debido a los tiempos de procesamiento por compresión y decompresión de la imagen**. Sin embargo, en el caso de querer implementar aplicaciones que impliquen una mayor carga computacional no ejecutable en el vehículo, y que estén optimizadas para ser ejecutadas en un servidor (por ejemplo utilizando tarjetas gráficas), llevar la ejecución de estas a la red puede permitir retardos del sistema mucho menores.

5.3. Guiado del vehículo con procesamiento en servidor a través de 4G sin *handover*

En este caso de uso se va a evaluar el funcionamiento del sistema cuando se ejecuta el algoritmo de conducción autónoma en un servidor de aplicaciones a través de nuestra propia red 4G implementada con las herramientas explicadas en el capítulo 4. Se comprueban los retardos implicados, el *jitter* y el ancho de banda utilizado.

La implementación de este caso de uso es similar a la implementación de un MEC con EPCs distribuidos. Tal y cómo se explicó en la sección 2.1.4, en este tipo de implementaciones el despliegue del host MEC se encuentra detrás del núcleo de la red, y es el núcleo el encargado de asignar la IP y la información DNS concreta para resolver las aplicaciones MEC. En nuestro caso de uso únicamente utilizaremos un núcleo e ignoraremos la asignación de IPs realizada por el núcleo, es el mismo vehículo el que está configurado para solicitar el servicio directamente al MEC y no a un servidor externo alejado.

5.3.1. Preparación de la red

La topología de red utilizada en este caso de uso es la que se muestra en la Figura 60. Los elementos relacionados con la red 4G implementada son los siguientes:

- **Núcleo de la red 4G:** Utilizamos un Intel NUC (especificaciones dadas en la sección 1.4) para implementar el núcleo de la red 4G. Este ordenador ejecuta todos los servicios del *software* Open5GS⁷ relacionados con 4G explicados en la sección 4.4.2. Dichos servicios están configurados⁸ para conectarse correctamente al eNodeB e incluir los datos de la SIM del vehículo. El servicio está configurado para que asigne IPs a los UEs dentro de la subred 10.45.0.0/16. Ese ordenador también se encarga de hacer de servidor de aplicaciones y, por lo tanto, será quien ejecute la aplicación de guiado autónomo en la red. El ordenador tiene dos interfaces de red *ethernet*, ambas conectadas al *router* con diferentes IPs, el núcleo está configurado para utilizar una interfaz y la aplicación la otra. De esta manera independizamos el tráfico relacionado con el núcleo de la red del tráfico relacionado con la aplicación, facilitando el análisis.
- **eNodeB de la red 4G:** Utilizamos un Intel NUC (especificaciones dadas en la sección 1.4) para implementar el eNodeB de la red 4G. Este ordenador ejecuta el *software* srsENB explicado en la sección 4.3, configurado⁸ para conectarse correctamente al núcleo de la red y al dispositivo SDR utilizado. Utilizamos la banda con ARFCN 3200 de 10MHz, licitada por Vodafone.⁹ El ordenador ejecuta este *software* en modo de tiempo real con un *kernel* de baja latencia instalado (siguiendo las instrucciones explicadas en la sección 4.3.3). Está conectado al *router* a través de una interfaz *ethernet*.

⁷ Este caso de uso también fue probado con el núcleo srsEPC desarrollado por srsRAN. Sin embargo, como los resultados fueron muy similares, se analiza en profundidad sólo el caso de uso con el núcleo Open5GS, que ofrece varias ventajas explicadas en la sección 2.2.1.5).

⁸ Todos los ficheros de configuración utilizados para este caso de uso se pueden encontrar en el repositorio Artemis de la página GitHub del Grupo de Comunicaciones Ópticas de la Universidad de Valladolid [45].

⁹ En el Anexo V se puede observar un estudio las bandas FDD 4G LTE activas en España compatibles con el módem que utilizamos, donde se ve cuáles están libres y cuáles están ocupadas en la zona de la ETSIT UVA. Dicho análisis se realizó utilizando los dispositivos SDR como analizadores de espectro. Así mismo, en el anexo se encuentra un análisis de otras bandas compatibles con el módem que se utilizan en otros países, así como su correspondencia en España.

- **Dispositivo SDR:** Utilizamos el dispositivo BladeRFxA9 (explicado en la sección 4.2.1) con dos antenas para transmisión y dos antenas para recepción (para realizar MIMO 2x2). Este se conecta al eNodeB a través de un puerto USB 3.0.
- **Cliente 4G en el vehículo:** Utilizamos el módem 4G Huawei E3372 (instalación de este dispositivo en el vehículo explicada en la sección 3.2.1). Incluimos en este módem una de la SIMs programables explicadas en la sección 4.1 cuyos datos están registrados en el núcleo de la red. Este módem 4G implementa un servicio NAT y DHCP, de forma que la IP asignada al vehículo no es la IP que asigna el núcleo de la red 4G al módem. En nuestro caso, mientras el núcleo está configurado para asignar la IP 10.45.0.2 al módem 4G, el módem 4G, con sus servicios NAT y DHCP, asigna al vehículo la IP 192.168.8.100, siendo la IP del módem vista desde el vehículo la 192.168.8.1.

Aparte de estos dispositivos, al igual que en los casos de uso anteriores es necesario que el vehículo pueda conectarse al servidor MQTT y, por lo tanto, poder recibir órdenes de encendido o apagado del algoritmo en local (gracias al *software* instalado explicado en la sección 3.2.3.). Para ello, existe un ordenador del administrador que implementa un servidor MQTT conectado al *router*. Este se comunicará con el vehículo a través de la misma red 4G.

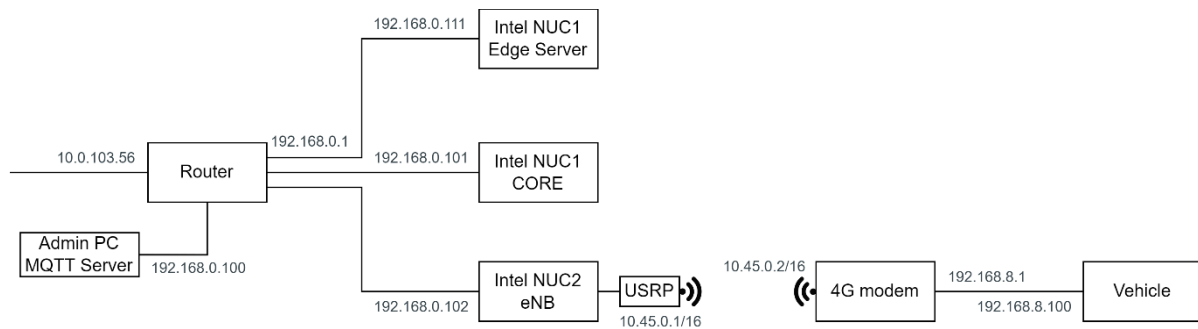


Figura 60. Topología de red para el guiado del vehículo con procesamiento en servidor de aplicaciones a través de nuestra propia red 4G.

5.3.2. Preparación de las medidas

Para preparar las medidas en este caso de uso, es necesario tener en cuenta cómo interactúan los diferentes elementos del sistema. En la Figura 61 se muestra por donde circula un mensaje desde que sale del vehículo hasta que llega al servidor de aplicaciones. Una vez que el mensaje sale del vehículo, el módem 4G realiza un primer NAT, asignando al mensaje la dirección origen del módem. Una vez llega al eNodeB este encapsula el mensaje en GTP y lo envía al núcleo de la red. El núcleo desencapsula el mensaje e implementa otro NAT, de manera que el mensaje enviado al servidor de aplicaciones incluye la IP origen del núcleo de la red, no el del módem 4G ni el del vehículo.

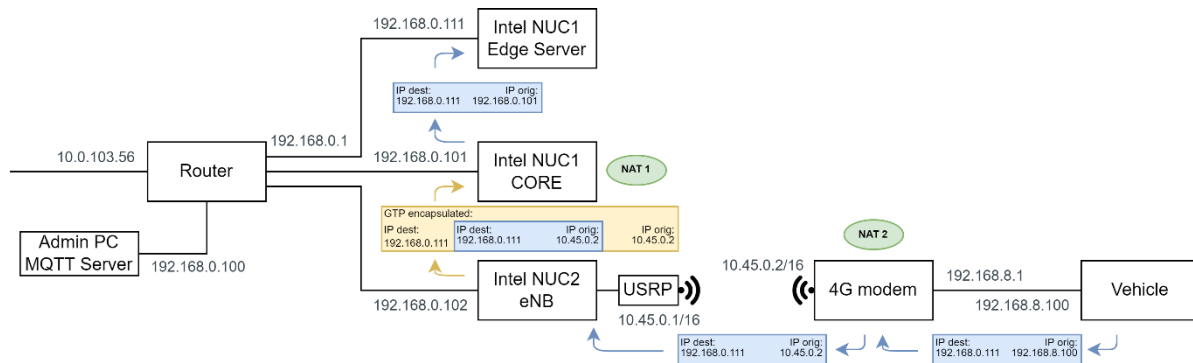


Figura 61. Ejemplo de envío de mensaje desde el vehículo hasta el servidor de aplicaciones para el caso de uso de guiado del vehículo con procesamiento en servidor de aplicaciones a través de nuestra propia red 4G. Direcciones IPs, NAT y encapsulado GTP.

Como vemos existen múltiples elementos que van a añadir retardos al sistema completo. A continuación se explica cómo se realiza cada medida:

- Para medir los **retardos de procesamiento de la aplicación** de conducción autónoma en el servidor de la red se ha **modificado el software** de la aplicación para que genere una media de lo que tarda en responder la llamada a la función `proceso_fotograma`.
- Para medir los **retardos de procesamiento por descompresión** de la imagen JPEG recibida en el servidor se ha **modificado el software** de la aplicación para que genere una media de lo que tarda en ejecutarse la función correspondiente.
- Para medir los **retardos de procesamiento por compresión** de la imagen JPEG en el vehículo se ha **modificado el software** del nodo ROS de control autónomo en la red para que genere una media de lo que tarda en ejecutarse la función correspondiente.
- Para medir los **retardos por encapsulado/desencapsulado y reenvío de paquetes del núcleo** se ha utilizado la herramienta **Wireshark** [61] y se ha modificado el contenido de los paquetes UDP de forma que estos incluyan un número del 0 al 255 en su cabecera. De esta forma se puede hacer una correspondencia entre los paquetes de una interfaz y los de la otra y, por lo tanto, calcular el retardo medio.
- Para medir los **retardos que añade el módem 4G**, debido a la imposibilidad de instalar ningún *software* en el módem, sólo podemos comprobar el tiempo de ida y vuelta a través de un mensaje ICMP al cual el módem responde.
- Para medir los **retardos**, el **jitter**, y el **ancho de banda** utilizado en la conexión entre el vehículo y el servidor de aplicaciones y, entre el vehículo y el eNodeB, se ha utilizado el **software Qosium** junto con el sistema de sincronización de relojes explicado en la sección anterior. Debido a que en este caso se trabaja con una red más compleja, con dos servicios NAT en medio, se requiere realizar una configuración ligeramente diferente a la explicada en el caso de uso anterior. Esta configuración se explica a continuación.

Tal y como se ha explicado en el caso de uso anterior, en la sección 5.2.2, el *software* Qosium incluye dos *softwares* principales: Qosium Scope, *software* que se puede instalar en cualquier ordenador; y Qosium Probe, *software* que se instala en los elementos relevantes en las medidas a realizar. Al realizar la medida, en Scope se configuran las IPs y puertos donde se encuentran los dispositivos con Probe instalado, por lo tanto, Scope necesita poder establecer conexión con estos elementos. Este *software*, en nuestro caso lo tenemos instalado en el ordenador del administrador (donde también se encuentra el bróker MQTT, ver Figura 61). Al tener que atravesar dos servicios NAT para poder alcanzar el vehículo, es necesario seguir una serie de pasos:

1. Para conectarse a un vehículo desde el exterior, lo hacemos a través de la IP que le asigna el núcleo de la red, en nuestro caso 10.45.0.2. Necesitamos que estos paquetes lleguen al *núcleo*

de la red así que es necesario **configurar el router con una nueva regla de reencaminamiento** para que todos los paquetes que se encuentren dentro de la subred 10.45.0.0/16 se dirijan hacia el núcleo de la red.

2. Si bien el núcleo de la red está configurado para realizar un NAT con la subred de las UEs, cualquier paquete que reciba por cualquier interfaz con una dirección IP de la subred de las UEs la reencaminará por la interfaz virtual que genera el *software* Open5GS (ver Figura 53) y, por lo tanto, serán paquetes enviados procesados por el núcleo que llegarán al UE.
3. Debido al servicio NAT que implementa el **módem 4G**, es necesario **abrir los puertos correspondientes** en dicho dispositivo y configurar una regla de redireccionamiento. El módem 4G utilizado no permite realizar esta configuración a través de su interfaz web pero implementa el **protocolo UPnP**, un conjunto de protocolos de comunicación que permite comunicarse con el módem para configurar, entre otros, el redireccionamiento de puertos. Para abrir los puertos y configurar el redireccionamiento a través de este protocolo debemos instalar el *software* upnpc en el vehículo a través del siguiente comando:

```
sudo apt install miniupnpc
```

Una vez instalado, debemos ejecutar el siguiente comando para abrir los puertos y redireccionarlos al mismo puerto de la red interna. Los puertos que abrimos son los relacionados con el servicio de sincronización de relojes PTP (319 y 320) y con el servicio de Qosium (8177):

```
upnpc -r 319 udp 320 udp 319 tcp 320 tcp 8177 tcp 8177 udp
```

Una vez seguidos estos pasos, podremos ejecutar Qosium Scope en el ordenador del administrador y, mediante una configuración concreta del *software* (disponible en el repositorio Artemis de la página GitHub del Grupo de Comunicaciones Ópticas de la Universidad de Valladolid [45]), podremos realizar medidas de red conectándonos al Qosium Probe que ejecuta el vehículo.

Dichos pasos también son necesarios para establecer una correcta sincronización de relojes entre el vehículo y otro elemento de la red. Utilizamos también el protocolo PTPD explicado en el caso de uso anterior (sección 5.2.2) pero, en vez de utilizar el modo multicast, en este caso es necesario ejecutar el servicio ptpd en modo unicast.

Para configurar la sincronización de relojes entre el servidor de aplicaciones y el vehículo deberíamos ejecutar los siguientes comandos:

- En el servidor de aplicaciones: `sudo ptpd --masteronly -u 10.45.0.2 --interface eno1 -verbose`
- En el vehículo: `sudo ptpd --slaveonly -u 192.168.0.111 --interface eth0 -verbose`

Para configurar la sincronización de relojes entre el eNodeB y el vehículo deberíamos ejecutar los siguientes comandos:

- En el eNodeB: `sudo ptpd --masteronly -u 10.45.0.2 --interface eno1 -verbose`
- En el vehículo: `sudo ptpd --slaveonly -u 192.168.0.102 --interface eth0 --verbose`

5.3.3. Resultados

Tras ordenar al vehículo a seguir diferentes rutas a través de la maqueta con el algoritmo autónomo siendo ejecutado en el servidor a través de nuestra propia red 4G se obtuvieron los siguientes resultados:

- El **ancho de banda medio necesario de subida** es de **7,8Mbps**. Este ancho de banda varía en función de las imágenes a transmitir debido a la compresión JPEG. Imágenes monotonas, con pocos contrastes ocupan poco ancho de banda y viceversa.

- El **ancho de banda medio necesario de bajada** es de **5,58Kbps**. Es mucho más bajo debido a que los únicos paquetes que recibe el vehículo son los relacionados con el mantenimiento de la conexión MQTT y los mensajes UDP de control provenientes del servidor.
- El **retardo total del sistema** desde que se captura una imagen hasta que se recibe el mensaje de control correspondiente es de **90,4ms** (ver Figura 62). Este retardo se divide en los siguientes:
 - Retardo de procesamiento por **compresión** de la imagen en el vehículo: **12,6ms**.
 - Retardo de **enlace ascendente** desde el **vehículo** hasta el **eNodeB**: **24,4ms**.
 - Retardo de **enlace descendente** desde el **eNodeB** al **vehículo**: **18,9ms**.
 - Tareas de encapsulado/desencapsulado y **reenvío** del **núcleo**: **0,2ms**.
 - Retardo de procesamiento por **decompresión** de la imagen en servidor: **11,5ms**.
 - Retardo de procesamiento de la aplicación de **guiado** autónomo en servidor: **21,5ms**.
 - Retardo de transmisión en **envío** de mensaje de control a vehículo a través de **router wifi**: **1ms**.

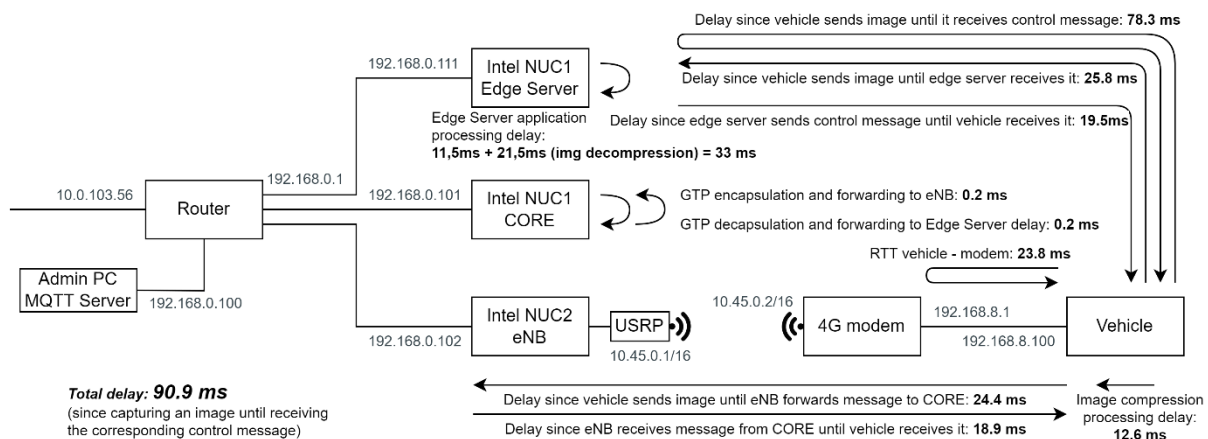


Figura 62. Retardos con guiado del vehículo con procesamiento en servidor de aplicaciones a través nuestra red 4G.

Tal y como explicamos anteriormente (sección 5.3.2), un aspecto que no hemos podido analizar es el retraso añadido por el **módem 4G**. A través de los mensajes ICMP comprobamos que el **retardo de ida y vuelta es de 23,8ms**, lo que sugiere que el módem añade un retraso considerable.

Comprobamos como en este caso obtenemos un retardo global mayor que en el caso anterior. Mientras que a través de WiFi teníamos un retardo de subida y de bajada al servidor de aplicaciones de 1ms , a través de nuestra red 4G tenemos un retardo de subida de $25,8\text{ms}$ y un retardo de bajada de $19,5\text{ms}$. Desconocemos qué parte del retardo es debido al módem 4G utilizado. Por otro lado, el retardo en el servidor de aplicaciones ha subido respecto al caso de uso por wifi (20ms en el caso de uso por wifi y 33ms en este caso de uso). Esto es debido a que dicho servicio se ejecuta en el mismo ordenador en el que se ejecutan los servicios del *núcleo* de la red. En caso de ejecutar el servidor de aplicaciones en otra máquina independiente no existiría diferencia en estos tiempos de procesado.

Concluimos con que **es posible realizar off-loading en la red a través de 4G con un servidor en la frontera de la red**. Al no superar el retardo global de 100ms (condición necesaria obtenida en el primer caso de uso, sección 5.1.3), el vehículo puede ser guiado de forma estable siempre que no se supere la velocidad de 1m/s .

5.4. Guiado del vehículo con procesamiento en servidor a través de 4G con *handover*

En este caso de uso se pretendía evaluar el funcionamiento del sistema cuando se ejecuta el algoritmo de conducción autónoma en un servidor de aplicaciones a través de nuestra propia red 4G, implementada con las herramientas explicadas en el capítulo 4, con dos estaciones base y la realización de *handover* entre ambas. Este caso de uso no se logró implementar correctamente debido a una serie de inconvenientes que se explicarán más adelante. Únicamente se comprobó el funcionamiento de la red en general y no se llegó a ejecutar el algoritmo de guiado autónomo en la red con *handover*.

El *handover* que se quería implementar era el del tipo SI intrafrecuencia, es decir, utilizando la misma frecuencia en ambas estaciones base. Este tipo de *handover* tiene lugar a través de la interfaz SI cuando un UE pasa de la cobertura de un eNB al siguiente. El servicio MME, conectado a ambas estaciones base, es el que gestiona y habilita dicho *handover* comunicándose simultáneamente con ambos eNodeB a través de las interfaces SI.

5.4.1. Preparación de la red

La topología de red utilizada para evaluar el *handover* SI en nuestra red 4G es la que se muestra en la Figura 63. Es muy similar a la utilizada en el caso de uso anterior (sección 5.3.1) con la diferencia de que, en este caso, se utilizan dos estaciones base en vez de una. Utilizamos el núcleo implementado con Open5GS ya que srsEPC no es compatible con el *handover* SI. Como eNodeB utilizamos en ambas estaciones base la implementación srseNB conectada a los dispositivos SDR BladeRF. Finalmente, como módem 4G utilizamos el Huawei E3372 conectado al vehículo.

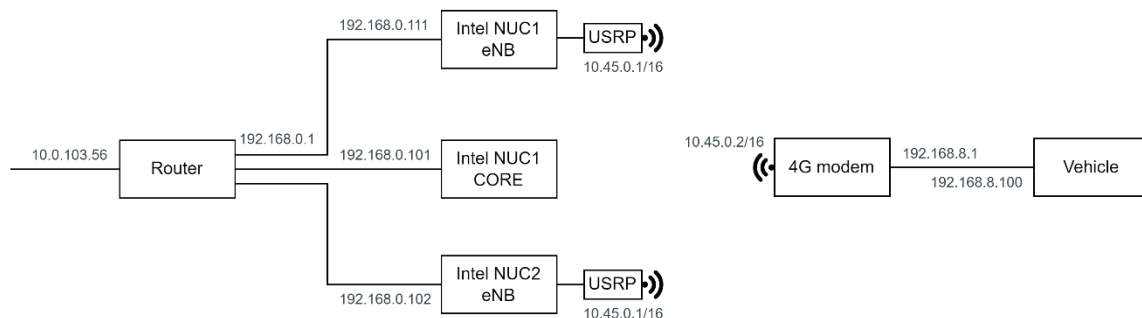


Figura 63. Topología de red para la evaluación del *handover* en nuestra propia red 4G.

Para implementar el *handover* SI, el núcleo Open5GS no necesita ninguna configuración diferente a la utilizada en el caso de uso anterior (sección 5.3.1). Sin embargo, las estaciones base sí que deben de tener una configuración especial. En la documentación oficial de srsRAN se encuentra un manual que explica como configurar srsRAN para implementar *handover* utilizando como núcleo Open5GS [62]. A continuación se explican las modificaciones mínimas necesarias:

- El parámetro `enb.enb_id` (identificador del eNodeB) del fichero de configuración `enb.conf` de cada estación base implicada en el *handover* debe de ser diferente, para poder diferenciar cada estación base. En nuestro caso, ponemos los identificadores `0x19B` y `0x19C`.
- Cada estación base debe de llevar asignado también un PCI (*Physical Cell Identifier*) que se debe de configurar en el fichero `rr.conf` de cada estación base. Dicho identificador también debe de ser diferente. En nuestro caso hemos utilizado el PCI 1 y 6. Cada estación debe conocer el resto de estaciones base con las que un cliente puede hacer *handover*. Para configurar esto, se debe modificar el fichero `rr.conf` de cada estación base y rellenar el apartado

meas_cell_list indicando el identificador de cada celda (incluida ella misma), el ARFCN y el PCI. A continuación se muestra un ejemplo:

```
// Cells available for handover
meas_cell_list =
(
{
eci = 0x19B01;           # Celda 1
dl_earfcn = 2850;
pci = 1;
},
{
eci = 0x19C01;           # Celda 2
dl_earfcn = 2850;
pci = 6;
}
);
```

- Todas las estaciones base deberán estar configuradas para conectarse al mismo núcleo a través de la misma IP y puerto.

Existen otros parámetros de configuración en el fichero rr.conf que permiten configurar el tipo de evento, el ciclo de histéris, temporizadores y otras variables relacionadas con el *handover*. En la siguiente referencia [63] se puede observar un análisis de las variables implicadas en el *handover*, todas ellas configurables en el fichero rr.conf de las estaciones base srsRAN.

5.4.2. Resultados

A la hora de poner en marcha este caso de uso nos encontramos con una serie de dificultades.

En la primera evaluación, el cliente se conectaba correctamente a la estación base. Al entrar en la celda de la otra estación base y ejecutarse el proceso del *handover*, el proceso finalizaba aparentemente con éxito pero el enlace descendente dejaba de funcionar, mientras que el ascendente seguía funcionando. Leyendo los registros de la estación base destino en el *handover* encontramos que el núcleo Open5GS envía correctamente los mensajes a través de GTP con el TEID (Identificador del túnel GTP) correcto y srseNB lo recibía. Sin embargo srseNB dejaba los mensajes en el buffer como podemos ver en los siguientes logs:

```
[GTPU ] [I] Rx S1-U SDU, 192.168.0.102:0x1 > DL (buffered), rmti=0x46, eps-BearerID=5, n_bytes=90, IPv4 8.8.4.4 > 10.45.0.23
[GTPU ] [I] Rx S1-U SDU, 192.168.0.102:0x1 > DL (buffered), rmti=0x46, eps-BearerID=5, n_bytes=90, IPv4 8.8.8.8 > 10.45.0.23
[GTPU ] [I] Rx S1-U SDU, 192.168.0.102:0x1 > DL (buffered), rmti=0x46, eps-BearerID=5, n_bytes=48, IPv4 13.107.6.158 > 10.45.0.23
[GTPU ] [I] Tx S1-U SDU, UL > 192.168.0.102:0x1, rmti=0x46, eps-BearerID=5, n_bytes=40, IPv4 10.45.0.23 > 13.107.6.158
[GTPU ] [I] Tx S1-U SDU, UL > 192.168.0.102:0x1, rmti=0x46, eps-BearerID=5, n_bytes=40, IPv4 10.45.0.23 > 20.61.102.89
[GTPU ] [I] Tx S1-U SDU, UL > 192.168.0.102:0x1, rmti=0x46, eps-BearerID=5, n_bytes=41, IPv4 10.45.0.23 > 140.82.112.25
```

Este comportamiento se comentó en el foro de GitHub de srsRAN [64] y nos respondieron que el motivo era que el núcleo de la red Open5GS debía enviar un marcador de finalización de *handover* al eNodeB fuente en el *handover* para que este al mismo tiempo lo enviara al eNodeB destino. Si este marcador no se envía, la estación destino se mantiene a la espera para comenzar a utilizar el enlace de descarga. Por lo tanto, era un problema de la implementación del núcleo Open5GS. Se nos recomendó configurar un temporizador en el fichero enb.conf de ambas estaciones base de forma que, si tras cierto tiempo no se enviaba dicho mensaje, se ignorara su no recepción. El parámetro a modificar era `gtpu_tunnel_timeout` dentro del fichero `enb.conf`.

Una vez modificado, la conexión se establecía correctamente al realizar un *handover*, teniendo enlace de bajada y de subida funcional. Sin embargo, el enlace de subida tenía muy limitado el ancho de banda.

Tras analizar los registros vimos que, al realizar el *handover* el valor de PHR¹⁰ (*Power Headroom*) se vuelve negativo (-21) incluso cuando nuestro módem 4G estaba junto a la estación base a la que está conectado. Esto hacía que el esquema de modulación del canal de subida bajara drásticamente para intentar tener una conexión más robusta, con menor ruido. Un esquema de modulación con menos símbolos implica menor tasa de transmisión, lográbamos únicamente 1Mbps, insuficiente para nuestro caso de uso. Este fenómeno sólo ocurría cuando el módem 4G se conecta a un eNB mediante el procedimiento de *handover*. Sin embargo, cuando se conecta a dicho eNB directamente desde el principio teníamos un PHR positivo (34 aproximadamente).

Este comportamiento también se explicó en el foro de GitHub de srsRAN [64] pero no se recibió contestación. Por lo tanto, si bien se logró una red con *handover* funcional, la tasa de subida resultante al realizar el *handover* era insuficiente para evaluar correctamente nuestro caso de uso.

¹⁰ El parámetro *Power Headroom* indica cuánta potencia de transmisión le queda a un equipo de usuario para utilizar, además de la potencia utilizada por la transmisión actual. Si es positivo significa que el equipo de usuario no está utilizando toda la potencia disponible para transmitir porque no es necesario (tiene buena cobertura). Si es negativo significa que el equipo de usuario está utilizando toda la potencia disponible para transmitir y que se estima que no es suficiente para lograr una buena calidad de conexión (tiene mala cobertura).

6. Conclusiones y líneas futuras

6.1. Conclusiones

Tomando en cuenta los objetivos explicados en la sección 1.2, podemos concluir que en este Trabajo Final de Máster se ha elaborado un banco de pruebas de servicios de vehículo conectado a través de redes 4G, y que se ha implementado un caso de uso basado en la conducción de un vehículo autónomo realizando *off-loading* en la red.

Respecto al banco de pruebas, se ha diseñado y elaborado una maqueta que facilita la implementación de un algoritmo de guiado para convertir un vehículo en autónomo, mediante líneas pintadas en el centro de las carreteras; suficientemente grande para poder simular escenarios realísticos; con desvíos, incorporaciones y vías de varios carriles para evaluar múltiples casos de uso; y modular para poder montarse y desmontarse con facilidad y ofrecer flexibilidad a la hora de crear nuevos circuitos.

Se ha seleccionado un vehículo que pueda emular sistemas de sensórica de los vehículos autónomos en la actualidad, con un ordenador programable capaz de comunicarse con los actuadores y la sensórica y con wifi. El *hardware* de dicho vehículo se ha modificado para agregarle mayor batería y comunicación 4G, permitiendo evaluar casos de uso de red tanto wifi como 4G.

Se ha realizado un análisis y modificado el *software* de dichos vehículos, permitiendo la ejecución de un algoritmo de guiado autónomo dentro y fuera del vehículo en tiempo real y permitiendo al administrador comunicarse con dichos vehículos a través de la red para enviarles órdenes y recibir información. Así mismo, se ha desarrollado una aplicación ejecutable en un servidor, que puede comunicarse con los vehículos para ejecutar el algoritmo de guiado autónomo en la red.

Se ha desarrollado un algoritmo de guiado autónomo completo basado en el seguimiento de las líneas pintadas en el centro de las carreteras de la maqueta. Dicho algoritmo se puede ejecutar tanto en el vehículo como en un servidor de aplicaciones externo. Utiliza las imágenes de la cámara y la información del LiDAR del vehículo y, mediante un procesado dividido en tres capas (percepción, planificación y control) devuelve los parámetros de control de giro y aceleración.

Respecto a la red móvil, se ha realizado un análisis de la arquitectura de una red 4G y 5G NSA y de diversos métodos para la implementación de una red 4G mediante SDR. Se ha implementado la red con algunos de estos métodos, como son la USRP NI-2901 y la BladeRF 2.0 micro xA9 como dispositivo SDR, srseNB como eNodeB, y srseEPC y Open5GS como núcleo de la red. Para cada opción, se ha documentado su instalación y configuración para su correcto funcionamiento.

Finalmente, se ha evaluado la estabilidad del algoritmo de guiado autónomo en función de la latencia del sistema y se han realizado múltiples evaluaciones del caso de uso de ejecución del algoritmo de guiado fuera del vehículo a través de una red wifi y de nuestra propia red 4G. Se han obtenido múltiples parámetros de la red en cada caso y se ha concluido con que es posible realizar *off-loading* en la red a través de 4G con un servidor en la frontera de la red con retardos menores de 100ms.

Este Trabajo Final de Máster constituye un gran resumen de muchas de las competencias que he adquirido a lo largo de mi estancia en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universidad de Valladolid. El diseño de un algoritmo de conducción autónoma, la programación desde cero de unos vehículos para que puedan comunicarse a través de la red y la puesta en marcha y evaluación de mi propia red 4G completamente funcional ha sido una experiencia muy excitante y enriquecedora. Los numerosos retos a los que me he enfrentado durante el desarrollo de este trabajo me han permitido aprender a tener perseverancia, buscar alternativas y ver los resultados de una actitud proactiva; toda una experiencia que apreciaré en el futuro.

6.2. Líneas futuras

La principal línea futura de este Trabajo Final de Máster es llegar a **implementar una red 5G SA**, pudiendo evaluar las características de MEC que incluyen estas redes de forma nativa. Para ello, el método más compatible con todo el trabajo desarrollado en este trabajo consiste en aprovechar la última actualización del *software srsRAN* lanzada a primeros de Mayo de 2022 (versión 22.4). Esta versión permite la implementación de gNodesB con una configuración muy parecida a la implementación de eNodesB explicada en este trabajo. Por otro lado, el *software Open5GS* que se analiza y utiliza en este trabajo para implementar el núcleo de la red 4G, también ofrece los servicios el núcleo 5G SA, servicios con configuraciones similares a las explicadas aquí. Los análisis y las implementaciones de este Trabajo Final de Máster constituyen un gran resumen del *software* disponible para la implementación de redes móviles, y sientan las bases para continuar trabajando en la dirección del 5G, habiendo elegido el *software* con un desarrollo más activo en la actualidad. También se pueden probar otros softwares no probados en este Trabajo Final de Máster como OpenAirInterface.

Atendiendo a la **red 4G implementada**, sería interesante lograr **implementar el handover en la red 4G**. Esta tarea se intentó implementar en este trabajo pero, tal y como se explica en la sección 5.4, nos enfrentamos a una serie de dificultades que impidieron la correcta puesta en marcha de dicho caso. Asimismo, es de interés **implementar el MEC a través de la herramienta 5GEmpower** analizada en el estado del arte de este trabajo. Este MEC utilizaría el método *bump-in-the-wire*, método no evaluado en este trabajo.

Respecto al **algoritmo de conducción autónoma** del vehículo, se puede modificar su funcionamiento para que **utilice las líneas blancas laterales de las carreteras** en vez de una línea central coloreada, de esta forma su funcionamiento sería más fiel a la realidad. Además, se puede **añadir nueva sensórica a los vehículos**, como geolocalización y velocidad del vehículo. De esta forma, se podría optimizar el algoritmo, permitiendo un control longitudinal más preciso y una capa de planificación más compleja. Asimismo, estos parámetros permitirían **evaluar nuevos casos de uso** relacionados con el vehículo conectado para el banco de pruebas. Este trabajo de adición de nueva sensórica se está haciendo actualmente por el estudiante Iván Viloría Vazquez en su Trabajo Final de Máster. Por otro lado, el estudiante Íñigo Nieto Cuadrado en su Trabajo Final de Grado está evaluando un caso de uso relacionado con el reconocimiento de señales de tráfico en el MEC a través del banco de pruebas desarrollado en este trabajo.

7. Bibliografía

- [1] A. Asensio, X. Masip-Bruin, R. Duran, I. d. Miguel, G. Ren, S. Daijavad y A. Jukan, «Designing an efficient clustering strategy for combined Fog-to-Cloud scenarios,» de *Future Generation Computer Systems*, vol. 8 ed., 2020, pp. 302-406.
- [2] L. Galluccio, C. Grasso, M. Grasso, R. Raftopoulos y G. Schembra, «Measuring QoS and QoE for a Softwarized Video Surveillance System in a 5G Network,» 2019.
- [3] 3. T. 38.913, «Study on scenarios and requirements for next generation access technologies,» Julio 2020. [En línea].
- [4] T. Linget, «5GAA: Visionary Roadmap for Advanced Driving Use Cases, Connectivity Technologies, and Radio Spectrum Needs,» 5G Automotiva Association, Noviembre 2020. [En línea]. Available: <https://5gaa.org/wp-content/uploads/2020/09/A-Visionary-Roadmap-for-Advanced-Driving-Use-Cases-Connectivity-Technologies-and-Radio-Spectrum-Needs.pdf>. [Último acceso: 28 Febrero 2022].
- [5] T. Cohen y C. Cavoli, «Automated vehicles: exploring possible consequences of government (non)intervention for congestion and accesibility,» Septiembre 2018. [En línea]. Available: <https://www.tandfonline.com/doi/full/10.1080/01441647.2018.1524401>. [Último acceso: 28 Febrero 2022].
- [6] E. Coronado, G. Cebrián Marquez y R. Riggio, «Enabling Autonomous and Connected Vehicles at the 5G Network Edge,» [En línea]. Available: https://www.researchgate.net/publication/343622557_Enabling_Autonomous_and_Connected_Vehicles_at_the_5G_Network_Edge. [Último acceso: 28 Febrero 2022].
- [7] Intel, «Kit Intel® NUC 9 Extreme - NUC9i9QNX,» [En línea]. Available: <https://www.intel.es/content/www/es/es/products/sku/190107/intel-nuc-9-extreme-kit-nuc9i9qnx/specifications.html>. [Último acceso: 3 Marzo 2022].
- [8] National Instruments, «USRP-2901,» [En línea]. Available: <https://www.ni.com/es-es/support/model.usrp-2901.html>. [Último acceso: 3 Marzo 2022].
- [9] Nuand, «bladeRF2.0 micro xA9,» [En línea]. Available: <https://www.nuand.com/product/bladerf-xa9/>. [Último acceso: 4 Marzo 2022].
- [10] sysmocom, «sysmocom SIM / USIM / ISIM cards,» [En línea]. Available: <https://sysmocom.de/products/sim/sysmousim/index.html>. [Último acceso: 2 Mayo 2022].
- [11] Amazon, «AWS Deep Racer,» [En línea]. Available: <https://aws.amazon.com/es/deepracer/>.
- [12] Slamtec, «RPLIDAR AI,» [En línea]. Available: <https://www.slamtec.com/en/Lidar/AI>. [Último acceso: 4 Marzo 2022].
- [13] ITU-R, «Requirements related to technical performance for IMT-Advanced radio interface(s),» [En línea]. Available: <https://www.itu.int/pub/R-REP-M.2134-2008/en>. [Último acceso: 6 Marzo 2022].

- [14] M. Bakni, «TIA/EIA IS-3: TIA/EIA IS-3 Cellular System Mobile Station - Land Station Compatibility Specification, Cellular network standards and generation timeline,» [En línea]. Available: https://en.wikipedia.org/wiki/List_of_mobile_phone_generations#/media/File:Cellular_network_standards_and_generation_timeline.svg. [Último acceso: 15 Mayo 2022].
- [15] 3GPP, «LTE,» [En línea]. Available: <https://www.3gpp.org/technologies/keywords-acronyms/98-lte>. [Último acceso: 7 Marzo 2022].
- [16] IEEE, «IEEE 802.16e-2005,» [En línea]. Available: <https://standards.ieee.org/ieee/802.16e/3659/>. [Último acceso: 7 Marzo 2022].
- [17] Wiley Telecom, «LTE Introduction,» IEEE, 2013. [En línea]. Available: <https://ieeexplore.ieee.org/document/8042462>. [Último acceso: 7 Marzo 2022].
- [18] Qualcomm, «OnQ Blog: Understanding 3GPP - starting with the basics,» [En línea]. Available: <https://www.qualcomm.com/news/onq/2017/08/02/understanding-3gpp-starting-basics>. [Último acceso: 15 Mayo 2022].
- [19] J. Zyren, NXP, Freescale semiconductor, 2007. [En línea]. Available: <https://www.nxp.com/docs/en/white-paper/3GPPEVOLUTIONWP.pdf>. [Último acceso: 7 Marzo 2022].
- [20] Frédéric Firmin, 3GPP MCC, «The Evolved Packet Core,» [En línea]. Available: <https://www.3gpp.org/technologies/keywords-acronyms/100-the-evolved-packet-core>. [Último acceso: 7 Marzo 2022].
- [21] ITU, «<https://www.itu.int/es/ITU-T/focusgroups/imt-2020/Pages/default.aspx>,» 14 Marzo 2022. [En línea]. Available: <https://www.itu.int/es/ITU-T/focusgroups/imt-2020/Pages/default.aspx>. [Último acceso: 14 Marzo 2022].
- [22] L. Casaccia, «Qualcomm, Understanding 3GPP – starting with the basics,» 2 Agosto 2017. [En línea]. Available: <https://www.qualcomm.com/news/onq/2017/08/02/understanding-3gpp-starting-basics#:~:text=3GPP%20began%20work%20in%202016,in%20next%2Dgeneration%205G%20networks..> [Último acceso: 14 Marzo 2022].
- [23] Hocell, «5G NSA to SA Migration Paths,» [En línea]. Available: <https://www.hocell.com/newsinfo/529243.html>. [Último acceso: 15 Mayo 2022].
- [24] srsRAN, «srsRAN docs, srsRAN Application Notes, 5G NSA End-to-Edn,» Octubre 2021. [En línea]. Available: https://docs.srsran.com/en/latest/app_notes/source/5g_nsa_zmq/source/index.html. [Último acceso: 14 Marzo 2022].
- [25] RedHat docs, «What is multi-access edge computing (MEC)?,» 26 Marzo 2021. [En línea]. Available: <https://www.redhat.com/en/topics/edge-computing/what-is-multi-access-edge-computing>.
- [26] M. Patel, B. Naughton, C. Chan, N. Sprecher, S. Abeta y A. Neal, «Mobile-Edge Computing,» Septiembre 2014. [En línea]. Available:

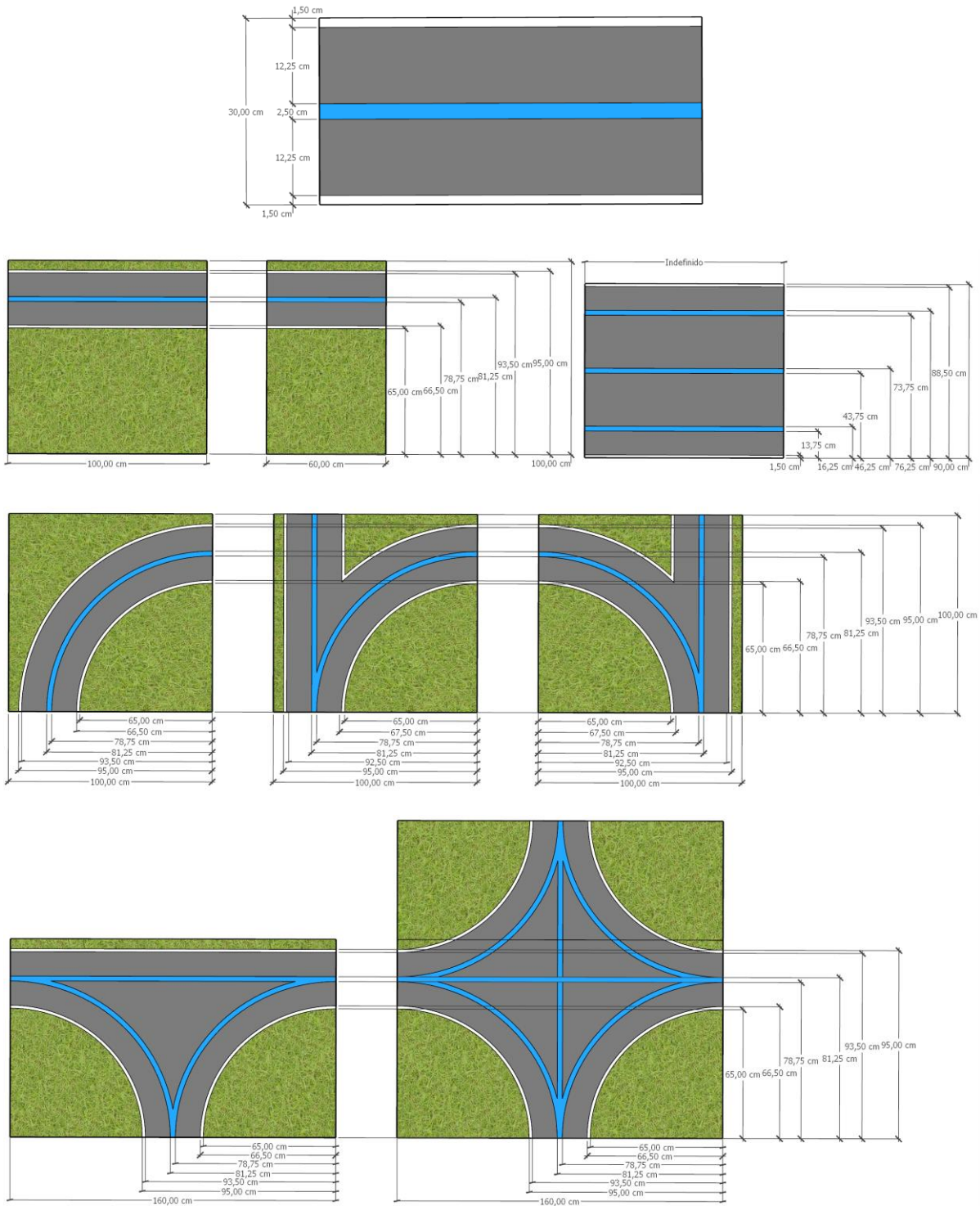
- https://portal.etsi.org/portals/0/tbpages/mec/docs/mobile-edge_computing_-_introductory_technical_white_paper_v1%2018-09-14.pdf. [Último acceso: 14 Marzo 2022].
- [27] ETSI, «MEC Deployments in 4G and Evolution Towards 5G.» Febrero 2018. [En línea]. Available: https://www.etsi.org/images/files/etsiwhitepapers/etsi_wp24_mec_deployment_in_4g_5g_final.pdf. [Último acceso: 14 Marzo 2022].
- [28] J. Wang, K. Han, C. Zhiyu, A. Alexandridis, Z. Ziliz, Y. Pang y J. Lin, «A Software Defined Radio Evaluation Platform for WBAN Systems,» 31 Octubre 2018. [En línea]. Available: <https://www.mdpi.com/1424-8220/18/12/4494>. [Último acceso: 17 Marzo 2022].
- [29] Open5GS, «Open5GS,» [En línea]. Available: <https://open5gs.org/>.
- [30] Open Air Interface, «Open Air Interface,» [En línea]. Available: <https://openairinterface.org>. [Último acceso: 19 Marzo 2022].
- [31] Open Air Interface, «5G Core Network,» [En línea]. Available: <https://openairinterface.org/oai-5g-core-network-project/>. [Último acceso: 21 Marzo 2022].
- [32] Open Air Interface, «OpenAirInterfaceTM (OAI): Towards Open Cellular Ecosystem,» [En línea]. Available: <https://openairinterface.org/getting-started/openairinterface-an-open-cellular-ecosystem/>. [Último acceso: 19 Marzo 2022].
- [33] Z. Geng, X. Wei, H. Liu, R. Xu y K. Zheng, «Performance Analysis and Comparison of GPP based SDR Systems,» [En línea]. Available: <https://ieeexplore.ieee.org/document/8250816>. [Último acceso: 20 Marzo 2022].
- [34] Open5GS, «Quickstart,» [En línea]. Available: <https://open5gs.org/open5gs/docs/guide/01-quickstart/>. [Último acceso: 19 Marzo 2022].
- [35] srsRAN, «srsRAN Features,» 2022. [En línea]. Available: https://docs.srsran.com/en/latest/feature_list.html. [Último acceso: 19 Marzo 2022].
- [36] C.-Y. Li, H.-Y. Liu, P.-H. Huang, H.-T. Chien, G.-H. Tu, P.-Y. Hong y Y.-D. Lin, «Mobile Edge Computing Platform Deployment in 4G LTE Networks: A Middlebox Approach,» *Proc. of USENIX HotEdge*, 2018.
- [37] J. Haavisto, M. Arif, L. Lovén, T. Leppänen y J. Riekkki, «Open-source RANs in Practice: an Over-The-Air Deployment for 5G MEC,» 2019. [En línea]. Available: <https://ieeexplore.ieee.org/document/8801973>. [Último acceso: 19 Marzo 2022].
- [38] E. Coronado, Z. Yousaf y R. Riggio, «LightEdge: Mapping the Evolution of Multi-access,» *IEEE Communications Magazine*, Abril 2022. [En línea]. Available: <https://www.researchgate.net/publication/340801665>. [Último acceso: 5 Junio 2022].
- [39] OpenCV, «OpenCV,» [En línea]. Available: <https://opencv.org/>. [Último acceso: 20 Marzo 2022].
- [40] 5G-emPOWER, «5G-emPOWER,» [En línea]. Available: <http://5g-empower.io/>. [Último acceso: 20 Marzo 2022].
- [41] «nextEPC,» [En línea]. Available: <https://nextepc.org/>. [Último acceso: 20 Marzo 2022].

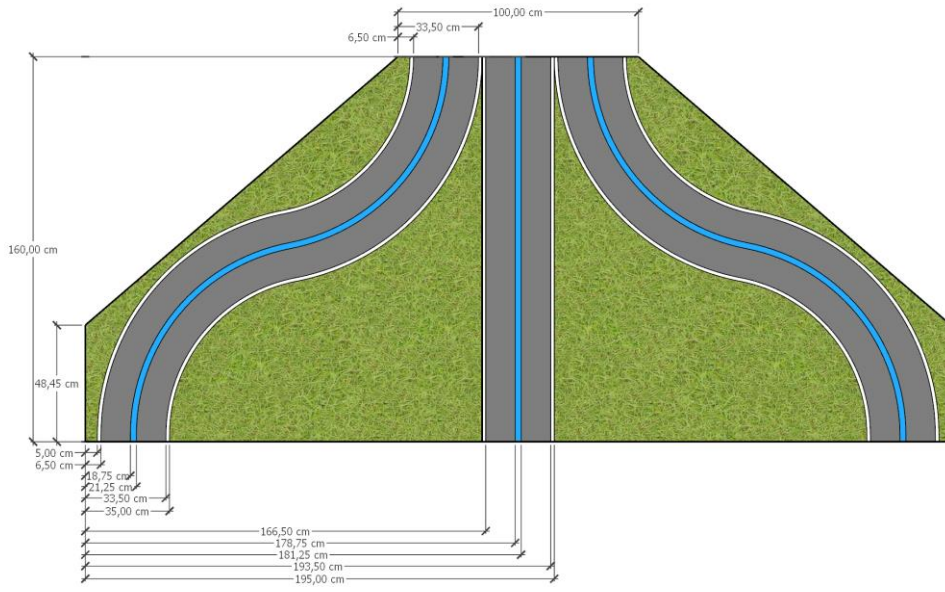
- [42] Bosch, «IMU: BMI160,» [En línea]. Available: <https://www.bosch-sensortec.com/products/motion-sensors/imus/bmi160/>. [Último acceso: 10 Abril 2022].
- [43] Huawei, «Huawei 4G Dongle E3372,» [En línea]. Available: <https://consumer.huawei.com/en/routers/e3372/specs/>. [Último acceso: 28 Marzo 2022].
- [44] ROS, «ROS Kinetic Kame,» 23 Mayo 2016. [En línea]. Available: <http://wiki.ros.org/kinetic>. [Último acceso: 28 Marzo 2022].
- [45] ROS, «ROS Command-line tools,» [En línea]. Available: <http://wiki.ros.org/ROS/CommandLineTools>. [Último acceso: 4 Abril 2022].
- [46] Python, «Python,» [En línea]. Available: <https://www.python.org/>. [Último acceso: 4 Abril 2022].
- [47] I. Royuela González, «GitHub, Artemis,» Grupo de Comunicaciones Ópticas, [En línea]. Available: <https://github.com/gcoUVA/Artemis>. [Último acceso: 14 Abril 2022].
- [48] MQTT, «MQTT: The Standard for IoT Messaging,» [En línea]. Available: <https://mqtt.org/>. [Último acceso: 10 Abril 2022].
- [49] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt y P. Stang, «Stanley: The Robot that Won the DARPA Grand Challenge,» Journal of Field Robotics, Wiley Periodicals, 27 Junio 2006. [En línea]. Available: <http://robots.stanford.edu/papers/thrun.stanley05.pdf>. [Último acceso: 15 Abril 2022].
- [50] ITU, «International Numbering Resources Database,» [En línea]. Available: <https://www.itu.int/net/ITU-T/inrdb/>. [Último acceso: 2 Mayo 2022].
- [51] Nuand, «GitHub BladeRF,» [En línea]. Available: <https://github.com/Nuand/bladeRF>. [Último acceso: 1 Mayo 2022].
- [52] Nuand, «GitHub Wiki BladeRF,» [En línea]. Available: <https://github.com/Nuand/bladeRF/wiki>. [Último acceso: 1 Mayo 2022].
- [53] ETTUS, «USRP Hardware Driver and USRP Manual,» [En línea]. Available: https://files.ettus.com/manual/page_install.html. [Último acceso: 1 Mayo 2022].
- [54] srsRAN, «srsRAN Docs: Installation Guide,» [En línea]. Available: https://docs.srsran.com/en/latest/general/source/1_installation.html#package-installation. [Último acceso: 2 Mayo 2020].
- [55] Share Tech Note, «LTE Quick Reference: SIB (System Information Block),» [En línea]. Available: https://www.sharetechnote.com/html/Handbook_LTE_SIB.html. [Último acceso: 2 Mayo 2022].
- [56] ScienceDirect, «Physical Resource Block,» [En línea]. Available: <https://www.sciencedirect.com/topics/computer-science/physical-resource-block>. [Último acceso: 7 Mayo 2022].
- [57] Kaitotek, «Qosium,» [En línea]. Available: <https://www.kaitotek.com/qosium>. [Último acceso: 21 Mayo 2022].

- [58] Kaitotek, «Kaitotek,» [En línea]. Available: <https://www.kaitotek.com/>. [Último acceso: 21 Mayo 2022].
- [59] INCIBE, «NTP, SNTP y PTP: ¿qué sincronización de tiempo necesito?,» 5 Marzo 2020. [En línea]. Available: <https://www.incibe-cert.es/blog/ntp-sntp-y-ntp-sincronizacion-tiempo-necesito>. [Último acceso: 22 mayo 2022].
- [60] Kaitotek, «Clock Synchronization in Linux,» [En línea]. Available: <https://www.kaitotek.com/resources/documentation/probe/install/debian-ubuntu/clock-synchronization-linux>. [Último acceso: 22 Mayo 2022].
- [61] Wireshark, «Wireshark,» [En línea]. Available: <https://www.wireshark.org/>. [Último acceso: 22 Mayo 2022].
- [62] srsRAN, «srsRAN docs, Application Notes, Intra eNB & SI Handover, SI Handover,» [En línea]. Available: https://docs.srsran.com/en/latest/app_notes/source/handover/source/index.html#si-handover. [Último acceso: 28 Mayo 2022].
- [63] Z. Akhundov, «A3 Intra-Frequency Handover in LTE,» Telecompedia, [En línea]. Available: <https://telecompedia.net/a3-intra-frequency-handover-in-lte/>. [Último acceso: 2022].
- [64] GitHub, «Issue #778: Downlink stops working when performing a SI handover,» [En línea]. Available: <https://github.com/srsran/srsRAN/issues/778>. [Último acceso: 28 Mayo 2022].
- [65] "sghong", «GitHub: "srsst",» [En línea]. Available: <https://github.com/sghong/srsst/tree/master>. [Último acceso: 13 Mayo 2022].
- [66] Open5GS, «Open5GS Quickstart,» [En línea]. Available: <https://open5gs.org/open5gs/docs/guide/01-quickstart/>. [Último acceso: 10 Mayo 2022].

Anexo I

En este anexo se incluye toda la colección de piezas utilizadas para la versión final de la maqueta. Todas las carreteras tienen una anchura total de 30cm que incluyen líneas blancas laterales de 1,5cm y una línea central de 2,5cm

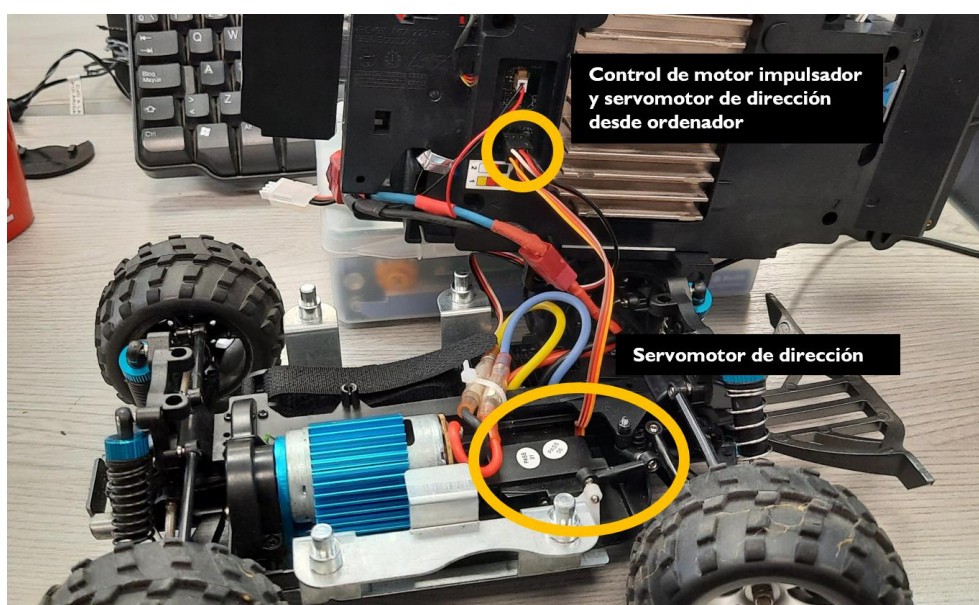




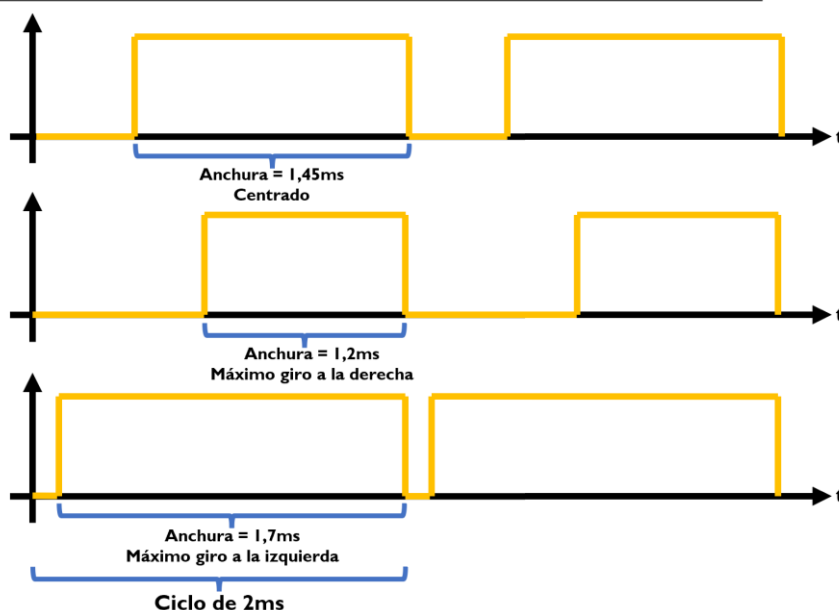
Anexo II

El ángulo de dirección del vehículo Amazon DeepRacer debe de ser calibrado. Cada vehículo, si bien viene con una calibración por defecto, debido a las tolerancias de fabricación puede no comportarse de forma óptima, haciendo que, por ejemplo, se desvíe hacia un lado cuando tuviera que mantenerse recto o no gire la dirección al máximo.

El giro del vehículo se realiza mediante un servomotor que permite un control preciso en términos de posición angular. Este servomotor es controlado por el ordenador del vehículo a través de unas señales PWM (*Pulse Width Modulation*), señal basada en una secuencia de pulsos donde, en función de la anchura de estos, el servomotor girará más hacia un lado o hacia otro. El ciclo de la modulación PWM es de 2ms de manera que la anchura del pulso deberá ser siempre menor que 2ms. Con los valores de calibración por defecto, el giro máximo hacia la izquierda se realiza cuando la anchura del pulso es de 1,7ms y el giro máximo a la derecha cuando la anchura del pulso es de 1,2ms.



Señal PWM de ordenador hacia servomotor de dirección



La calibración de la dirección se realiza mediante la configuración de la anchura de los pulsos PWM en el giro máximo a la derecha, a la izquierda y en el punto con las ruedas centradas. El nodo ROS `/servo_node` es el encargado de configurar las calibraciones. Mediante el tópico `/servo_cal` se le puede enviar un mensaje que incluye dichos parámetros de calibración. Estos parámetros se dan en forma de anchura del pulso de la señal PWM en nanosegundos:

- `max`: Máximo giro a la izquierda (1700000 nanosegundos de anchura de pulso por defecto)
- `mid`: Posición central del vehículo (1450000 nanosegundos de anchura de pulso por defecto)
- `min`: Máximo giro a la derecha (1200000 nanosegundos de anchura de pulso por defecto)

Técnicamente, una vez enviados los parámetros de calibración a través del tópico `/servo_cal`, el nodo debería guardarlos indefinidamente, disponiendo del vehículo calibrado aun reiniciando el vehículo. Sin embargo, se ha comprobado que no sucede esto. Es por ello que, el nodo `/admin_communications_node` guarda la calibración configurada en el fichero `vehicle.conf` y envía dicha calibración a través del tópico `/servo_cal` cada vez que se inicia.

Anexo III

A continuación, se muestra un ejemplo del fichero de configuración `vehicle.conf` que utiliza el *software* desarrollado para el vehículo Amazon DeepRacer en este Trabajo Final de Máster. El fichero se encuentra en formato INI y se divide en 3 secciones MQTT, aplicación de guiado en servidor y parámetros de calibración.

Fichero `vehicle.conf`:

```
[mqtt] # Parámetros relacionados con MQTT
vehicle_id = 1 # Identificador del vehículo
mqtt_server_ip = 192.168.0.100 # Dirección IP del servidor MQTT
mqtt_server_port = 1883 # Puerto del servidor MQTT

[cloud_autonomous_driving] # Parámetros relacionados con la aplicación de guiado en servidor
cloud_server_ip = 192.168.0.112 # Dirección IP del servicio de guiado autónomo
cloud_server_port = 20001 # Puerto del servicio de guiado autónomo

[steering_calibration] # Parámetros de calibración del vehículo
max = 1630000
mid = 1460000
min = 1180000
```

Anexo IV

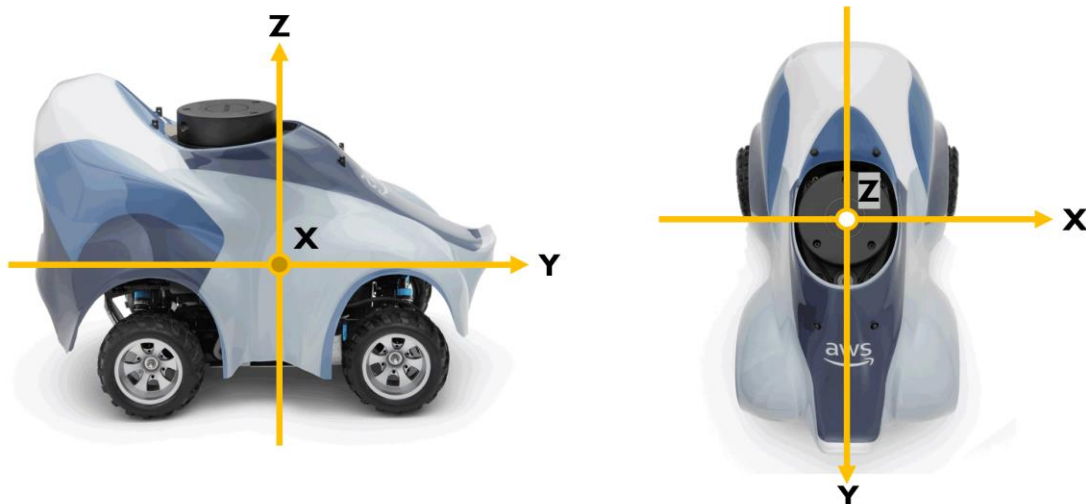
El *software* añadido al vehículo incluye dos clases auxiliares denominadas `Net_artemis` y `bmi160`. A continuación, se explican sus métodos principales.

La clase `Net_artemis` incluye una serie de métodos que permiten obtener diferentes parámetros de la red. Estos son los siguientes:

- `get_SSID_actua1()`: Devuelve el nombre **SSID de la red WiFi** a la que está conectado el ordenador del vehículo. Si no está conectado a ninguna red WiFi salta una excepción.
- `get_dbm_signal_actua1()`: Devuelve la **potencia de señal WiFi** que recibe el ordenador en dbms. Si no está conectado a ninguna red WiFi salta una excepción.
- `scan()`: Ordena al transceptor WiFi **realizar un escaneo de las redes disponibles** y devuelve una **lista con los SSID** e intensidad de la señal recibida de ellas.
- `get_publicIP()`: Devuelve dos parámetros.
 - El **tipo de conexión**, que puede ser 4G/5G en caso de tener conectado un módem o wifi en caso de estar conectado a internet a través del transceptor wifi.
 - La **IP pública** del dispositivo. Este parámetro permite obtener la IP que tiene el dispositivo una vez se traspasa un servicio NAT. En el caso de estar conectado por Wifi la IP obtenida será la de salida del *router* Wifi hacia internet. En el caso de utilizar un módem 4G/5G que implemente NAT, será la IP que ha asignado la estación base al módem.

La clase `bmi160` permite comunicarse con la IMU de Bosh BMI 160 [42] que incorpora el vehículo. Esta clase fue obtenida del repositorio de GitHub “snsst” del usuario “sghong” [65]. Consiste en un driver de bajo nivel que trabaja directamente escribiendo y leyendo de los registros de la IMU a través de I2C. Incluye una serie de métodos para realizar tareas de calibrado y obtención de parámetros. Las más relevantes son las siguientes:

- `enable_acce1()`: Habilita las medidas de aceleración en la IMU escribiendo en los registros correspondientes a través de I2C.
- `read_acce1()`: Devuelve los valores de aceleración en m^2/s de los 3 ejes de coordenadas (X, Y, Z). En condiciones normales, con el vehículo detenido y nivelado, tenemos una aceleración nula en los ejes X e Y, y una aceleración de $-9.8 m^2/s$ en el eje Z. A continuación, se muestra la distribución de los ejes en el vehículo.



Anexo V

Bandas utilizadas en España												
Banda	ULmin	ULmax	ULcentral	DLmin	DLmax	DLcentral	BW	E/U/ARFCN	Estado	Licitación	Observación / Potencia RX	
Banda 20	832	862		791	821		30		Ocupada		Dividendo digital, utilizada hasta 2015 para TDT. Se utiliza exclusivamente para LTE.	
	832	842	837	791	801	796	10	6200	Ocupada	Orange	-8,1 dBm	
	842	852	847	801	811	806	10	6300	Ocupada	Vodafone	-14,2 dBm	
	852	862	857	811	821	816	10	6400	Ocupada	Movistar	-5,1 dBm	
Banda 8	880	915		925	960		35		Ocupada		Utilizada para GSM y 3G. Utilizada para 4G en entornos rurales.	
	880,1	885,1	882,6	925,1	930,1	927,6	5		Ocupada	Orange	No 4G Visto con analizador	
	885,1	890,1	887,6	930,1	935,1	932,6	5		Ocupada	Orange	No 4G Visto con analizador	
	890,1	900,1	895,1	935,1	940,1	937,6	5		Ocupada	Movistar	No 4G Visto con analizador	
	895,1	900,1	897,6	940,1	945,1	942,6	5		Ocupada	Movistar	No 4G Visto con analizador	
	900,1	904,9	902,5	945,1	949,9	947,5	4,8		Ocupada	Movistar	No 4G Visto con analizador	
Banda 3	904,9	914,9	909,9	949,9	959,9	954,9	10		Ocupada	Vodafone	No 4G Visto con analizador	
	1710	1785		1805	1880		75		Ocupada			
	1710,1	1730,1	1720,1	1805,1	1825,1	1815,1	20		Ocupada	Movistar	-24.5 dBm	
	1730,1	1750,1	1740,1	1825,1	1845,1	1835,1	20	1501	Ocupada	Vodafone	-28,7 dBm	
	1750,1	1760,1	1755,1	1845,1	1855,1	1850,1	10	1651	Ocupada	Yoigo	-22.2 dBm	
	1760,1	1764,9	1762,5	1855,1	1859,9	1857,5	4,8		Ocupada	Yoigo	-22.2 dBm	
Banda 7	1764,9	1784,9	1774,9	1859,9	1879,9	1869,9	20	1849	Ocupada	Orange	-24.4 dBm	
	2500	2570		2620	2690	2655	70		Parcialmente ocupada			
	2500	2520	2510	2620	2640	2630	20	2850	Ocupada	Movistar	-8.8 dBm	
	2520	2540	2530	2640	2660	2650	20	3050	Ocupada lejos	Orange	-34.5 dBm	
	2540	2550	2545	2660	2670	2665	10	3200	Libre	Vodafone		
	2545	2550	2547,5	2670	2675	2672,5	5	3275	Libre	Vodafone		
Banda 1	2560	2570	2565	2680	2690	2685	10	3400	Ocupada lejos	Orange	-45.4 dBm	
	1920	1935	1927,5	2110	2170	2140	60					
	1920	1925	1922,5	2110	2115	2112,5	5		Ocupada	Yoigo	-19.5 dBm	
	1925	1935	1930	2115	2125	2120	10		Ocupada	Yoigo	-19.5 dBm	
	1935	1950	1942,5	2125	2140	2132,5	15		Ocupada lejos	Orange	-31.4 dBm	
	1950	1965	1957,5	2140	2155	2147,5	15		Ocupada	Vodafone	-22.2dBm	
Banda 42	1965	1960	1962,5	2155	2160	2157,5	5		Ocupada	Movistar	-17.2 dBm	
	1970	1980	1975	2160	2170	2165	10		Ocupada	Movistar	-17.2 dBm	
Banda 42	3500	3600	3550	3400	3500	3450	100	41690	Libre		Utilizada para accesos a internet vía WiMAX	

Bandas utilizadas en otros países										
Banda	ULmin	ULmax	ULcentral	DLmin	DLmax	DLcentral	BW	Estado	País en el que se utiliza	Observaciones
Banda 4	1710	1755	1732,5	2110	2155		45	Ocupada		Se superpone a las bandas 1 y 3
Banda 5	824	849	836,5	869	894	881,5	25	Ocupada	Corea e Israel	Se superpone a la banda 20
Banda 6										Obsoleta
Banda 9	1749,9	1784,9	1767,4	1844,9	1879,9	1862,4	35	Ocupada	Japón	Se superpone a la banda 3
								Parcialmente ocupada		En España banda reservada para ofrecer Servicio Fijo (SF). Enlaces Punto a Punto, enlaces móviles de TV, enlaces punto multipunto, etc (UN-46). También para ministerio de defensa, radiodifusión (UN-154) (Detectada portadora en 1480MHz)
Banda 11	1427,9	1447,9	1437,9	1475,9	1495,9	1485,9	20		Japón	
Banda 12	699	716	707,5	729	746	737,5	17	Ocupada	Estados Unidos y Canadá	
Banda 13	777	787	782	746	756	751	10	Ocupada	Estados Unidos, Canadá y Bolivia	En España: Banda 700MHz compartida para TDT y 5G
Banda 14	788	798	793	758	768	763	10	Ocupada	No utilizada	En España: Banda 700MHz compartida para TDT y 5G
Banda 15										Obsoleta
Banda 16										Obsoleta
Banda 17	704	716	710	734	746	740	12	Ocupada	Estados Unidos, Canadá y Bolivia	En España: Banda 700MHz compartida para TDT y 5G
Banda 18	815	830	822,5	860	875	867,5	15	Ocupada	Japón	Se superpone a la banda 20
Banda 19	830	845	837,5	875	890	882,5	15	Ocupada	Japón	Se superpone a la banda 20
								Parcialmente ocupada		En España: UN-46 (Servicio Fijo, Radiodifusión), UN-154 (Radares para sondeo de suelos y paredes, baja potencia)
Banda 21	1447,9	1462,9	1455,4	1495,9	1510,9	1503,4	15		Japón	
Banda 22	3410	3490	3450	3510	3590	3550	80	Libre		Se superpone a la banda 42
Banda 23	2000	2020	2010	2180	2200	2190	20	Libre		En España: UN-48 (PMSE y servicios móviles por satélite), UN-154 (Radares para sondeo de suelos y paredes) y 3G
								Parcialmente ocupada		En España: Móvil por satélite, Exploración de la tierra por satélite, servicio fijo, Banda de socorro (1645,5-1646,5), UN-154 (Radares para sondeo de suelos y paredes, baja potencia)
Banda 24	1626,5	1660,5	1643,5	1525	1559	1542	34		Estados Unidos	
Banda 25	1850	1915	1882,5	1930	1995	1962,5	65	Ocupada	Estados Unidos	Se superpone a la banda 3
Banda 26	814	849	831,5	859	894	876,5	35	Ocupada	Estados Unidos	Se superpone a la banda 20
Banda 27	807	824	815,5	852	869	860,5	17	Ocupada		Se superpone a la banda 20
Banda 28	703	748	725,5	758	803	780,5	45	Ocupada	Australia, Japón, Francia, Nueva Zelanda, Panamá, Nueva Guinea y Taiwan	En España: Banda 700MHz compartida para TDT y 5G
Banda 29										No compatible
Banda 30	2305	2315	2310	2350	2360	2355	10	Libre	Estados Unidos	Radio enlaces móviles de televisión (ENG)
								Parcialmente ocupada		Protección pública y operaciones de socorro en caso de catástrofe (PPDR)
Banda 31	452,5	457,5	455	462,5	467,5	465	5			
								Ocupada		Se superpone en parte con banda 1, el resto en España: Móvil por satélite, Exploración de la tierra por satélite, servicio fijo, Banda de socorro (1645,5-1646,5), UN-154 (Radares para sondeo de suelos y paredes, baja potencia)
Banda 65	1920	2010	1965	2110	2200	2155	90			
								Ocupada		Se superpone en parte con banda 1, el resto en España: Móvil por satélite, Exploración de la tierra por satélite, servicio fijo, Banda de socorro (1645,5-1646,5), UN-154 (Radares para sondeo de suelos y paredes, baja potencia)
Banda 66	1710	1780	1745	2110	2200	2155	90		Estados Unidos y Canadá	