UNIVERSIDAD DE VALLADOLID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

GRADO EN TECNOLOGÍAS ESPECÍFICAS DE TELECOMUNICACIÓN
MENCIÓN EN SISTEMAS DE TELECOMUNICACIÓN

Arquitecturas LTE-5G mediante SDR y OpenAirInterface

Autor:

D. Íñigo Nieto Cuadrado

Tutor:

Dr. D. Juan Carlos Aguado Manzano

Valladolid, 26 de agosto 2022

TÍTULO:	**Arquitecturas LTE-5G mediante SDR y OpenAirInterface**
AUTOR:	**D. Íñigo Nieto Cuadrado**
TUTOR:	**Dr. D. Juan Carlos Aguado Manzano**
DEPARTAMENTO:	**Teoría de la Señal y Comunicaciones e Ingeniería Telemática**

TRIBUNAL	
PRESIDENTE:	**Dr. D. Ignacio de Miguel Jiménez**
VOCAL:	**Dr. D. Ramón José Durán Barroso**
SECRETARIO	**Dr. D. Juan Carlos Aguado Manzano**
FECHA:	
CALIFICACIÓN:	

Resumen

El propósito de este proyecto es comprobar todas las funcionalidades y mejoras sobre otros *softwares* de OpenAirInterface. Esta herramienta de código abierto permite establecer redes móviles 4G y 5G-NSA.

Durante el trabajo de fin de grado se realizó una investigación sobre cómo está construido el estándar sobre el que se apoya 4G, LTE, así como del que lo hace con 5G. Después se comparó el resultado con el del *software* srsRAN, utilizado por el grupo anteriormente. Tras muchas pruebas, concluimos que para obtener un rendimiento óptimo de OAI se requiere un afinamiento muy costoso de los parámetros que lo controlan, frente a la alternativa srsRAN que tiene mejor rendimiento y estabilidad sin necesidad de afinar.

Como caso de uso particular se construyó una maqueta con un circuito para un AWS DeepRacer, un coche que implementa conducción autónoma. Se ha desarrollado el código para comunicar el vehículo con la estación OAI.

Finalmente, se ha confeccionado un código para que el coche sea capaz de reconocer señales de tráfico en tiempo real gracias al algoritmo de inteligencia artificial SVM.

Palabras clave

AWS DeepRacer, conducción autónoma, estándar, inteligencia artificial, LTE, OAI, srsRAN, SVM, 4G, 5G-NSA.

Abstract

The goal of this project is to check all the features and improvements of OpenAirInterface over other software. This open-source tool allows to deploy 4G and 5G-NSA mobile networks.

While this Project was being carried out, an investigation about how is built the standard that underpins 4G, LTE, as well as the one that underpins 5G. Then, the results were compared with the ones of srsRAN, previously used by the group. After several tests, we concluded that getting a high performance of OAI requires a complex tunning process, contrary to srsRAN which gives good performance without any tunning.

As a particular case of use, a circuit was built for an AWS DeepRacer, a car that implements autonomous driving. The code for the communication between the vehicle and the network was written as well.

Eventually, a code has been developed to enable the car to recognize traffic signs in real time, thanks to artificial intelligence algorithm SVM.

Keywords

Autonomous driving, artificial intelligence, AWS DeepRacer, LTE, OAI, srsRAN, standard, SVM, 4G, 5G-NSA.

Agradecimientos

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 VA231P20), 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

Índice 1
Índice de Figuras 3
Glosario de Siglas 5
Capítulo 1. Introducción 1
1.1. Antecedentes y motivaciones 1
1.2. Objetivos 4
1.3. Fases del proyecto 4
1.4. Recursos 5
1.5. Estructura de la memoria 6
Capítulo 2. Estado del arte 7
2.1 Redes móviles 5G 7
2.1.1 Arquitectura de las redes 5G-NSA 7
2.1.2 Arquitectura de las redes 5G-SA 9
2.1.3 Paradigmas para la implementación del *EDGE Computing* 13
2.2 Algoritmos de clasificación basados en Machine Learning 15
2.2.1 Decision Tree 16
2.2.2 Naïve Bayesian 17
2.2.3 K – Nearest Neighbors 17
2.2.4 Support Vector Machines 17
2.2.5 Elección del algoritmo 23
Capítulo 3. Estudio de OpenAirInterface 24
3.1 Sobre OpenAirInterface 24
3.2 Escenarios de implementación 24
3.2.1 Arquitectura implementada 25
3.3 Preparación de las máquinas 28
3.3.1 Clonación del repositorio 29
3.4 Instalación del CN 30
3.4.1 Instalación de la base de datos 30
3.4.2 Instalación de los Snaps 31

3.4.3 Configuración de HSS 31
3.4.4 Configuración de MME 33
3.4.5 Configuración de SPGWC y SPGWU 35
3.5 Instalación del eNB 36
3.5.1 Instalación de los *drivers* de BladeRF 36
3.6 Instalación del gNB 39
3.7 Conectar un UE comercial 40
3.7.1 Configuración de red del CN 43
3.7.2 Optimización del enlace 45
3.8 Resultados 46
3.8.1 Pruebas con 5G-NSA 46
3.8.2 Monitorización de los eNB con T Tracer 46
3.8.3 Osciloscopio Soft-Scope 48
3.8.4 Test de velocidad 51
4. Reconocimiento de señales 53
4.1 Introducción 53
4.2 Implementación del algoritmo 54
4.2.1 Estructura del *software* original 54
4.2.2 Modificaciones realizadas 55
4.3 Pruebas realizadas 60
4.4 Resultados 61
4.4.1 Medidas de precisión 61
4.4.2 Resultados generales 63
Capítulo 5. Conclusiones y líneas futuras 64
5.1 Red móvil utilizando OpenAirInterface 64
5.2 Reconocimiento de señales 65
Referencias 68

Índice de Figuras

Figura 1. Evolución de la demanda de tráfico [3] 1
Figura 2. Representación del paradigma Edge Computing [8] 2
Figura 3. AWS DeepRacer EVO [10] 3
Figura 4. Arquitectura red 5G-NSA [19] 8
Figura 5. Diagrama de los servicios de una red 5G-SA [23] 10
Figura 6. Arquitectura 5G con un UE y sin roaming [23] 11
Figura 7. Arquitectura 5G para múltiples sesiones PDU [23] 12
Figura 8. Arquitectura 5G habilitando roaming [23] 13
Figura 9. Representación de Fog Computing 14
Figura 10. Diferencia entre Machine Learning y Deep Learning [33] 16
Figura 11. Aplicación del procesado HOG [36] 18
Figura 12. Detección de bordes con el operador de Sobel [39] 20
Figura 13. (a) Imagen dividida en celdas. (b) Histograma de cada celda [35] ... 20
Figura 14. Ejemplo de separación de dos nubes de puntos con SVM [42] 21
Figura 15. Implementación de soft-margin [41] 22
Figura 16. Comparativa entre algoritmos [46] 23
Figura 17. Diagrama de la red 26
Figura 18. Diagrama de la red propuesta por OAI 27
Figura 19. Implementación real de la red 28
Figura 20. Información de la base de datos 42
Figura 21. Diagrama de la red 43
Figura 22. Pila de protocolos en la interfaz radio [57] 47
Figura 23. Vista de la herramienta T Tracer 48
Figura 24. Interfaz gráfica soft-scope 49
Figura 25. LLR para 16 QAM y 64 QAM [60] 50
Figura 26. Test de velocidad 52
Figura 27. Arquitectura propuesta en el TFM de I. Royuela [9] 54
Figura 28. Ejemplo de reconocimiento en el vídeo de ejemplo 55
Figura 29. Ejemplo del dataset original 56
Figura 30. Señales empleadas en la maqueta 57
Figura 31. Cálculo del tiempo de entrenamiento 58

Figura 32. Función de entrenamiento del modelo 59
Figura 33. Cálculo de los contornos 60
Figura 34. a) Imagen binaria b) Resultado final 60
Figura 35. Maqueta fabricada 61
Figura 36. Arquitectura propuesta para la maqueta final..... 61
Figura 37. Matriz de confusión..... 62
Figura 38. Cálculo de la precisión 62
Figura 39. Medición del tiempo de reconocimiento de una señal 63

Glosario de Siglas

3GPP. 3rd Generation Partnership Project
4G. Cuarta Generación de telefonía móvil
5G. Quinta generación de telefonía móvil
AGC. Automatic Gain Control
BW. Bandwidth
CN. Core Network
DL. Deep Learning
DL. Downlink
eNB. Evolved Node B
Gbps. Gigabits por segundo
GCO. Grupo de Comunicaciones Ópticas
GTP. GPRS Tunneling Protocol
HOG. Histogram of Gradients
HSS. Home Subscriber Server
IOT. Internet of Things
LTE. Long Term Evolution
Mbps. Megabits por Segundo
MEC. Multi Edge Computing
MIMO. Multiple Input, Multiple Output
ML. Machine Learning
MME. Mobility Management Entity
MQTT. MQ Telemetry Transport
NR. New Radio
OAI. OpenAirInterface
PDN. Packet Data Network
PGW. PDN Gateway
PLMN. Public Land Mobile Network
PRB. Physical Resource Block
QoS. Quality of Service
RBF. Radial Basis Function
SDR. Software Defined Radio
SGW. Serving Gateway
SVM. Support Vector Machines
TAC. Tracking Area Code
UE. User Equipment
UL. Uplink

Capítulo 1. Introducción

1.1. Antecedentes y motivaciones

Las primeras redes que se pueden considerar de tercera generación (3G) fueron lanzadas en 2002 por el 3GPP (3rd Generation Partnership Project). A finales de 2005 había 100 redes activas en todo el mundo, pero este número creció de manera exponencial en pocos años [1]. Tanto fue así que, en Japón, a mediados del 2008, había 83 millones de suscriptores 3G, suponiendo un 80% de todos los abonados a redes móviles [2].

Pero no solo aumentó el número de usuarios, sino que también lo hizo la demanda de datos de estos abonados. Con la mejora de las redes móviles los abonados comenzaron a compartir contenido multimedia, el cual es mucho más pesado que los mensajes de texto. La Figura 1muestra la demanda de tráfico en TB soportada por cada generación de telefonía móvil en España desde 2012, según datos del CNMC (Comisión Nacional de los Mercados y la Competencia). Se puede observar que la demanda aumenta de manera exponencial, llegando a sobrepasar los 3 millones de TB en 2020.

Figura 1. Evolución de la demanda de tráfico [3]

Aunque el crecimiento tan grande de 2020 se deba, principalmente, a la pandemia de Covid-19, se prevé que esta tendencia siga al alza durante los próximos años. Una de las grandes causas de ese incremento de la demanda en redes móviles es el

IoT (*Internet of Things*). Este nuevo paradigma tiene por objetivo dotar a múltiples objetos de "inteligencia", agregándoles conexión a Internet. Un ejemplo de aplicación de IoT son las pulseras inteligentes, las cuales miden constantemente el ritmo cardiaco de una persona. De esta manera, son capaces de detectar posibles problemas cardiacos.

Resulta lógico pensar que un gran aumento de los usuarios conectados provoca una sobrecarga en la red. El ancho de banda de las redes tradicionales es demasiado limitado para dar servicio a todos los nuevos UE (*User Equipment*) [4]. Una de las soluciones más inmediatas a este problema consiste en minimizar el consumo de ancho de banda de los dispositivos móviles, como lo hacen RFID o ZigBee [5].

Sin embargo, una solución más innovadora a este problema es el *Edge Computing*. Este paradigma se opone al tradicional *Cloud Computing*, el cual propone un sistema centralizado, en el que los UEs delegan toda su carga computacional y almacenamiento a servidores en la nube [6]. En cambio, *Edge Computing* propone acercar los servidores a los terminales de usuarios. Una de las grandes ventajas de este nuevo paradigma es la disminución de la latencia, la cual es provocada por la comunicación entre el UE y el servidor *cloud* [7].

En la Figura 2 se muestra la representación de una red. En ella, se coloca al servidor de *cloud* en el centro, y a los dispositivos móviles a su alrededor. Al disponerse en forma circular, cobra sentido el nombre de *Edge Computing*, ya que sitúa los UEs en la frontera de la red.

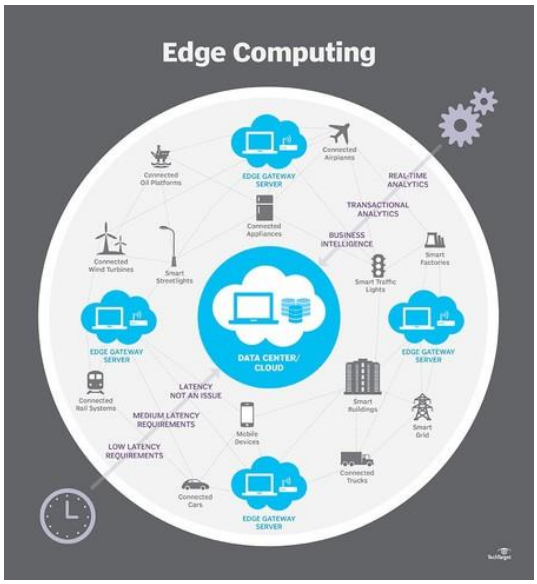

Figura 2. Representación del paradigma Edge Computing [8]

Este trabajo forma parte del proyecto de investigación ARTEMIS (Arquitectura *Edge* para el Soporte de Vehículos Conectados en Entornos Metropolitanos – VA231P20), proyecto concedido al Grupo de Comunicaciones Ópticas por la Junta de Castilla y León. Entre los objetivos que se plantea este proyecto se encuentra el de explorar el uso del MEC (*Multi-Access Edge Computing*) en aplicaciones de vehículos conectados en entornos metropolitanos. Uno de sus paquetes de trabajo tiene por objetivo la construcción de un entorno de pruebas donde probar y demostrar el uso de esta tecnología. Para ello se ha construido una maqueta con un circuito de carreteras por el que un vehículo de control remoto Amazon AWS DeepRacer Evo puede circular [9]. Dentro del Trabajo Fin de Máster de Ignacio Royuela [9], se ha desarrollado un *software* de control de dicho vehículo, *software* que permite la comunicación de datos con la red móvil y la ejecución de aplicaciones en ambos lados, tanto en la parte del vehículo como en la parte de la red. Concretamente, el vehículo envía a la red información recogida por sus sensores, particularmente de la cámara de video, y todos los cálculos que el coche necesita realizar para determinar su ruta son llevados a cabo en un ordenador situado en algún punto de la arquitectura de la red móvil, actuando de esta forma como un servidor *Edge*. La comunicación móvil en el trabajo de Ignacio Royuela se realiza fundamentalmente a través de la implementación de una red LTE con el *software* srsRAN.

Figura 3. AWS DeepRacer EVO [10]

Sin embargo, el *software* utilizado por Ignacio Royuela tiene una serie de limitaciones, de entre las cuales destaca la imposibilidad de conectar más de una

estación base a un mismo núcleo de red. Es por eso que este trabajo trata de explorar OpenAirInterface, otro *software* que, en teoría, suple las limitaciones de srsRAN.

1.2. Objetivos

Como ya se ha mencionado, este trabajo forma parte de un proyecto más amplio en el que otros compañeros han contribuido o están contribuyendo en su desarrollo. El objetivo principal de este trabajo fin de grado es desplegar una red 4G/LTE y una red 5G mediante el uso del *software* OpenAirInterface, un código *open-source* ampliamente utilizado en la literatura y que podría tener ventajas importantes sobre otros *softwares* ya probados como srsRAN.

Además, se establece un objetivo secundario relacionado con el proyecto ARTEMIS. Dicho proyecto pretende la demostración de casos de uso de vehículos conectados mediante tecnología inalámbrica y *edge computing*. Por ello, para el desarrollo de este Trabajo Final de Grado se persigue el desarrollo de una aplicación basada en inteligencia artificial para el reconocimiento de señales de tráfico en tiempo real utilizando la tecnología *edge*.

Para conseguir estos dos objetivos principales se establecieron una serie de objetivos secundarios:

- Puesta en marcha de la SDR (*Software Defined Radio)*, sintonizando los parámetros correctamente
- Probar la tecnología 5G
- Reconocimiento de señales en escenarios controlados
- Estudiar cómo implementar señales de tráfico luminosas

1.3. Fases del proyecto

Para la consecución de todos los objetivos planteados se dividió el proyecto en diferentes fases. La fase inicial consistió en un estudio teórico acerca de las redes móviles, enfocado en la arquitectura de los estándares LTE y NR (New Radio), los cuales rigen las tecnologías 4G y 5G, respectivamente. Además, también fue necesario conocer el trabajo que había realizado mi compañero Ignacio Royuela con el *software*

srsRAN [9], para afianzar conceptos y conocer los problemas típicos de este tipo de proyectos, como el manejo de la SDR.

La segunda fase del proyecto consistió en el estudio y despliegue del *software* OpenAirInteface (OAI), pasando a un nivel práctico el estudio teórico sobre redes 4G y 5G realizado anteriormente. Esta segunda fase requirió el estudio del *hardware* necesario, qué tipo y versión del sistema operativo soportaba el *software*.

Posteriormente se pasó a la fase de mejora del rendimiento de OAI. El rendimiento de este *software* es muy dependiente de la configuración de los parámetros que afectan al enlace radio.

Finalmente, la cuarta fase del proyecto consistió en el desarrollo de la aplicación para el vehículo conectado de la maqueta, AWS DeepRacer, para que fuera capaz de reconocer señales de tráfico en tiempo real mediante tecnología *edge*. Para ello fue necesario conocer el protocolo MQTT (*MQ Telemetry Transport*) ya que es quien gestiona la comunicación entre el servidor y el coche.

1.4. Recursos

Como ya se ha mencionado, para el desarrollo de este trabajo se ha necesitado el siguiente material:

- 6 Intel NUC BXNUC9I9QNX [11] 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.
- 6 BladeRF 2.0 micro xA9 de Nuand [12]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 sysmoISIM-SJA2 de la empresa Sysmocom [13].
- 4 vehículos Amazon Deep Racer con el pack de expansión Evo [14] 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 [15] de una capa con alcance de 12 metros.

- 4 módems Huawei E3372H-320 4G USB [16]. Actuarán como los UE de la red virtualizada. Además, al ser USB, se pueden utilizar para dar servicio a los AWS DeepRacer.

1.5. Estructura de la memoria

La presente memoria se estructura de la siguiente manera. Después del primer capítulo de introducción y motivación, el segundo capítulo contiene una revisión del estado del arte tanto de redes móviles como de algoritmos de clasificación basados en *machine learning*. El estado del arte de las redes móviles está centrado en las redes 5G, dado que las redes 4G han sido extensamente estudiadas en el trabajo de Ignacio Royuela [9]. El tercer capítulo reúne todo lo concerniente a OpenAirInterface, desde su instalación a su optimización. El capítulo 4 aborda la aplicación práctica relativa al reconocimiento de señales, incluyendo la instalación y entrenamiento del algoritmo de inteligencia artificial. Finalmente, en el último capítulo, se exponen las conclusiones y líneas futuras acerca del proyecto.

Capítulo 2. Estado del arte

En este capítulo se van a revisar todos los conceptos previos necesarios para la total comprensión del presente trabajo. Primero, se analizará el funcionamiento de las redes móviles quinta generación (5G). Después, se revisarán diferentes algoritmos de inteligencia artificial que se han barajado para realizar la tarea del reconocimiento de señales. Se incluye una comparativa entre los distintos algoritmos planteados.

2.1 Redes móviles 5G

La quinta generación de redes móviles surge como una evolución natural de las redes 4G. El aumento de la demanda, la necesidad de tasas de transmisión mayores y requerimientos más estrictos de retardo propiciaron el salto a una tecnología más avanzada.

En cuanto a la velocidad de transmisión, los 100 Mbps del 4G ascienden hasta 10 Gbps con 5G. Eso significa que es 100 veces superior a la generación anterior. Así se consigue mejorar la calidad de servicio (QoS) de los diferentes servicios. Esto hace posible transmitir volúmenes mucho mayores de datos, permitiendo, por ejemplo, transmitir vídeo en calidad 8K [17].

2.1.1 Arquitectura de las redes 5G-NSA

Las redes 5G son relativamente modernas, por lo que todavía estamos presenciando los comienzos de esta tecnología. En el año 2015 la ITU publicó los requisitos IMT-2020 (International Mobile Telecommunications – 2020) en los que se comenzó a definir lo que sería el estándar 5G NR [18] .

Las recomendaciones se pensaron para que el despliegue de la nueva tecnología fuera compatible durante un cierto periodo de transición con la tecnología LTE siempre que el operador lo considerara necesario. Esta modalidad de despliegue se denomina 5G-NSA (Non Stand Alone) y permiten ofrecer a los usuarios capacidades de radio con calidad 5G apoyándose en el sistema de servicios desarrollado para LTE. En la Figura 4 se puede observar una representación de una red 5G-NSA. En ella se encuentran tres

estratos bien diferenciados: el núcleo de la red (CN – Core Network), la red de acceso (RAN – Radio Access Network) y el lado de los usuarios (UE – User Equipment).

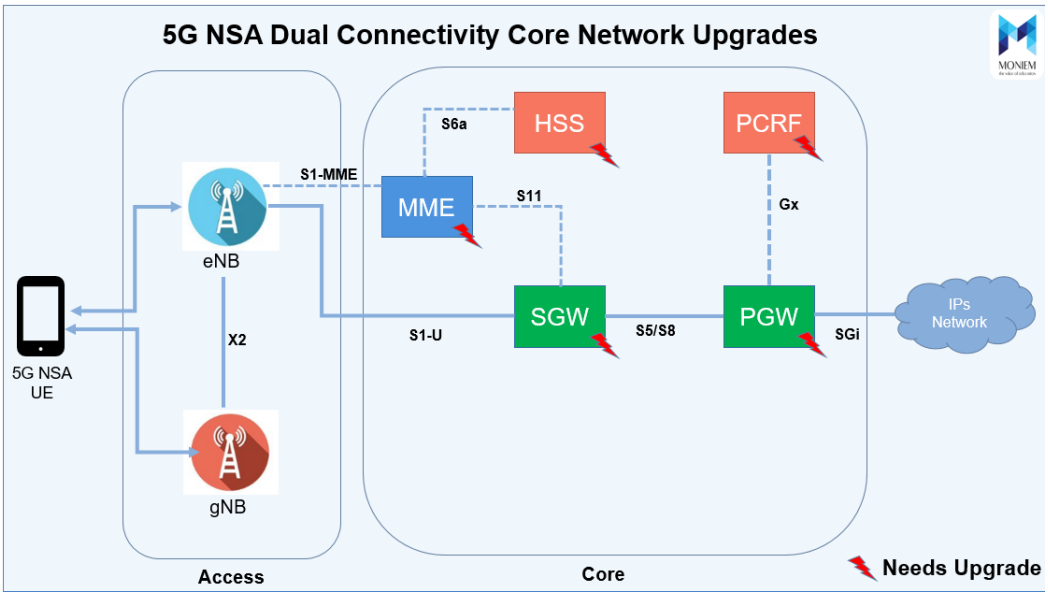

Figura 4. Arquitectura red 5G-NSA [19]

El CN de la figura pertenece a una red 4G – LTE. Este implementa todos los servicios necesarios para el correcto funcionamiento de la red. A continuación, se comenta la utilidad de cada uno de ellos:

- **HSS** (Home Subscriber Server) es el encargado de gestionar los datos de los usuarios. Tiene conexión directa con la base de datos. Genera los vectores de autenticación de la SIM.
- **MME** (Mobility Management Entity) es el servicio principal. Se encarga de coordinar al resto de servicios, así como de controlar los distintos UE que se conecten a la red.
- **SGW** (Serving Gateway) tramita el flujo de paquetes IP entre el CN y el UE. Actúa como un router IP e implementa encapsulado GTP.
- **PGW** (PDN Gateway) es, como se ve en la Figura 4, quien se encarga de la salida del tráfico a la red externa. Es también quien establece las QoS y el ancho de banda de la comunicación.

Además, se observa también en la Figura 4 que los diferentes servicios están intercomunicados por interfaces virtuales, como la S1 o la S6. Es importante conocer estas interfaces, ya que en el capítulo 3 se implementarán en el *software* OAI.

Entre el segmento de usuarios y el CN se encuentra la red de acceso E-UTRAN (Evolved UMTS Terrestrial Radio Access Network). En la Figura 4 se ve que está formada por un eNB (evolved Node B) y por un gNB (next-generation Node B). El estándar LTE define los eNB como estaciones base, y tienen la función de comunicar los UE con el CN, además de gestionar los recursos radio tanto en el enlace de bajada (DL, *Downlink*) como de subida (UL, *Uplink*) [20]. Su función principal es la de establecer un enlace virtual directo entre el UE y el CN utilizando el protocolo GTP (GPRS *Tunneling Protocol).*

Además, será importante en el desarrollo del trabajo conocer que el eNB puede trabajar con diferentes anchos de banda, según la situación lo requiera. En LTE y en 5G-NR se establece un número determinado de PRB (Physical Resource Block). Un PRB equivale a 12 subportadoras OFDM (Orthogonal Frequency Division Multeplexing) equiespaciadas por 15 KHZ. Por tanto, un PRB ocupa 180 KHz [21]. Existe, pues, una relación directa entre el número de PRB utilizados y el ancho de banda (BW) de la transmisión. En la Tabla 1 se recoge la relación entre valores estandarizados de PRB y su ancho de banda asociado en megahercios.

PRB	**6**	**15**	**25**	**50**	**75**	**100**
BW (MHz)	1.4	3	5	10	15	20

Tabla 1. Relación entre el número de PRB y el ancho de banda [21]

Finalmente, la idiosincrasia de las redes 5G-NSA reside precisamente en la segunda estación base gNB. Los gNB son las estaciones base que se definen para las redes 5G. En 5G-NSA un gNB se comunica directamente con el eNB a través de la interfaz X2. Obviamente, el rendimiento no es tan bueno como el de una red 5G SA (Stand Alone), pero mejora al de LTE. Las velocidades de descarga llegan a 2 Gbps y la latencia entre el UE y el CN puede llegar a solamente 10 milisegundos [22].

2.1.2 Arquitectura de las redes 5G-SA

Las redes 5G-SA no necesitan apoyarse en una estructura LTE para su correcto funcionamiento. El organismo responsable de la definición de la tecnología 5G es el 3GPP (3rd Generation Partnership Project), una colaboración de asociaciones de telecomunicaciones pertenecientes a la ETSI (European Telecommunications Standards Institute).

La arquitectura de las redes 5G-SA está definida en 3GPP TS 23.501. Las claves principales de la arquitectura 5G son [23]:

- Separar el plano de usuarios (UP) del plano de control (CP). Así se consigue una mayor escalabilidad, permitiendo despliegues flexibles adaptados a la necesidad de cada situación.
- Modularizar los servicios al máximo. El objetivo de separar todas las funciones en servicios independientes es el de implementar la tecnología *network slicing*, que consiste en fragmentar la red en redes lógicas virtuales (VLAN).
- Soportar acceso concurrente a servicios locales y centralizados. Para soportar servicios de baja latencia puede ser necesario llevar tareas del plano de usuarios (UP) cerca de la red de acceso (AN).

En la Figura 5 se observa la arquitectura más sencilla para una red 5G-SA. Este diagrama supone que no se está aplicando *roaming*, solamente un UE está conectado a la red y todos los servicios del *core* de la red se comunican entre sí. Sin embargo, esta es una versión simplificada de las conexiones de la red. En la Figura 6 se muestra la totalidad de las conexiones virtuales en el núcleo de una red 5G.

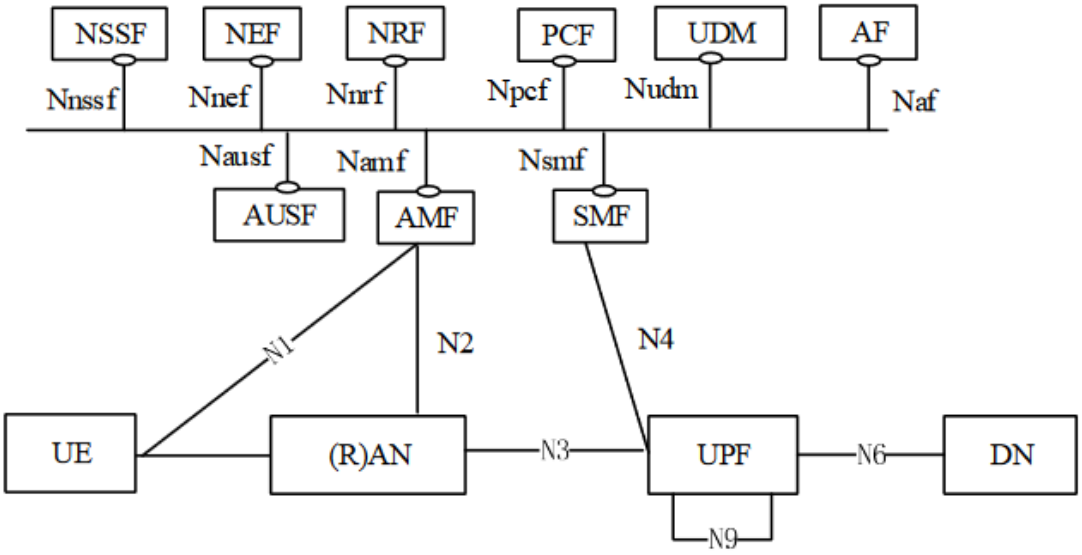

Figura 5. Diagrama de los servicios de una red 5G-SA [23]

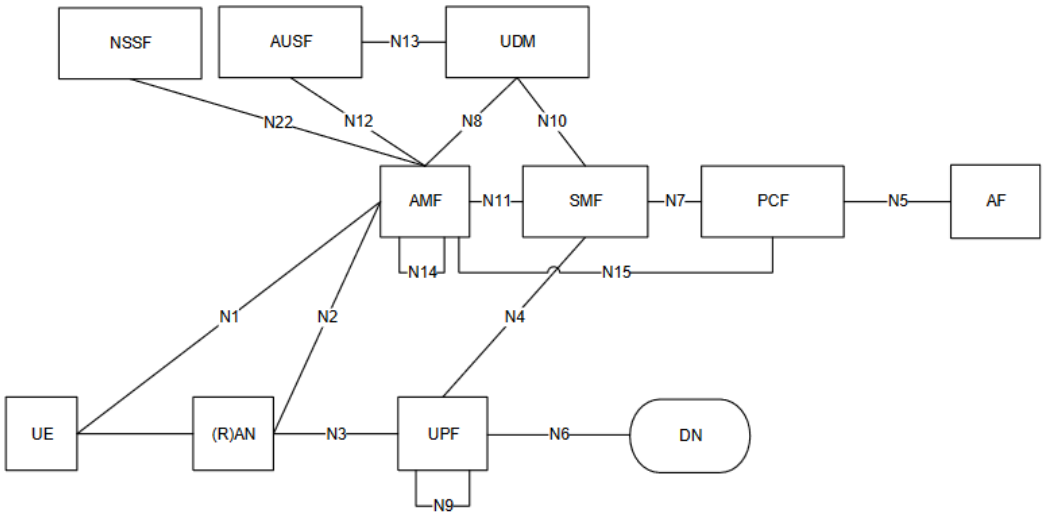

Figura 6. Arquitectura 5G con un UE y sin roaming [23]

En la Figura 5 y en la Figura 6 se ven muchos servicios, llamados *Network Functions (NF)* en inglés. Y es que, al ser 5G la evolución natural de 4G, se sigue la misma filosofía de implementar servicios dedicados a funciones muy concretas. En este sentido podemos afirmar que 5G sigue una filosofía de microservicios. A continuación, se enumeran todos ellos:

- AUSF. Authentication Server Function
- AMF. Access and Mobility Management Function
- DN. Data Network
- UDSF. Unstructured Data Storage Function
- NEF. Network Exposure Function
- NRF. Network Repository Function
- NSSF. Network Slice Selection Function
- PCF. Policy Control Function
- SMF. Session Management Function
- UDM. Unified Data Management
- UDR. Unified Data Repository
- UPF. User Plane Function
- AF. Application Function
- UE. User Equipment
- (R)AN. (Radio) Access Network
- 5G-EIR. 5G-Equipment Identity Register

- SEPP. Security Edge Protection Proxy
- NWDAF. Network Data Analytics Function

Una de las claves de la arquitectura de las redes 5G es que estas son muy flexibles y deben adaptarse a las necesidades de cada aplicación. Es por ello que las redes 5G se pueden llegar a complicar mucho, como se observa en la Figura 7 y en la Figura 8. En ellas se hacen ligeras modificaciones como permitir múltiples sesiones PDU o la habilitación de *roaming*.

Se puede concluir entonces que las redes 5G, a diferencia de 4G, no tienen una estructura concreta, sino que esta es flexible. Los diferentes servicios de la red se pueden intercomunicar de infinitas maneras diferentes, según lo requiera la situación. Por ejemplo, en la Figura 8 se habilita *roaming*, es decir, se consideran dos regiones diferentes de cobertura. En ella, se realiza una distinción entre VPLMN y HPLMN, dos códigos que identifican las diferentes regiones de cobertura. Es posible situar unos servicios solamente en un lado y no en otro.

Sin embargo, esta flexibilidad no solo es práctica para complicar la red y adaptarse a escenarios complejos. Los servicios descritos anteriormente realizan tareas muy concretas. Es posible también que, en una aplicación muy sencilla, haya servicios que no sea necesario implementar y, por tanto, sea posible obviar gracias a esta filosofía modular.

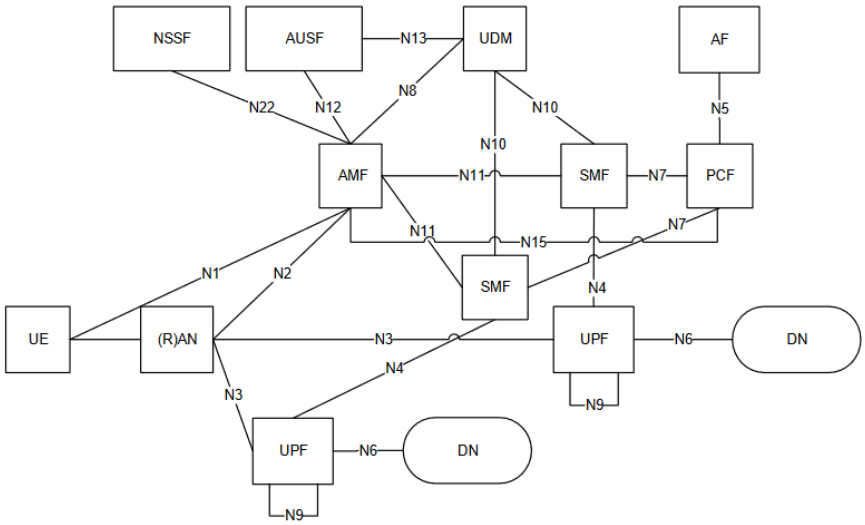

Figura 7. Arquitectura 5G para múltiples sesiones PDU [23]

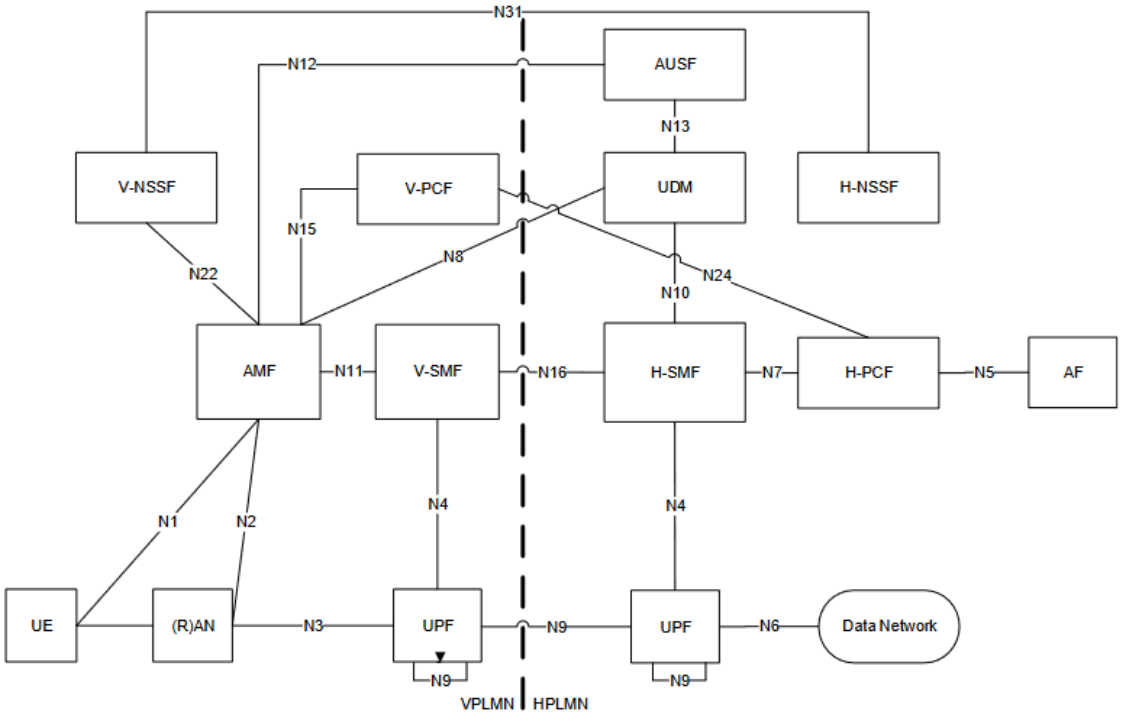

Figura 8. Arquitectura 5G habilitando roaming [23]

Es interesante destacar que algunas organizaciones como el Fraunhofer FOKUS Institute ya están desarrollando la sexta generación de telefonía móvil 6G. El objetivo de la investigación consiste en simplificar el funcionamiento de la red, virtualizando aún más la arquitectura. De esta manera se consigue una red más flexible y escalable. Además, se está tratando de reducir el número de *Network Functions*, ya que en 5G el número de servicios es excesivamente alto [24].

2.1.3 Paradigmas para la implementación del *EDGE Computing*

Como ya se ha mencionado, el 5G surge de una mejora general de las capacidades de las redes inalámbricas y de entre ellas la necesidad de reducir la latencia en las comunicaciones con los servidores centrales. Una posible solución para este problema en particular, además de mejorar las capacidades de red, pasa por acercar los servidores a la frontera de la red, pero existen diferentes paradigmas acerca de cómo llevar a cabo esta labor. En este trabajo se van bajo se van a comparar dos de los enfoques más comunes: Fog Computing y MEC Computing.

El principal objetivo del **Fog Computing** consiste en cubrir la distancia entre los servidores de *cloud* y la frontera de la red [25]. Para ello, sitúa la computación, almacenamiento y gestión de los datos en servidores cercanos a los dispositivos finales,

además de en los servidores centrales, como se muestra en la Figura 9. De esta manera se puede aligerar el flujo de datos de la red, Por ejemplo, la compresión de datos GPS se realiza en servidores Fog antes de ser enviados a la nube [26]. Además de reducir la cantidad de información que se debe enviar a los servidores centrales, los computadores Fog también son capaces de dar respuesta a peticiones sencillas de los dispositivos finales, reduciendo así los retardos asociados a la latencia entre servidores. [27].

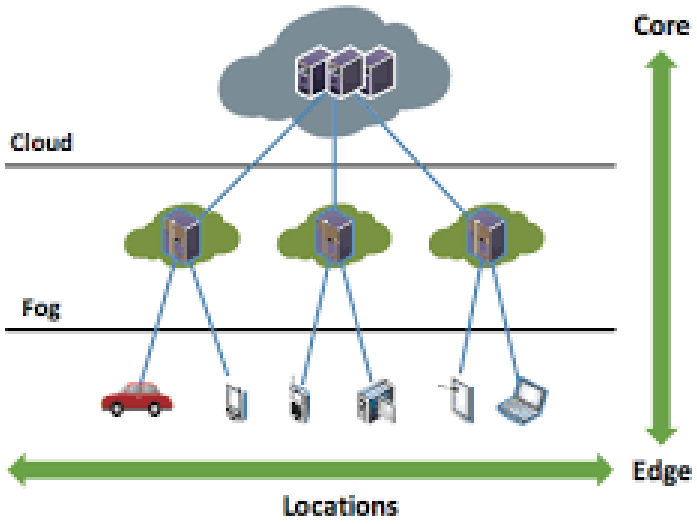

Figura 9. Representación de Fog Computing

Por otro lado, el MEC Computing es el caso extremo de computación en la frontera de la red. En este paradigma, los dispositivos finales pueden realizar el procesamiento de los datos, además de almacenar información, si el dispositivo tiene la capacidad para ello. En caso de que este no posea la capacidad de procesamiento necesaria, la gestión de la información la llevará a cabo un ordenador cercano al dispositivo (normalmente aquel que funcione como *gateway* en la red) [25]. De esta manera se reducen los retardos asociados a la latencia entre servidores al mínimo.

Por tanto, en el caso de uso que se va a desarrollar en el presente proyecto (la gestión de un vehículo que implemente conducción autónoma) resulta más interesante implementar una arquitectura MEC Computing. La información recogida por los diferentes sensores del coche será gestionada por un ordenador cercano que actuará como servidor MEC. Se profundiza en esta arquitectura en la sección 4.1.

2.2 Algoritmos de clasificación basados en Machine Learning

En el presente trabajo se desarrolla como caso de uso para aplicar técnicas de *off-loading* y arquitectura MEC el reconocimiento automático de señales de tráfico. Es común en la literatura resolver este problema utilizando la inteligencia artificial. La inteligencia artificial engloba todas aquellas tareas que un ordenador es capaz de realizar gracias a la interpretación de una serie de datos de entrada [28]. Esto quiere decir que el propio computador debe establecer relaciones y patrones sin que el programador se lo especifique. Sin embargo, el término de inteligencia artificial es demasiado amplio. Se debe matizar la rama de la inteligencia artificial de la que se habla. En particular, hay que distinguir entre *Machine Learning* (ML) y *Deep Learning* (DL).

A pesar de que muchas veces se toman como sinónimos, los paradigmas de ML y DL tienen sutiles diferencias que es necesario aclarar. En el *Machine Learning* el programador parte de una serie de datos de entrada de los cuales debe sacar una serie de características. Ese proceso es el que en la Figura 10 se presenta como *Feature extraction*. Sigamos el ejemplo de dicha figura, donde el objetivo del problema es decidir si una imagen de entrada es un coche o no. En ML el programador debe sacar características de las imágenes para poder introducirlas en el algoritmo. Esas características podrían ser color, tamaño, peso, velocidad máxima, etc. Son esas características las que se deben introducir en el algoritmo [29].

Además, dentro del *Machine Learning,* cabe destacar dos familias de algoritmos: los supervisados y los no supervisados. Los algoritmos supervisados cuentan con una respuesta esperada, con la cual pueden comprobar si la predicción es correcta o no. Por el contrario, los algoritmos no supervisados no poseen esa salida esperada. No se pueden aplicar métricas que calculen la coherencia del resultado.

Por otro lado, el Deep Learning no necesita esa etapa previa de extracción de características. Un algoritmo de DL es capaz de tomar las características necesarias para realizar la clasificación. Como desventaja de este paradigma, se necesita una gran base de datos para entrenar al algoritmo de manera correcta. Además, el tiempo de ejecución es, normalmente, demasiado alto para ser práctico. En aplicaciones de tiempo real no

tiene cabida, a no ser que la capacidad de procesamiento gráfico del ordenador sea extraordinaria.

En las siguientes subsecciones se hace un repaso de algunos de los algoritmos de clasificación basados en *Machine Learning* más comunes en la literatura. Se explica el principio de funcionamiento de todos ellos, además de sus ventajas e inconvenientes. Finalmente, se argumenta cuál es el algoritmo escogido.

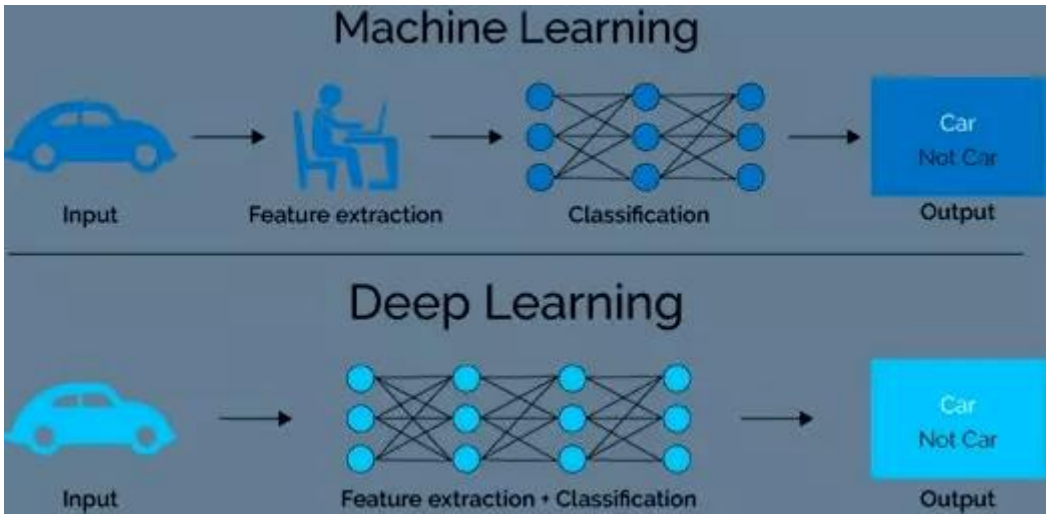

Figura 10. Diferencia entre Machine Learning y Deep Learning [30]

2.2.1 Decision Tree

El primer algoritmo de *Machine Learning* a revisar es el árbol de decisión (DT). Este algoritmo es muy polivalente, ya que es capaz de realizar tareas tanto de clasificación como de regresión. En tareas de clasificación cada nodo del árbol representa una decisión categórica (la respuesta es sí/no), mientras que en regresión las variables son continuas [28]. En nuestro caso de uso, el reconocimiento de señales, necesitamos que las variables sean categóricas, puesto que nos enfrentamos a una tarea de clasificación. Durante la ejecución del algoritmo se van evaluando cada una de estas decisiones y, en función de las respuestas, se van 'podando' unas ramas u otras.

Sin embargo, el algoritmo del árbol de decisión no es demasiado utilizado. Si la clasificación es relativamente compleja, el tamaño del árbol puede ser demasiado grande, haciendo que su ejecución no sea realmente eficiente.

2.2.2 Naïve Bayesian

El algoritmo Naïve Bayesian (NB) es probablemente el más simple de los que se van a revisar. Su funcionamiento se basa en el cálculo de probabilidades condicionadas, gracias a la fórmula de Bayes. El objetivo del algoritmo consiste en agrupar los datos de entrada en clases (tarea de clasificación). Para ello, calcula un histograma de todas las variables del sistema, gracias al cual puede calcular la probabilidad de cada variable. Así, aplicando la ecuación (1), el sistema es capaz de clasificar nuevos datos de entrada.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \quad (1)$$

Debido a que solo necesita hacer cálculos sencillos para la clasificación, NB es un algoritmo tremendamente utilizado en aplicaciones como la diferenciación de correos personales y SPAM. No obstante, esa sencillez provoca que su rendimiento en general sea bajo [31], por lo que no será el algoritmo implementado en nuestra aplicación.

2.2.3 K – Nearest Neighbors

El algoritmo K-Nearest Neighbors (KNN) trata de separar dos nubes de puntos situados en un plano contando cuántos 'vecinos', o puntos adyacentes de la misma clase, posee cada elemento. Por lo tanto, es un algoritmo de clustering o clasificación. Una de las grandes ventajas de este algoritmo es que no presupone ningún tipo de distribución en los datos de entrada – es decir, es un método no paramétrico. Sin embargo, este algoritmo tiene un gran inconveniente. Cuando el dataset de entrada es muy extenso, el cálculo de los vecinos de cada punto de entrada se hace muy laborioso para el procesador. El vasto tiempo de ejecución hace que no sea el algoritmo idóneo para aplicaciones en tiempo real [31].

2.2.4 Support Vector Machines

Finalmente, el algoritmo SVM es el escogido para realizar la tarea de reconocimiento de señales de tráfico. El objetivo de SVM, al igual que el de KNN, consiste en separar nubes de puntos situados en el espacio (de tantas dimensiones como el algoritmo requiera). Para ello, trata de buscar el hiperplano capaz de separar de la

mejor forma posible estos grupos de puntos. Se trata de un algoritmo que una vez entrenado tiene un tiempo de respuesta muy rápido y además suele tener rendimientos bastante elevados, por lo que es ideal para nuestro caso.

Existen distintas implementaciones del algoritmo. Como se ha expuesto en la sección 2.2, los algoritmos de ML necesitan que el programador extraiga una serie de características. En este trabajo se va a utilizar, y por tanto se va a explicar en detalle, el algoritmo **HOG-SVM.** HOG, o el histograma de los gradientes orientados, será la característica que necesitamos extraer de las imágenes de entrada para poder entrenar el algoritmo. A nuestro sistema le llegará una imagen $A(x, y)$, la cual debemos convertir a una imagen binaria $B(x, y)$, es decir, convertir a blancos y negros puros. Posteriormente, y después de aplicar una transformación Gamma (corrección de la saturación), se obtiene el gradiente de la imagen $G(x, y)$ [32]. Tras este preprocesado, se obtiene una 'imagen' compuesta de vectores que apuntan en dirección de máximo crecimiento de la intensidad. En la Figura 11 se ve un ejemplo de este preprocesado para una aplicación capaz de reconocer números escritos a mano.

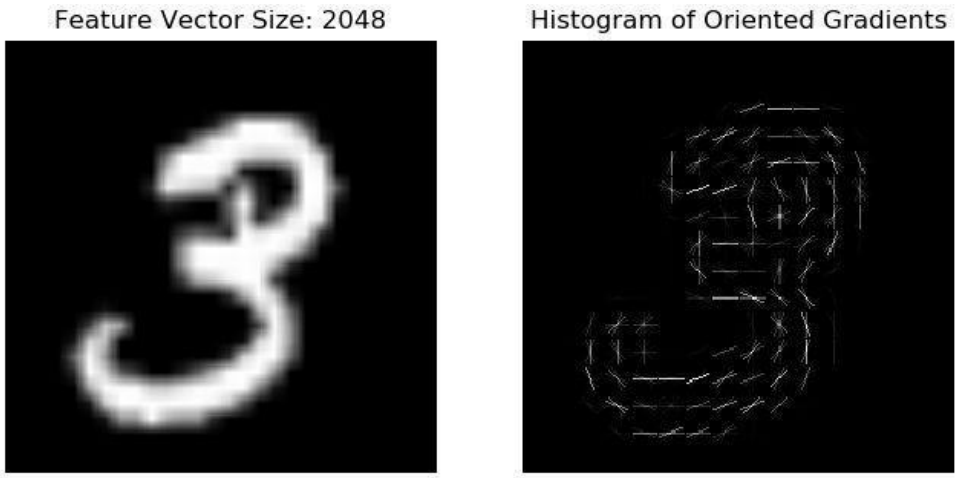

Figura 11. Aplicación del procesado HOG [33]

El gradiente de una función de N variables se obtiene aplicando la derivada a cada una de las componentes del campo. En nuestra aplicación, la función de entrada es una imagen $H(x, y)$, donde H representa, para cada valor de (x, y), la intensidad de cada píxel. El gradiente, por tanto, se obtiene calculando la derivada unidireccional tanto en x como en y. Si tomamos $G(x, y) = \left[G_x(x, y), G_y(x, y)\right]$ como el gradiente de $H(x, y)$, cada una de las componentes se obtiene con:

$$\begin{cases} G_x(x,y) = H(x+1,y) - H(x-1,y) \\ G_y(x,y) = H(x,y+1) - H(x,y-1) \end{cases}$$

Comprobamos que simplemente se aplica la operación derivada discreta en cada componente. Si pensamos en la derivada aplicada a imágenes, nos damos cuenta de que va a eliminar las zonas donde la intensidad sea uniforme, y va a resaltar donde existan cambios abruptos en la intensidad. Esencialmente, la derivada actúa como un filtro paso alto. Por tanto, un filtro de estas características aplicado a imágenes actúa como un detector de bordes.

En la práctica, sin embargo, es común en la literatura el uso del **operador de Sobel** para la detección de bordes. Este algoritmo calcula el gradiente convolucionando la imagen original con una serie de matrices, o '*kernels* de Sobel'. Con un coste computacional muy similar al del cálculo de las derivadas, ofrece unos resultados muy superiores al introducir los valores '2' y '-2' en sus máscaras. Esto proporciona un peso mayor a los píxeles colindantes al borde, dando más intensidad al borde [34]. El gradiente, pues, se calcula como:

$$G_x(x,y) = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 2 \end{bmatrix} * H(x,y), \quad G_y(x,y) = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * H(x,y)$$

De esta manera se consiguen las componentes horizontal y vertical del gradiente. Sin embargo, el algoritmo necesita el gradiente en representación módulo-fase, para conocer la longitud de cada vector y su dirección. Si denotamos al módulo del gradiente como $|G(x,y)|$ y a la fase como $\theta(x,y)$, se tiene:

$$|G(x,y)| = \sqrt{G_x(x,y)^2 + G_y(x,y)^2} \qquad \theta(x,y) = arctg\left(\frac{G_y(x,y)}{G_x(x,y)}\right)$$

En la Figura 12 se muestra un ejemplo de aplicación del operador de Sobel. En ella se muestra la imagen de una mujer. El resultado de la aplicación del operador es una imagen en la que solo permanecen los bordes más notables.

Tras la obtención del gradiente de las imágenes de entrada, el algoritmo debe realizar una ***orientation binning*** [32]. Este proceso consiste en dividir la imagen en celdas para, posteriormente, calcular el histograma de cada celda. En [35] se observa

que el mejor rendimiento se tiene cuando se utilizan 9 canales de histogramas combinados con gradientes sin signo. La Figura 13 ejemplifica a la perfección el proceso descrito.

Finalmente, tras un proceso de normalización, los datos ya estarían listos para ser introducidos en el sistema SVM.

Figura 12. Detección de bordes con el operador de Sobel [36]

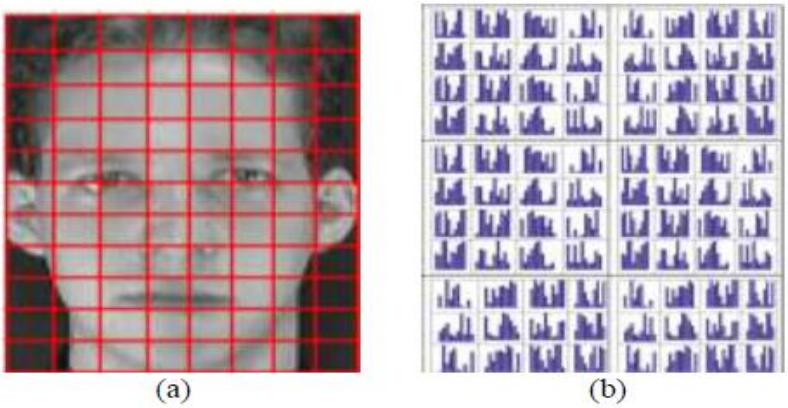

Figura 13. (a) Imagen dividida en celdas. (b) Histograma de cada celda [32]

Como ya se ha comentado, el algoritmo **Support Vector Machines** se encarga de calcular el hiperplano de máximo margen que separe diferentes grupos de muestras [37]. Sean $h(x)$ el hiperplano que separa las nubes de puntos, $\vec{w}$ el vector normal al plano, $\vec{x}$ el vector de entrada y b la desviación. La ecuación del hiperplano es:

$$h(x) = sign(\vec{w}^{\,T} \cdot \vec{x} + b)$$

El producto escalar de dos vectores representa su proyección, que es lo que realmente se necesita en el algoritmo. Suponiendo que la distancia del vector $\vec{w}$ desde el

origen al plano es c, entonces tomamos la proyección de los vectores $\vec{x}$ y $\vec{w}$ [38]. La Figura 14 muestra un ejemplo de ejecución SVM en dos dimensiones.

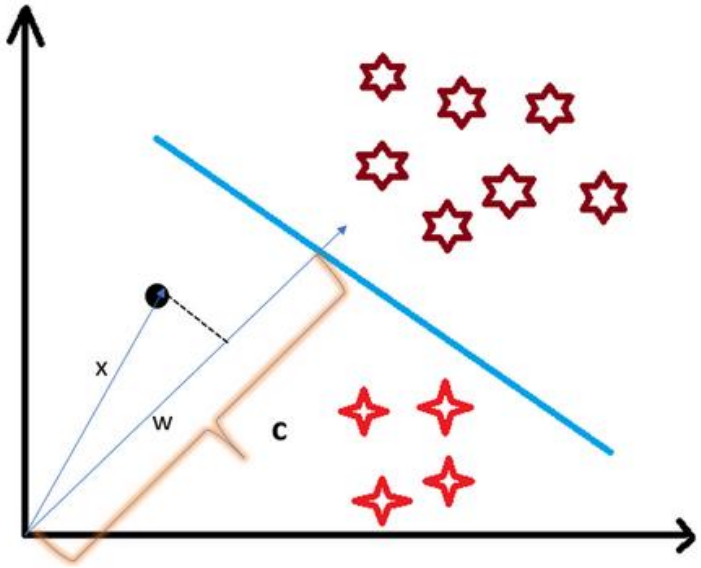

Figura 14. Ejemplo de separación de dos nubes de puntos con SVM [39]

Según el producto escalar entre $\vec{x}$ y $\vec{w}$ es posible deducir a qué grupo pertenece una muestra. En este ejemplo, si el resultado es menor que c, asignaremos el punto al conjunto de estrellas de cuatro puntas. Si es mayor, al de estrellas de seis puntas. Por último, si es exactamente igual a c, entonces el algoritmo no sabrá a qué grupo pertenece esa muestra [39]. Para evitar esta situación, SVM introduce un cierto margen de decisión.

Existen diversas maneras de establecer estos límites, en función del dataset de entrada o de las necesidades de la aplicación. Los dos límites más utilizados son el *hard-margin* y el *soft-margin.* El *hard-margin* se emplea cuando los diferentes grupos de puntos son separables, es decir, las nubes de puntos están muy bien diferenciadas y es trivial establecer una frontera entre ellas. Sin embargo, eso no es lo habitual. En la mayoría de casos es necesario implementar un *soft-margin*, capaz de trabajar con grupos de muestras mezclados. En nuestra aplicación se ha optado por un *soft-margin.*

Establecer este margen, más flexible, no es una tarea sencilla. Su formulación matemática obedece a una operación de optimización. Se trata de resolver la siguiente ecuación:

$$mín_w \quad \frac{1}{2}\,\|\vec{w}\|^2 + C\sum_{i=1}^{N}\xi_i$$

donde C es una constante que ajusta la laxitud del margen y ξ_i es una variable que permite clasificar de manera errónea algunas muestras [38]. La Figura 15 muestra un ejemplo de *soft-margin* aplicado a dos grupos de muestras inseparables.

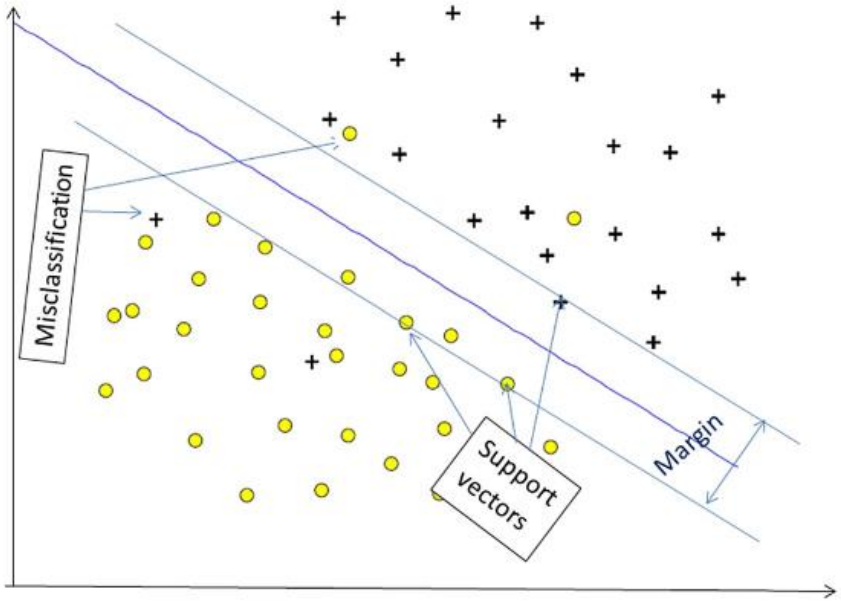

Figura 15. Implementación de soft-margin [38]

Resolver las ecuaciones para obtener los márgenes óptimos para cada clasificación es una tarea demasiado compleja para un ordenador. El tiempo de cómputo es excesivo para aplicaciones en tiempo real. Para solventar esta problemática se han desarrollado los *kernels.* Un *kernel* es un truco matemático para resolver problemas no lineales como si fueran lineales [40]. Existen diferentes tipos de *kernel,* como el *Sigmoid Kernel Function* (SKF) o el *Hyperbolic Tangent Kernel* (HTK). El *kernel* implementado en nuestra solución es el *Radial Basis Function* (RBF).

El *kernel* RBF es uno de los más utilizados en algoritmos SVM ya que es relativamente sencillo, en comparación a otros *kernels*, y porque no asume ningún tipo de distribución en los datos de entrada [41]. La formulación matemática es la siguiente:

$$K(x_i, x_j) = \exp\left(-\gamma\,\|x_i - x_j\|^2\right)$$

Se puede apreciar que esta fórmula es bastante más sencilla que la del *soft-margin*, ya que solo necesita calcular la distancia euclídea entre puntos adyacentes para funcionar. Aparece un cierto parámetro γ, el cual es inversamente proporcional al cuadrado de la desviación típica del dataset de entrada. La simplicidad es una de las grandes virtudes del *kernel* RBF ya que solo necesita ajustar dos variables (C, γ). Se recuerda que C es la constante de laxitud de los márgenes.

2.2.5 Elección del algoritmo

Este trabajo se apoya en comparaciones realizadas por [42] y [43]. En estos dos trabajos se realiza una comparativa entre algoritmos de clasificación en diferentes ámbitos, con resultados similares. La Figura 16 muestra una gráfica con la precisión de los diferentes algoritmos en función del *Number of folds.* En la literatura, el *Number of Folds* hace referencia al porcentaje del dataset que se utiliza para el entrenamiento del algoritmo en relación con el porcentaje destinado a test [43] . Se observa una tendencia al alza para cualquier algoritmo. Cuanto mayor sea el porcentaje del dataset destinado al entrenamiento, mejores resultados se obtienen. En cualquier caso, el algoritmo basado en ML que ofrece mejores resultados de precisión es SVM. Es SVM, por tanto, el algoritmo utilizado en este Trabajo de Fin de Grado.

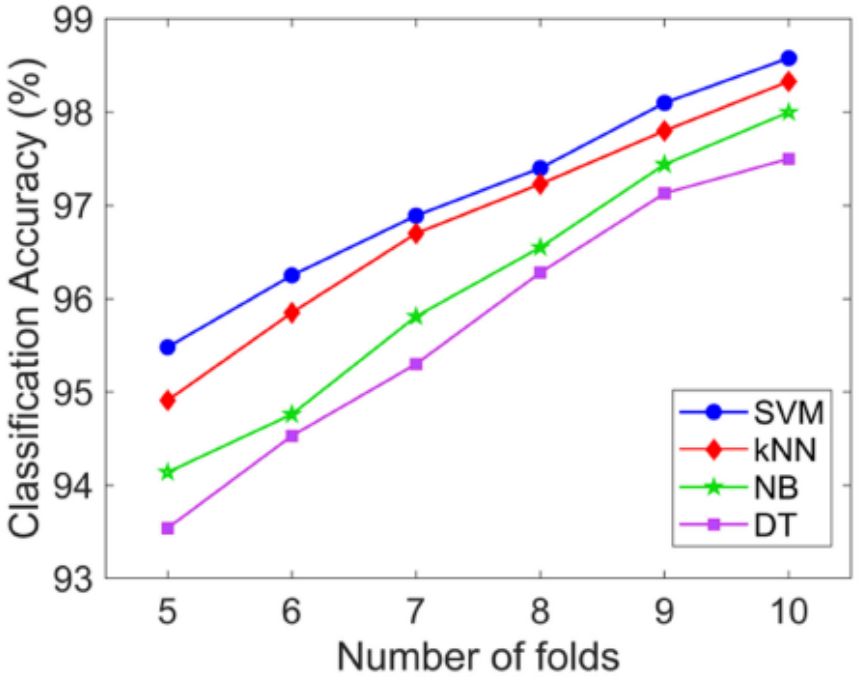

Figura 16. Comparativa entre algoritmos [43]

Capítulo 3. Estudio de OpenAirInterface

El *software* escogido para el despliegue de la red de datos móvil es OpenAirInterface. Este programa de código abierto es capaz de virtualizar redes 4G y 5G-NSA, con diferentes escenarios de implementación. A lo largo de este capítulo se explicarán estos diferentes casos de implementación, y se profundizará en el escenario escogido para este trabajo. Además, se explicará el proceso de instalación y despliegue de la red. Finalmente, se comentarán los resultados obtenidos de rendimiento.

3.1 Sobre OpenAirInterface

OpenAirInterface es una empresa que pertenece a la OpenAirInterface Software Alliance (OSA). La OSA es un consorcio sin ánimo de lucro que desarrolla un ecosistema para aplicaciones *open-source* capaces de emular la arquitectura EPC – EUTRAN de las redes 3GPP [44]. Actualmente, la compañía OpenAirInterface tiene cuatro proyectos abiertos: 5G RAN, 5G Core Network, Mosaic5G y CI/CD (*Continuous Integration* / *Continuous Deployment*). Se observa que todos los esfuerzos de la asociación están enfocados en la tecnología 5G.

El proyecto en el que se va a centrar este trabajo es Mosaic5G. En él, OAI se ha aliado con la empresa francesa Eurecom, líder en el sector de las telecomunicaciones galas. Se dedica, principalmente, a la investigación acerca de sistemas de comunicación, ciencia de datos y ciberseguridad [45]. Algunos de los miembros del consorcio son tremendamente relevantes, como el grupo BMW, Orange, el Principado de Mónaco o la Technische Universität München (TUM). Asimismo, imparte cuatro titulaciones de Máster en carreras de ciencias junto con el Institut Mines-Téleécom, todos ellos centrados en la ciencia de datos y sistemas de comunicaciones móviles [46].

3.2 Escenarios de implementación

Una de las grandes ventajas de OAI y una de las razones más importantes por las que resultaba interesante estudiarlo es su flexibilidad a la hora del despliegue de la red. OAI ofrece la posibilidad de simular un CN, un eNB y hasta un UE. Además, dado que sigue el estándar e implementa las interfaces definidas en él, es posible crear arquitecturas en las que se combinen *softwares* de distintos fabricantes (por ejemplo,

OAI EPC + srsRAN eNB), pudiendo de esta manera aprovechar al máximo las capacidades que ofrecen cada uno de ellos. Dependiendo de la aplicación que se necesite resolver se optará por un diseño u otro. A continuación, se muestran diferentes configuraciones que el *software* permite [47]:

- OAI CN + OAI eNB + UE comercial
- OAI CN + OAI eNB + OAI UE
- OAI CN + eNB comercial + OAI UE
- CN comercial + OAI eNB + OAI UE
- CN comercial + OAI eNB + UE comercial
- CN comercial + eNB comercial + OAI UE

Durante el desarrollo de este Trabajo Fin de Grado se han probado las tres primeras configuraciones enumeradas, y finalmente se optó por centrarse en la primera de ellas: OAI CN + OAI eNB + UE comercial, ya que era con la que mejor podíamos conocer todas las ventajas y limitaciones del *software*.

3.2.1 Arquitectura implementada

Uno de los objetivos finales del proyecto Artemis es implementar un caso de uso de tecnología *edge* sobre una maqueta. Dicha maqueta estará constituida por una red 4G o 5G y un vehículo a escala con conexión 4G o 5G mediante un modem USB. Es por ello que la implementación más lógica consiste en utilizar el CN y el eNB de OpenAirInterface, y un módem USB como UE.

Es posible desplegar la red en una única máquina o separar el CN y el eNB en dos máquinas diferentes. Por lo general, un ordenador no posee la potencia suficiente como para albergar toda la red, por lo que se suelen utilizar dos ordenadores distintos. Sin embargo, si se dispone de una máquina lo suficientemente potente, es posible lanzar en ella tanto el CN como el eNB.

Para el despliegue de OpenAirInterface en una única máquina, algunos trabajos como [47] optan por el uso de máquinas virtuales, de manera que cada segmento de la red se ejecuta en una máquina independiente. Sin embargo, en este trabajo no se ha considerado oportuno trabajar con máquinas virtuales, ya que se posee un ordenador

con dos interfaces de red. Además, como se aprecia en la Figura 17, es posible utilizar direcciones de *loopback*, es decir, direcciones IP que comienzan con 127 y hacen referencia a interfaces virtuales dentro de una misma máquina. Es por esto que no ha sido necesario utilizar máquinas virtuales en este proyecto.

En la Figura 17 se muestra un diagrama de la arquitectura propuesta, incluyendo las direcciones IP necesarias para las conexiones virtuales. Además, si se compara la red con la propuesta en el Gitlab de OAI [45], la cual se ve en la Figura 18, se observa que no aparece la interfaz S1-MME que conecta el MME con el gNB. Esto es debido a que esa conexión directa entre el CN y el gNB solo existe en 5G-SA. En este despliegue, el gNB solo se comunica con el eNB.

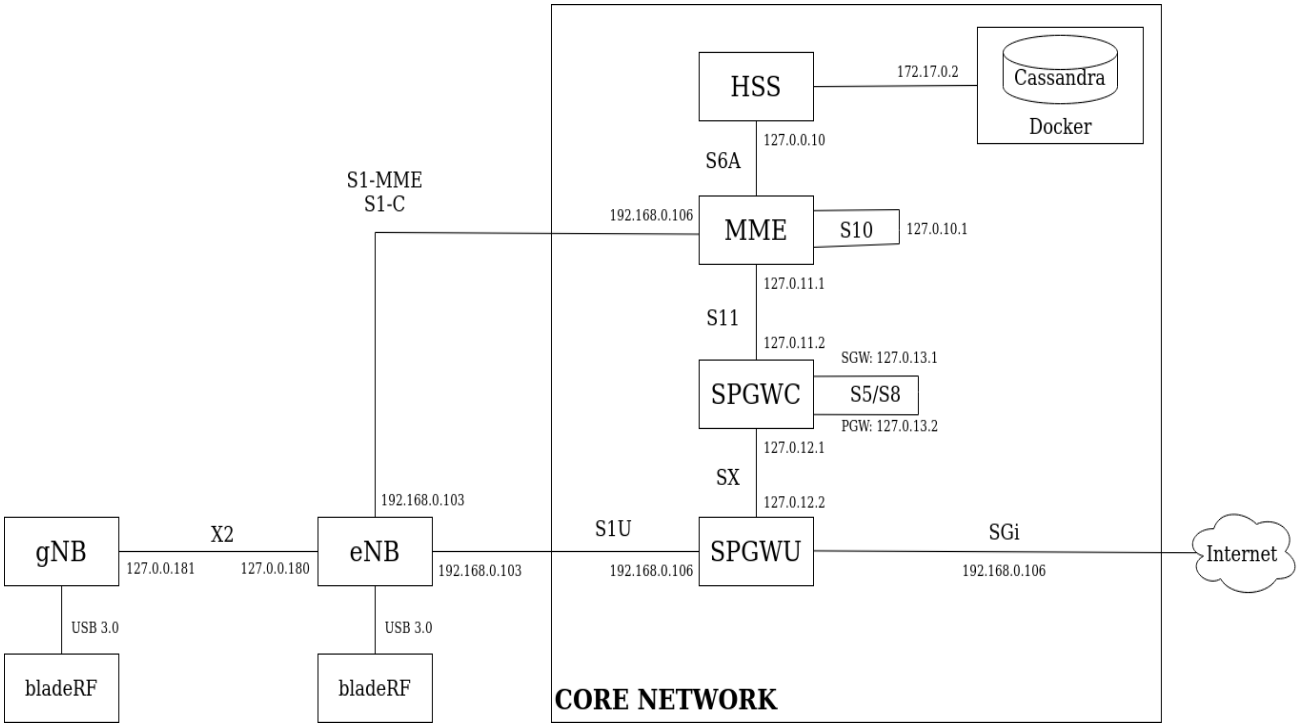

Figura 17. Diagrama de la red

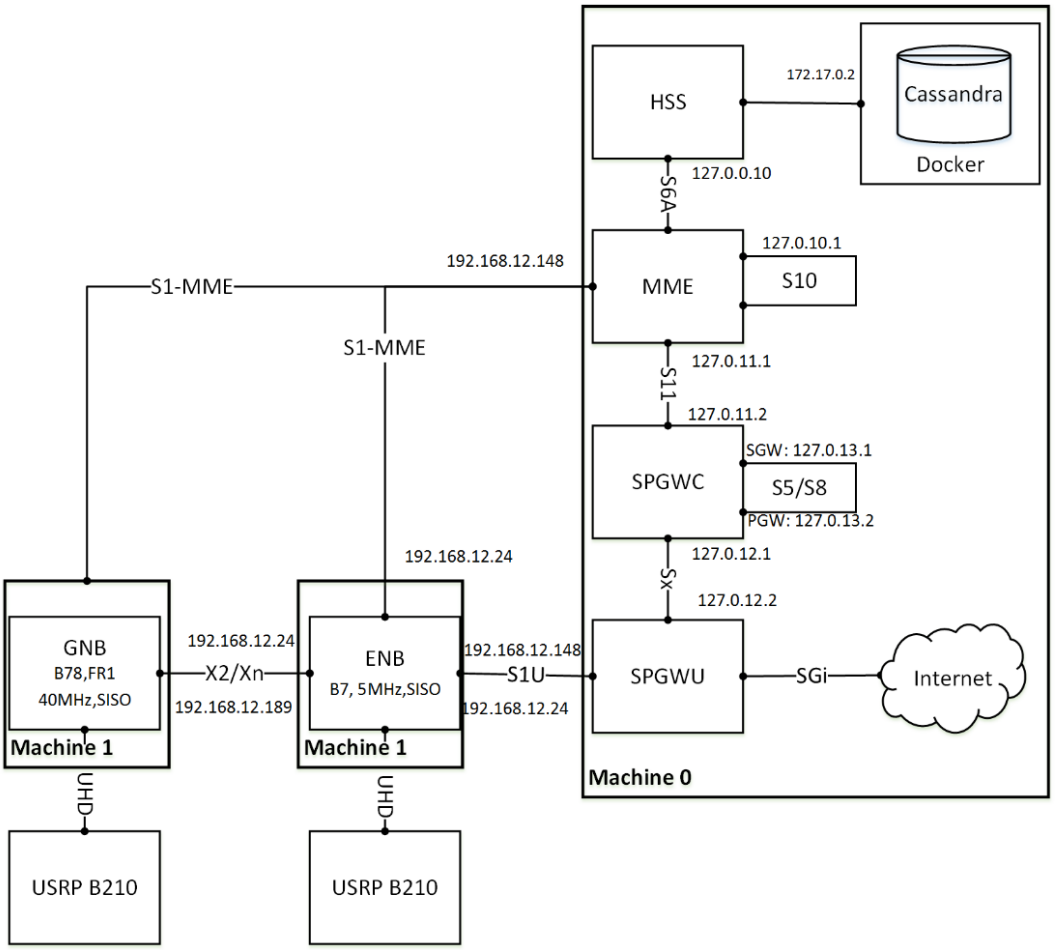

Figura 18. Diagrama de la red propuesta por OAI

En ambas figuras se muestran todos los elementos de una red LTE que se han explicado en el capítulo 2: HSS, MME, SPGWC y SPGWU. Además, se aprecia que la base de datos es Cassandra, que está contenida en un Docker. Esta configuración se explica en mayor detalle en la sección 3.4.1.

En la Figura 19 se muestra una foto del despliegue final de la red con todos los equipos. El portátil Thinkpad virtualiza el núcleo de la red, además de la base de datos. El ordenador de sobremesa, el Intel NUC, actúa como el eNB. Se puede observar que la SDR está conectada al NUC por USB 3.0. Finalmente, el portátil situado en la izquierda tiene conectado el módem USB que hace de UE.

Aunque se explica con más en detalle en la sección 3.8, en la pantalla de la máquina que virtualiza el eNB se muestra una herramienta que implementa OpenAirInterface llamada *soft-scope*. Esta utilidad muestra una serie de gráficas que analizan el rendimiento de la red.

Figura 19. Implementación real de la red

Sin embargo, también se realizaron pruebas desplegando toda la red en el ordenador central, ya que es el más potente. Aunque ya se comentará con mayor detalle en la sección 3.8.4, el desempeño de la red resultó ser mayor concentrando el CN y el eNB en una única máquina.

3.3 Preparación de las máquinas

Para la instalación del *software* se siguió principalmente la referencia [48], la wiki que ofrece OpenAirInterface. Hay que tener en cuenta que esta referencia instala toda la red en una sola máquina y utiliza una Ettus USRP B210 como radio definida por *software*. Dado que en nuestro caso vamos a utilizar varios ordenadores para separar las componentes del *software* y además nuestro dispositivo SDR es una BladeRF2.0, la instalación sufrió una serie de modificaciones para poderse llevar a cabo. Es por ello que, con objeto de simplificar la instalación, se ha desarrollado una wiki propia, la cual se encuentra en el repositorio Github del grupo Artemis [49].

La instalación se realizó de la siguiente manera: En el ordenador Thinkpad se instaló el core network de OAI y en el NUC de Intel se instalaron tanto el eNB como el gNB. Con esta selección de equipos y software, se ha instalado Ubuntu 20.04.3 LTS en el Thinkpad y Ubuntu 18.04.6 LTS en el NUC. Es importante señalar que la wiki de

instalación de OAI señala que hay que instalar un Ubuntu 16.4, pero esta versión conllevaba dos problemas:

1. El software de OAI para eNB realiza una instalación y calibrado automático de la BladeRF x40. Sin embargo, nosotros trabajamos con la BladeRF2.0 micro xa9. Por lo tanto, no se puede realizar esta instalación automática recomendada y hay que instalar las librerías manualmente. No existen librerías disponibles para la BladeRF2.0 en Ubuntu 16.4. La primera versión que lo soporta es la 18.04, razón por la que era necesario subir al menos a esta versión.
2. Versiones superiores de Ubuntu como la 20.04.3 que se utiliza en el Thinkpad son demasiado nuevas y el software de OAI no compila.

En definitiva, es crucial instalar Ubuntu 18 en la máquina que vaya a correr el eNB, ya que el *software* de OpenAirInterface está preparado para ello. Además, es interesante instalar el *kernel* de baja latencia en ambas máquinas. Este *kernel* requiere un consumo energético mayor, pero consigue que la latencia media de un ordenador baje hasta a un tercio de su valor original [50]. Para instalarlo necesitamos introducir el siguiente comando en una consola:

```
$ sudo apt-get update
$ sudo apt-get install linux-lowlatency
```

Es probable que sea necesario reiniciar la máquina. Para comprobar que la instalación ha sido correcta debemos introducir el comando `$ uname -a`. La salida esperada es:

```
$ uname -a
Linux artemis-NUC3 4.15.0-188-lowlatency #199-Ubuntu SMP PREEMPT Wed Jun 15 21:15:04 UTC 2022 x86_64 x86_64 x86_64 GNU/Linux
```

3.3.1 Clonación del repositorio

El primer paso para la instalación de OAI consiste en clonar el repositorio de Github de Mosaic5G. Para ello:

```
$ git clone http://gitlab.eurecom.fr/mosaic5g/mosaic5g.git
$ cd mosaic5g/
```

Según el tutorial de [48] deberíamos instalar los ficheros predeterminados para lanzar una red 5G-NSA. OAI facilita una instalación realmente simple para la arquitectura propuesta en la Figura 18. Sin embargo, como no se va a implementar esa configuración, no se va a seguir esta parte del tutorial. Se va a proceder a instalar y configurar, de manera manual, el CN y el eNB.

3.4 Instalación del CN

Como ya se expuso en el capítulo 2, el núcleo de una red LTE está formado por cuatro servicios principales: HSS, MME, SPGWC y SPGWU. En esta sección se van a instalar todos ellos.

3.4.1 Instalación de la base de datos

OpenAirInterface trabaja con Cassandra DB, una base de datos NoSQL 'clave-valor'. Esta base de datos se encuentra dentro de un contenedor Docker para hacer la instalación y el uso más simple. En línea de comandos:

```
$ sudo snap install docker
$ docker ps
$ docker run --name cassandra-docker -d -e CASSANDRA_CLUSTER_NAME="OAI HSS Cluster" \ -e CASSANDRA_ENDPOINT_SNITCH=GossipingPropertyFileSnitch \
mosaic5gecosys/cassandra-m5g:1.0       # Create a cassandra container for the HSS
$ docker exec -it cassandra-docker bash
$ nodetool status
$ cqlsh –file ./oai_db.cql 172.17.0.2
```

Se está asignando la dirección IP 172.17.0.2 al contenedor Docker. Esta IP es la que OAI asigna por defecto. Se recomienda mantenerla, ya que esta dirección está referenciada en muchos otros archivos.

En caso de que `$ docker ps` devuelva un error del tipo *permission denied* deberemos introducir los siguientes comandos:

```
$ sudo groupadd docker
$ sudo adduser $USER docker
```

Si el error persistiese, la razón más probable es que faltan permisos por conceder. Simplemente introduciendo `$ sudo chmod 777 /var/run/docker.sock` se debería de solucionar el problema, ya que se están otorgando permisos de lectura, escritura y ejecución. Para hacer efectivos los cambios es necesario reiniciar y volver a intentar `$ docker ps`.

Si, por cualquier razón, se quisiera eliminar un contenedor Docker, solo habría que escribir:

```
$ docker stop <Container_ID>
$ docker rm <Container_ID>
$ docker rmi -f <Image_Name>
```

3.4.2 Instalación de los Snaps

Para continuar, es necesario instalar los snaps del CN:

```
$ ./build_m5g -C v2
```

Una manera de comprobar si la instalación ha sido correcta es introducir el comando `$ snap list`. En la salida esperada se debería encontrar *oai-hss, oai-mme, oai-spgwc, oai-spgwu.*

3.4.3 Configuración de HSS

Primero, se ha de conceder una serie de permisos:

```
$ sudo snap connect oai-hss:log-observe
```

```
$ sudo snap connect oai-hss:process-control
$ sudo snap connect oai-hss:network-control
```

Entonces, se crean los ficheros de configuración para HSS:

```
$ sudo oai-hss.init
```

Para comprobar que la instalación ha sido correcta, los ficheros *hss_rel14_fd.conf, hss_rel14.json, hss_rel14.json, oss.json* deberían aparecer en la salida de:

```
$ sudo oai-hss.conf-get
```

Se debe comprobar si en *hss_rel14.conf* el campo 'Identity' coincide con ".openair5G.eur" (se puede conocer tu nombre de usuario con el comando *`hostname`*).

Posteriormente se debe recrear certificados para la autenticación del MME. Para ello, se debe introducir de nuevo el comando `oai-hss.init`. Esta acción no modificará la configuración actual.

```
$ sudo oai-hss.init
```

Ya está todo listo para arrancar el servicio HSS. Para ello, habiendo ejecutado el contenedor de la base de datos con `$ docker start cassandra-docker`, se debe introducir el siguiente comando:

```
$ sudo oai-hss
```

Si la instalación ha sido correcta, se nos debería de informar de que ha empezado el *freediameter*. En caso de error:

- Comprobar que los certificados en /var/snap/oai-hss/current/ terminan en pem.
- Si existe un error del tipo '`ERROR TLS: The certificate owner does not match the hostname '<host>.openair5G.eur''`, comprobar que /etc/hosts contiene una línea '`127.0.0.1 <host>.openair5G.eur <host> hss`'. Debería ser la única línea de este tipo. Después, volver a introducir `$ sudo oai-hss.init`.
- Si el error es del tipo '`Address already in use`', cerrar todas las instancias HSS e introducir `$ sudo oai-hss.stop`.

3.4.4 Configuración de MME

Al igual que con HSS, se debe conceder los siguientes permisos:

```
$ sudo snap connect oai-mme:log-observe
$ sudo snap connect oai-mme:process-control
$ sudo snap connect oai-mme:network-control
```

Creamos los ficheros de configuración:

```
$ sudo oai-mme.init
```

Para obtener la ruta de dichos ficheros de configuración se debe introducir el comando:

```
$ sudo oai-mme.conf-get
```

Dentro del fichero mme.conf, en la sección MME:

- Cambiar el `HSS_HOSTNAME` para adecuarlo al *hostname* de su máquina
- Cambiar el `GUMMEI_LIST` y el `TAI_LIST` para que coincida con el PLMN (*Public Land Mobile Network*) deseado. El PLMN es un código formado por

otros dos códigos: el MCC (*Mobile Country Code*) y el MNC (*Mobile Network Code).* En nuestro caso, utilizamos MCC = 901 y MNC = 70.

Además, en la sección NETWORK_INTERFACES, se debe cambiar la dirección IP para que esta coincida con la dirección de nuestra máquina. Por defecto, la red está preparada para ser implementada en una sola máquina.

En la sección WRR_LIST_INTERFACES, debemos asegurarnos de que el ID sigue la estructura:

‘`tac-lbXX.tac hbYY.tac.epc.mncMNC.mccMCC.3gppnetwork.org`’

donde XX e YY son los *lower-byte y higher-byte* del TAC (*Tracking Area Code)*, respectivamente. En la implementación propuesta se utiliza el TAC 7. La sentencia es, por tanto:

‘`tac-lb07.tac hb00.tac.epc.mnc070.mcc901.3gppnetwork.org`’

Además, la dirección de la interfaz S11 debe concordar con nuestra configuración.

En el fichero mme_fd.conf:

- Comprobar que el campo *Identity* contiene el *hostname* de nuestra máquina (al igual que en hss_rel14_fd.conf).
- Comprobar igualmente en la sección *ConnectPeer*

Ya está todo listo para lanzar el MME. La última línea de la salida debería indicar que MME se encuentra en STATE_OPEN.

```
$ sudo oai-mme
[...]
Initializing S6a interface: DONE
MME app initialization complete
Peer '<hostname>.openair5G.eur' is now connected...
```

Finalmente, debemos comprobar en la ventana del HSS que la conexión se ha establecido correctamente con el MME. La última línea debería ser algo parecido a:

```
'STATE_CLOSED'  -> 'STATE_OPEN' '<hostname>.openair5G.eur'
```

3.4.5 Configuración de SPGWC y SPGWU

Al igual que con los servicios anteriores, debemos asignar una serie de permisos:

```
$ sudo snap connect oai-spgwc:log-observe
$ sudo snap connect oai-spgwc:process-control
$ sudo snap connect oai-spgwu:process-control
$ sudo snap connect oai-spgwu:network-control
$ sudo snap connect oai-spgwu:firewall-control
$ sudo snap connect oai-spgwu:log-observe
```

Debemos crear los ficheros de configuración:

```
$ sudo oai-spgwc.init
$ sudo oai-spgwc.conf-get
$ sudo oai-spgwu.init
$ sudo oai-spgwu.conf-get
```

En ambos ficheros de configuración spgwc.conf y spgwu.conf, debemos comprobar que el campo INTERFACES se ajusta a nuestra red.

La sección de controladores remotos está orientada a manejar un LL-MEC (*Low Latency* – MEC), una implementación de baja latencia de MEC. Por el momento no se va a implementar este tipo de arquitecturas, por lo que debemos asegurarnos de que la variable REMOTE_CONTROLLER_ENABLED se encuentre en ‘no’.

Finalmente, ya está todo configurado para lanzar el CN. Debemos asegurarnos de que los SPGWC y SPGWU se envíen unos mensajes de control llamados HEARTBEATS.

```
$ sudo oai-spgwc
$ sudo oai-spgwu
```

3.5 Instalación del eNB

El fichero de configuración del eNB depende directamente del *hardware* que utilicemos como SDR. Por defecto, Open Air Interface presupone que se utiliza una USRP B210, pero en nuestro caso se usa una BladeRF xA9. Es posible acceder al fichero de configuración de la BladeRF como eNB utilizando el comando:

```
$ sudo vim openairinterface5g/configuration/bladeRF/enb-band7-5mhz.conf
```

En este fichero debemos asignar el PLMN de nuestra red. Se recuerda que el escogido en esta aplicación es MCC = 901, MNC = 70, de acuerdo con las tarjetas Sysmocom que utilizamos (se detallan en la sección 3.7).

```
plmn_list = ( { mcc = 901; mnc = 70; mnc_length = 2; } )
```

Entonces, nos centramos en la parte relacionada con el MME. Solo vamos a necesitar establecer la dirección ipv4, ya que, como se especifica en la cuarta línea, es prioritaria ante ipv6.

```
mme_ip_address = ({ipv4 = "192.168.0.106";
                    ipv6 = "192:168:30::17";
                    active = "yes";
                    preference = "ipv4";});
```

3.5.1 Instalación de los *drivers* de BladeRF

Como ya se ha mencionado, el *hardware* utilizado es una SDR BladeRF xA9. Para poder utilizarla se han de instalar previamente los drivers de esta tarjeta. Como se dijo en la sección 3.3, el ordenador utilizado tiene que tener instalado exactamente la distribución de Linux Ubuntu 18.04.6 LTS. Se puede comprobar la distribución de Linux instalada con el comando `$ lsb_release -a`. La instalación de las librerías y *drivers* se realiza de la siguiente manera:

```
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, necesitaremos descargar el firmware de BladeRF con `$ sudo bladeRF-install-firmware`. Para comprobar que la instalación ha sido correcta, podemos entrar en el intérprete en línea de comandos de bladeRF: `$ bladeRF-cli -p`.

Cada vez que se inicie una tarjeta BladeRF será necesario cargar la imagen FPGA. Esta se puede descargar desde la página oficial de Nuand: https://www.nuand.com/fpga_images/. Como nuestro dispositivo es una BladeRF xA9 necesitaremos el fichero 'hostedxA9-latest.rbf'. Se recomienda guardar esta imagen en la misma carpeta donde está alojado el fichero de configuración del eNB. De esta manera, para cargar la imagen FPGA:

```
$ bladeRF-cli -l openairinterface5g/configuration/bladeRF/hostedxA9-latest.rbf
```

De nuevo, para comprobar que la carga ha sido correcta, podemos entrar en la versión interactiva de nuestra BladeRF. La salida esperada es la siguiente:

```
$ bladeRF-cli -i
bladeRF> version

  bladeRF-cli version:        1.8.0-git-5a146b2a
  libbladeRF version:         2.4.1-git-5a146b2a

  Firmware version:           2.3.1
  FPGA version:               0.12.0 (configured by USB host)
```

Una vez se han instalado todos los *drivers* de la BladeRF, ya podemos pasar al último paso: compilar la BladeRF como eNB. Es importante recordar que la compilación fallará con Ubuntu 20 y *releases* posteriores. El sistema operativo instalado en esta máquina es Ubuntu 18.04.6 LTS.

```
$ sudo ./openairinterface5g/cmake_targets/build_oai -I
$ sudo ./openairinterface5g/cmake_targets/build_oai -c -x -w BLADERF --eNB
```

Los parámetros utilizados son los siguientes:

- -I: solo será necesario en la primera compilación
- -c: borra todos los archivos ya compilados
- -x: añade una función osciloscopio. Se desarrolla en la sección 3.8
- -w: nombre del *hardware*
- --eNB: crea el *environment lte-softmodem* para eNB

Ya está todo listo para lanzar una eNB. El comando es el siguiente:

```
$ sudo ./openairinterface5g/cmake_targets/lte_build_oai/build/lte-softmodem -
O ./openairinterface5g/configuration/bladeRF/enb-band7-5mhz.conf
```

Se puede apreciar que el único fichero que posibilita utilizar la tarjeta BladeRF como eNB es *enb-band7-5mhz.conf*. En el nombre ya se puede deducir que solo va a permitir transmitir con un ancho de banda de 5MHz, lo que corresponde a 25 PRB, como se vio en la Tabla 1. Este factor limitará bastante los resultados obtenidos.

Es posible recibir un error del tipo: '`Unable to acquire the dpkg frontend lock (/var/lib/dpkg/lock-frontend)`'. En este caso, simplemente es necesario introducir lo siguiente en una terminal:

```
sudo fuser -vki  /var/lib/dpkg/lock
sudo rm -f /var/lib/dpkg/lock
```

```
sudo dpkg --configure -a
sudo apt-get autoremove
```

Finalmente, para asegurar que el rendimiento de la red es el óptimo, se deben ajustar los parámetros de emisión y recepción de la bladeRF. Para ello, se van a ajustar la ganancia, el ancho de banda, la frecuencia de emisión y recepción y la tasa de muestreo. Se van a utilizar:

- $f_{Tx} = 2.68\ GHz$
- $f_{Rx} = 2.56\ GHZ$
- $G_{Tx} = 60\ dB$
- $BW = 5\ MHz$
- $f_S = 7.68\ MHz$

Nótese que no se incluye la ganancia en recepción G_{Rx}. Esto es debido a que la bladeRF incluye un control automático de ganancia (AGC) en recepción. Para establecer estos valores en la SDR:

```
$ sudo bladeRF-cli -i
bladeRF> set frequency tx 2680000000
bladeRF> set frequency rx 2560000000
bladeRF> set gain tx 60
bladeRF> set bandwidth 5000000
bladeRF> set samplerate 7680000
```

3.6 Instalación del gNB

En la fecha en la que se realizó el proyecto, el *software* para la ejecución de un gNB-NSA en OpenAirInterface se encontraba en desarrollo. En particular, se estaba desarrollando en una rama llamada *develop,* donde la última modificación de los ficheros era de pocos días. Es por esa razón que el lanzamiento del gNB es, todavía, demasiado inestable.

Para movernos a la rama *develop* se debe introducir el comando `$ git checkout develop`. En este trabajo se ha decidido clonar esta rama del repositorio en

una nueva carpeta llamada 'openairinterface_develop'. Es importante comprobar la rama de trabajo, que es posible comprobar gracias al comando `$ git branch`. Una vez dentro la rama *develop*, es necesario recompilar OAI con los ficheros de configuración del gNB, proceso similar al de la eNB. Estando en /openairinterface_develop/openairinterface5g/cmake_targets:

```
$ sudo ./build_oai -I
$ sudo ./build_oai -c -x -w BLADERF --gNB
```

Es importante destacar que se va a hacer uso de una segunda SDR BladeRF: una para el eNB y otra para el gNB. Al igual que ocurría con el eNB, el fichero de configuración se encuentra en *openairinterface_develop/openairinterface5g/ configuration/bladeRF*. En este caso, el fichero escogido es *gnb-band7-5mhz.conf.* La configuración de este es similar al de la eNB: solamente es necesario indicar el PLMN de nuestra red (901, 70), el *tracking_area_code* (7) y las direcciones IP de las interfaces involucradas, las cuales están especificadas en el diagrama de la Figura 17. Es necesario, además, establecer la variable *enable_x2* a *"yes",* tanto en el eNB como en el gNB. Una vez se han realizado todos estos pasos, se procede a lanzar un gNB con el comando:

```
$ sudo ./cmake_targets/ran_build/build/nr-softmodem -O
./configuration/bladeRF/gnb-band7-5mhz.conf
```

3.7 Conectar un UE comercial

Lo más importante de un UE comercial es la tarjeta SIM. Nuestro grupo de investigación utiliza tarjetas programables de la empresa Sysmocom, cuyo PLMN es 901 70. Además, también ofrecen información acerca de su clave Ki y de su Opcode, entre otros códigos que no son de utilidad en esta aplicación.

Con la base de datos usaremos, esencialmente, los siguientes comandos:

```
$ sudo oai-hss.dump-users
$ sudo oai-hss.add-users
```

Sirven, respectivamente, para conocer la información de los usuarios (o de un suscriptor en específico con la opción -I IMSI) y para añadir un nuevo usuario. Para conocer todos los detalles de la BBDD resulta interesante `$ sudo oai-hss.help`. En el ejemplo se muestra cómo añadir un usuario con un específico IMSI, una clave Ki, un Opcode y un nombre de APN. Es importante no dejar espacio entre la opción (por ejemplo, -k) y el valor a introducir.

```
$ sudo oai-hss.add-users -IIMSI -kKi -oOpcode -aoai.openair5G.eur
```

Estos cuatro valores son los únicos realmente necesarios para que la red reconozca un UE. Es importante que el APN sea 'oai.openair5G.eur', ya que en caso contrario la conexión no se establecerá. Por defecto, el Opcode se calcula automáticamente siguiendo la ecuación (2). Sin embargo, las tarjetas Sysmocom utilizadas poseen un Opcode fijo, el cual se ha especificado en la base de datos en el paso anterior. Por tanto, para su correcto funcionamiento, es necesario establecer la variable *reload_key* en *hss_rel14.json* a *false*. Se recuerda que para acceder a este fichero se puede introducir `$ sudo oai-hss.conf-get` en una terminal.

$$OPC = AES_{128}(k_i, OP) \oplus OP \tag{2}$$

Entonces, debemos asegurarnos de que el APN seleccionado se encuentra en la lista de APNs posibles del SPGW. En *spgwc.conf* debemos comprobar que el APN 'oai.openair5G.eur' se encuentra en la APN_LIST. Esta estará relacionada con la primera fila en IP_ADDRESS_POOL, de modo que las IP de los UE que conectemos estarán entre {12.1.1.2 - 12.1.1.128}. Al primer UE se le asignará la 12.1.1.2, al segundo la 12.1.1.3, al tercero la 12.1.1.4…

```
IP_ADDRESS_POOL : {
  IPV4_LIST = (
    {RANGE = "12.1.1.2 - 12.1.1.128";}, # Static IP address range
    {RANGE = "12.1.1.129 - 12.1.1.224";}, # Dynamic IP address range
    {RANGE = "192.169.0.2 - 192.169.255.253";},
    {RANGE = "192.170.0.2 - 192.170.255.253";},
    {RANGE = "192.171.0.2 - 192.171.255.253";}
  );
```

```
 IPV6_LIST = (
    {PREFIX = "2001:1:2::/64";},
    {PREFIX = "3001:1:2::/64";},
    {PREFIX = "4001:1:2::/64";},
  );
};

#REALM is must have
APN_LIST = (
  {APN_NI = "oai.openair5G.eur"; PDN_TYPE="IPv4";IPV4_POOL=0;IPV6_POOL=-1},
  {APN_NI = "internet";          PDN_TYPE="IPv4";IPV4_POOL=1;IPV6_POOL=-1},
  {APN_NI = "apn2";              PDN_TYPE="IPv4";IPV4_POOL=2;IPV6_POOL=-1},
  {APN_NI = "apn3";              PDN_TYPE="IPv4";IPV4_POOL=3;IPV6_POOL=-1},
  {APN_NI = "apn2";              PDN_TYPE="IPv4";IPV4_POOL=4;IPV6_POOL=-1}
);
```

Nótese que hay un comentario que dice '`REALM is must have`'. Es esta la razón por la que se mencionaba que el APN debe ser obligatoriamente 'oai.openair5G.eur'. Si este no es incluido en la APN_LIST, aparecerá un error del tipo '`Mandatory IE of type 93`', que causará un error con valor 78, el cual denota que el APN es desconocido o incorrecto [51]. No hay razón aparente para que sea obligatorio este nombre de APN, excepto que exista algún fichero de configuración donde pueda ser cambiado y que durante el desarrollo de este trabajo fin de grado no se ha logrado encontrar.

Comprobamos que la información de los suscriptores es correcta con el comando '`$ sudo oai-hss.dump-users`'. Si quisiéramos información detallada de uno de ellos, podríamos introducir '`$ sudo oai-hss.dump-users -I(IMSI) -s`'. En la Figura 20 se muestra la salida esperada del comando. Se recuerda que el número de secuencia sqn es aleatorio y se modifica cada vez que se conecta el UE.

```
artemis@artemis-ThinkPad-L380:~$ sudo oai-hss.dump-users
imsi            key                              opc                              mmehost                              mmerealm       sqn
-------------------------------------------------------------------------------------------------------------------------------------------
901700000052121   339E1393952281C83A6D96E2B44179EB   D28BF50E7704256AE23BE188707D2973   artemis-ThinkPad-L380.openair5G.eur   openair5G.eur   2614
901700000052123   4055D3C4EA2367FDD41AF9BAB4CC55C8   C8840A79D6313D69B9C76B3BCDAA520D   artemis-ThinkPad-L380.openair5G.eur   openair5G.eur   17781
901700000052122   1116B35021A7E0784D18F91D7E3BD3C5   81F19FDC9323293E5E2DAF1BD64FEBDB   artemis-ThinkPad-L380.openair5G.eur   openair5G.eur   4334
artemis@artemis-ThinkPad-L380:~$
```

Figura 20. Información de la base de datos

3.7.1 Configuración de red del CN

Una vez modificada la base de datos, la red es capaz de reconocer la tarjeta SIM del UE, pero no es capaz de acceder a Internet. Esto es debido a que la tabla de encaminamiento de la máquina que contiene el CN no está preparada para gestionar los paquetes provenientes del módem USB, el cual, por defecto, tiene asignada la dirección 12.1.1.12.

Es conveniente repasar, con perspectiva, cómo funciona la red que estamos desplegando. En la Figura 21 se muestra un diagrama de la red completa, incluyendo las direcciones IP de todas las interfaces. Para evitar confusiones, es importante señalar que, a pesar de que hayamos seleccionado la IP 12.1.1.2 para el UE, el módem Huawei utilizado asigna la dirección 192.168.8.100 al vehículo. Además, se aprecia el túnel GTP que crea el eNB: establece un enlace virtual directo entre el módem USB y el CN. Esa interfaz virtual que se genera se llama pdn, y será importante tenerlo en cuenta a la hora de modificar la tabla de encaminamiento del CN.

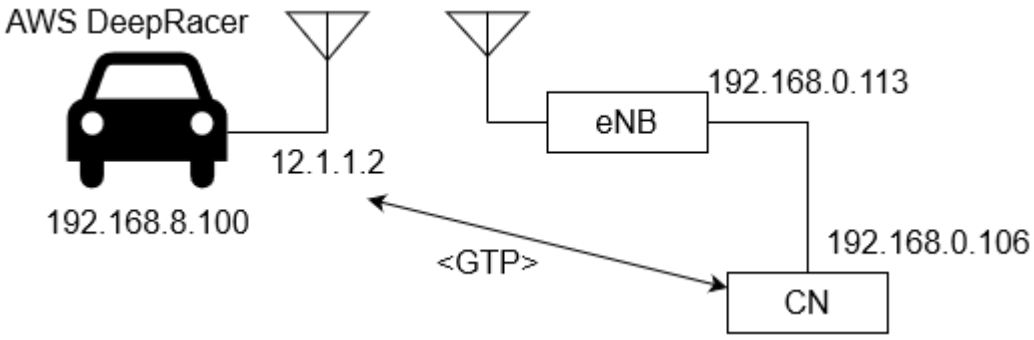

Figura 21. Diagrama de la red

Primero, se debe comprobar que el ordenador que soporta el CN tiene activado el *forwarding* (reenvío). Para comprobarlo es útil el comando:

```
$ sudo sysctl net.ipv4.ip_forward
```

La salida esperada es ‘net.ipv4.ip_forward = 1’, lo que significa que el *forwarding* está activado. Si la salida no fuese esa, necesitaríamos modificar el siguiente

fichero de configuración. Se eliminaría el comentario de la línea `# net.ipv4.ip_forward=1`.

```
$ sudo vim /etc/sysctl.conf
$ sudo sysctl -p
```

El siguiente paso es modificar la tabla de encaminamiento. Para ello, en la máquina que soporta el CN:

```
$ sudo iptables -I FORWARD -i pdn -s 12.1.1.0/24 -o wlp2s0 -d 0.0.0.0/0 -j ACCEPT
$ sudo iptables -I FORWARD -i wlp2s0 -s 0.0.0.0/0 -o pdn -d 12.1.1.0/24 -m conntrack --ctstate ESTABLISHED, RELATED -j ACCEPT
$ sudo iptables -t nat -A POSTROUTING -s 12.1.1.0/24 -o wlp2s0 -j MASQUERADE
```

Se puede comprobar la tabla de encaminamiento con `$ route -n`:

```
Kernel IP routing table
Destination     Gateway         Genmask      Flags  Metric Ref  Use  Iface
0.0.0.0        192.168.0.1    0.0.0.0         UG    600    0     0   wlp2s0
12.1.1.0        0.0.0.0       255.255.255.0   U     0      0     0   pdn
12.1.2.0        0.0.0.0       255.255.255.0   U     0      0     0   pdn
169.254.0.0     0.0.0.0       255.255.0.0     U     1000   0     0   wlp2s0
172.17.0.0      0.0.0.0       255.255.0.0     U     0      0     0   docker0
192.168.0.0     0.0.0.0       255.255.255.0   U     600    0     0   wlp2s0
192.169.0.0     0.0.0.0       255.255.255.0   U     0      0     0   pdn
192.170.0.0     0.0.0.0       255.255.255.0   U     0      0     0   pdn
192.171.0.0     0.0.0.0       255.255.255.0   U     0      0     0   pdn
```

Se observa que todos los paquetes provenientes de la red 12.1.1.0/24 serán reenviados por la interfaz pdn. Además, se configuran los paquetes salientes del CN para que el *flag* MASQUERADE esté en “on”

Por último, cabe destacar que esta configuración es temporal. Cada vez que se suspenda la máquina que soporta el CN la configuración se borrará. Para hacer que estos cambios sean permanentes [52]:

```
$ sudo apt-get install iptables-persistent
$ sudo iptables-save > /etc/iptables/rules.v4
```

3.7.2 Optimización del enlace

A pesar de que la conexión se pueda establecer, y la red funcione correctamente, es muy habitual que el rendimiento sea realmente pobre. Lo más probable es que se trate de la calidad del enlace radio. Para ello, se deben tener en cuenta diferentes factores:

- Ganancia de la SDR
- Ganancia de OpenAirInterface
- Posibles interferencias
- Antenas utilizadas

Parece lógico pensar que, si la calidad del enlace radio es insuficiente, la solución sea aumentar las ganancias. Sin embargo, es posible que un exceso en la potencia de la señal sature el *hardware*. Se recuerda que el control automático de ganancia de la bladeRF solo se encuentra activo en transmisión. En nuestro caso se han realizado varias pruebas y se ha encontrado que el mejor comportamiento se daba cuando se reducía la ganancia de OpenAirInterface a 100dB. Para ello se debe cambiar el parámetro *max_rxgain* del fichero de configuración del eNB.

En cuanto al entorno, las pruebas se han realizado en un laboratorio sin ningún tipo de aislamiento electromagnético. Aunque débil, es probable que la señal tenga problemas de interferencias. Se han utilizado antenas diseñadas para LTE, con una ganancia de 10dBi [53], colocadas perpendicularmente para evitar pérdidas por acoplo de polarización.

3.8 Resultados

En este apartado se definen las pruebas a las que ha sido sometida la red para evaluar su rendimiento. Se comentarán las diferentes herramientas que ofrece OpenAirInterface para ello, además de pruebas externas que se han realizado.

OpenAirInterface cuenta con dos herramientas de análisis de la red: una monitorización de los eNodeB por medio de *T Tracer* y una visión de osciloscopio *soft-scope*. En ellos se muestra información del enlace radio (físico, MAC, RLC, PDCP y RRC) [52].

3.8.1 Pruebas con 5G-NSA

Como ya se ha comentado en el apartado 3.6, el software que OAI proporciona para lanzar un gNB, y conseguir así una conexión 5G-NSA, se encuentra en desarrollo. Es por ello que no ha sido posible establecer una conexión entre un UE y un gNB de manera satisfactoria.

Para el desarrollo de las pruebas se siguió el modelo de la Figura 17 con dos máquinas físicas. En una de ellas se lanzó el CN, y en la otra el eNB y el gNB. La conexión entre estos tres elementos fue correcta, y funcionó sin problema. Sin embargo, en el momento en que se intentó agregar un UE que admitiese conexión 5G, saltaba un error en el gNB que hacía fallar toda la red.

Es por esto que todas las pruebas que se describan en posteriores subsecciones supondrán una red LTE, sin desplegar ningún gNB. Queda pendiente probar estas funcionalidades para futuros trabajos que continúen esta materia de estudio.

3.8.2 Monitorización de los eNB con T Tracer

La herramienta T Tracer viene instalada por defecto en OAI, y no es necesario realizar ninguna modificación en la compilación para activarla. Para lanzarla, es necesario iniciar un eNB con la opción –T_stdout 0. De esta manera, la ejecución del eNodeB no iniciará hasta que se lance un Tracer asociado. Después, se escribe en una terminal:

```
$ cd /openairinterface5g/common/utils/T/tracer
$ ./enb -d ../T_messages.txt
```

Tras la realización, se abrirá una ventana con la información del T Tracer acerca del enlace radio. La interfaz radio se divide en tres protocolos: la capa física L1, la capa de enlace L2 y la capa de red L3. Se muestra la pila de protocolos de la red de acceso UMTS en la Figura 22 para una mejor comprensión.

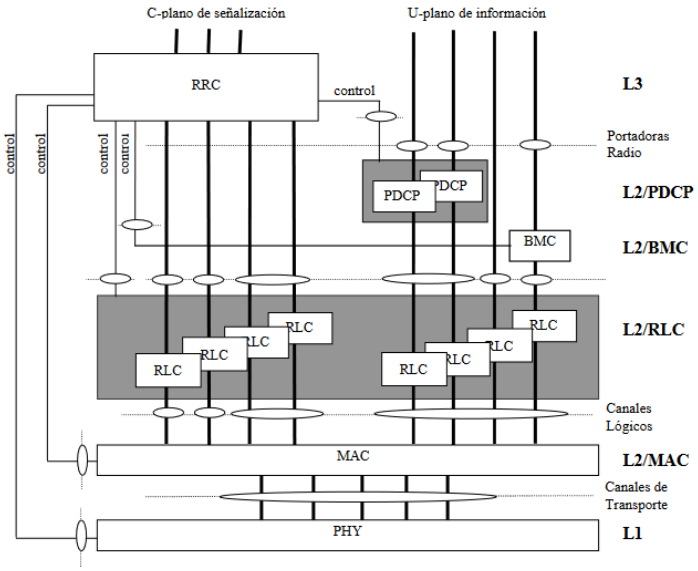

Figura 22. Pila de protocolos en la interfaz radio [54]

En la Figura 23 se observa la interfaz gráfica de la herramienta. Se ve que está dividida en diferentes regiones [55]:

- Parámetros del enlace físico (gráficas superiores). Se muestra la potencia de la señal de entrada, la constelación de la modulación empleada (16 QAM) y las velocidades binarias de los enlaces *uplink* y *downlink.*
- Tramas MAC (Media Access Control)
- Tramas RLC (Radio Link Control)
- Tramas PDCP (Packet Data Convergence Protocol)
- Tramas RRC (Radio Resource Control)

Las tramas MAC, RLC y PDCP son tres de las cuatro subcapas que forman el nivel de enlace L2. El protocolo RLC se encarga de corregir errores mediante ARQ

(*Automatic Repeat Query*), además de transferir las tramas MAC al nivel PDCP [56]. PDCP es el escalón más alto del nivel 2, y quien se comunica con el nivel L3. Su misión principal es la interpretación de cabeceras [54]. Finalmente, RRC es el protocolo principal del nivel L3, la capa más alta de todas y quien toma decisiones acerca de la gestión del enlace.

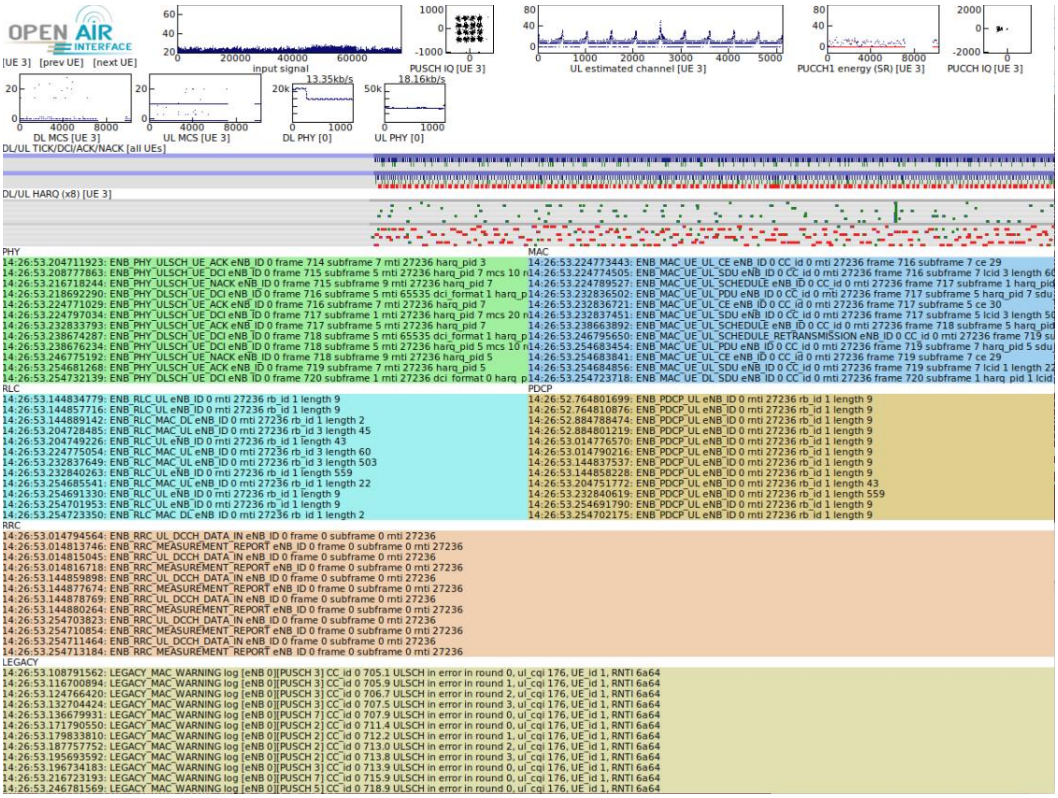

Figura 23. Vista de la herramienta T Tracer

3.8.3 Osciloscopio Soft-Scope

Además de T Tracer, OAI ofrece otra herramienta visual de análisis del enlace radio. A diferencia de esta última, para la instalación del *soft-scope* es necesario añadir el parámetro -x en la compilación del eNB (ver sección 3.5). Para lanzar el osciloscopio virtual es necesario añadir el parámetro -d en la ejecución del eNodeB:

```
$ sudo ./cmake_targets/lte_build_oai/build/lte-softmodem -d -O
./configuration/bladeRF/enb-band7-5mhz-conf
```

Una vez realizado, se abrirá una ventana con la interfaz gráfica de la herramienta. Cuando se haya establecido la conexión con un UE, la interfaz comenzará a dibujar gráficas, como se muestra en la Figura 24.

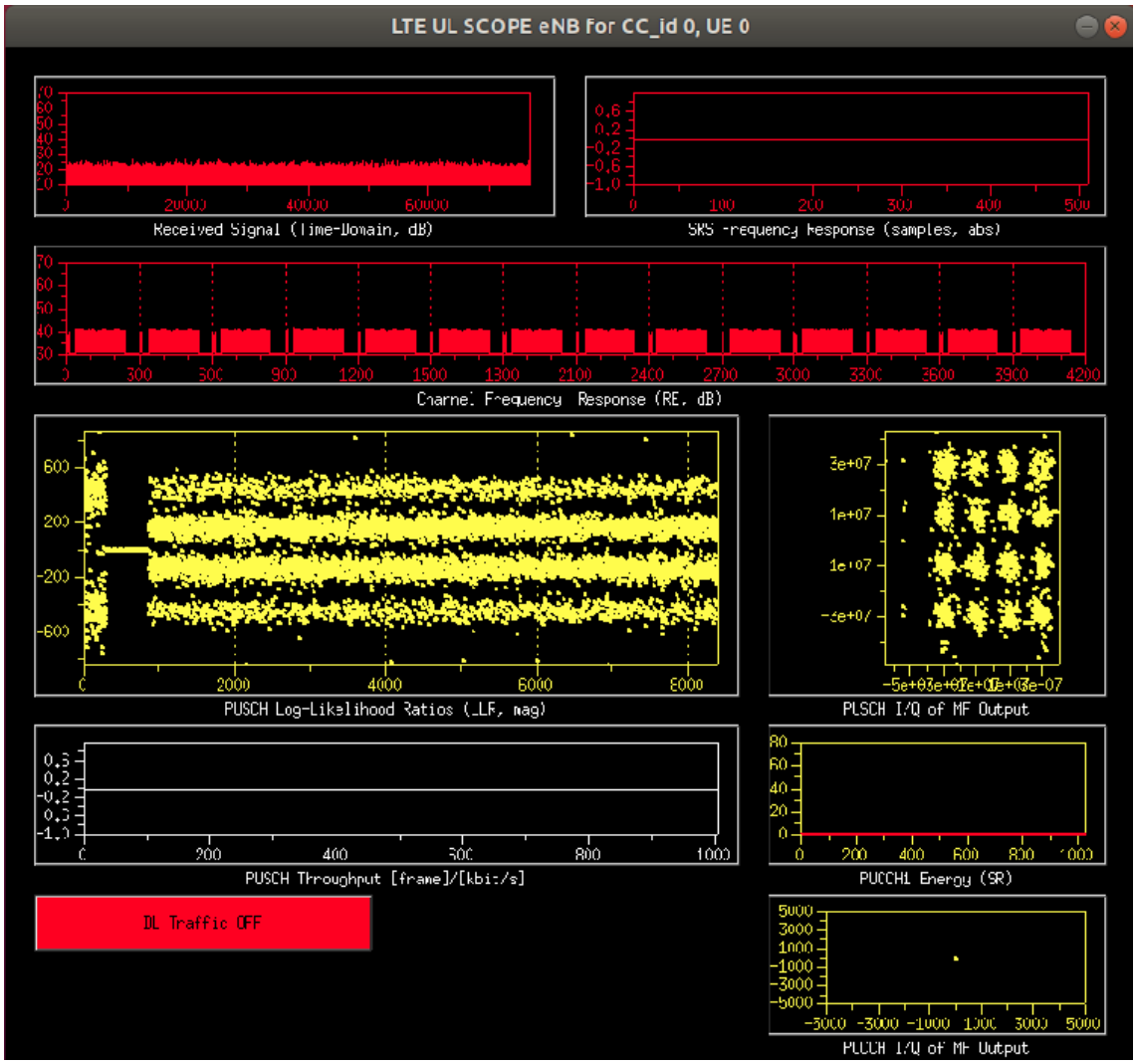

Figura 24. Interfaz gráfica soft-scope

Al igual que *T Tracer*, la interfaz del *soft-scope* está dividida en diferentes ventanas [47]:

- Señal recibida en el dominio temporal (dB)
- Respuesta frecuencial del canal (dB)
- PUSCH (Physical Uplink Shared Channel) Log-Likelihood Ratio
- Constelación del PUSCH

La información más relevante se encuentra en el PUSCH Log-Likelihood, la potencia de la señal recibida y la constelación del PUSCH. Se aprecia que la constelación es una 16 QAM y que el eNB no se encontraba transmitiendo, ya que la señal transmitida se mantiene fija alrededor de 30dB, el ruido de base. Se recuerda que esta herramienta solo es capaz de analizar el tráfico de *uplink.*

Para diagnosticar de manera visual y rápida el rendimiento de la red, la gráfica más útil es la LLR (Log-Likelihood Ratio). La primera gráfica representa las secuencias de bit recibidas. Como la constelación es una 16 QAM, hay cuatro bits por cada símbolo. Es por esa razón que en la gráfica se muestran cuatro líneas, una asociada a cada bit. En la Figura 25 se muestran las gráficas LLR para modulaciones 16 QAM y 64 QAM. Se observa que para la modulación 64 QAM existen 8 líneas, ya que en esta modulación cada símbolo está formado por 8 bits [57].

Para comprobar el estado de la red de un vistazo basta con mirar estas gráficas. Cuanto más separadas se encuentren las líneas, mejor se estarán demodulando los bits recibidos. En cambio, si existe mucho ruido en la gráfica y no es posible diferenciar claramente las diferentes líneas, se podrá deducir que la calidad del enlace radio no es la óptima.

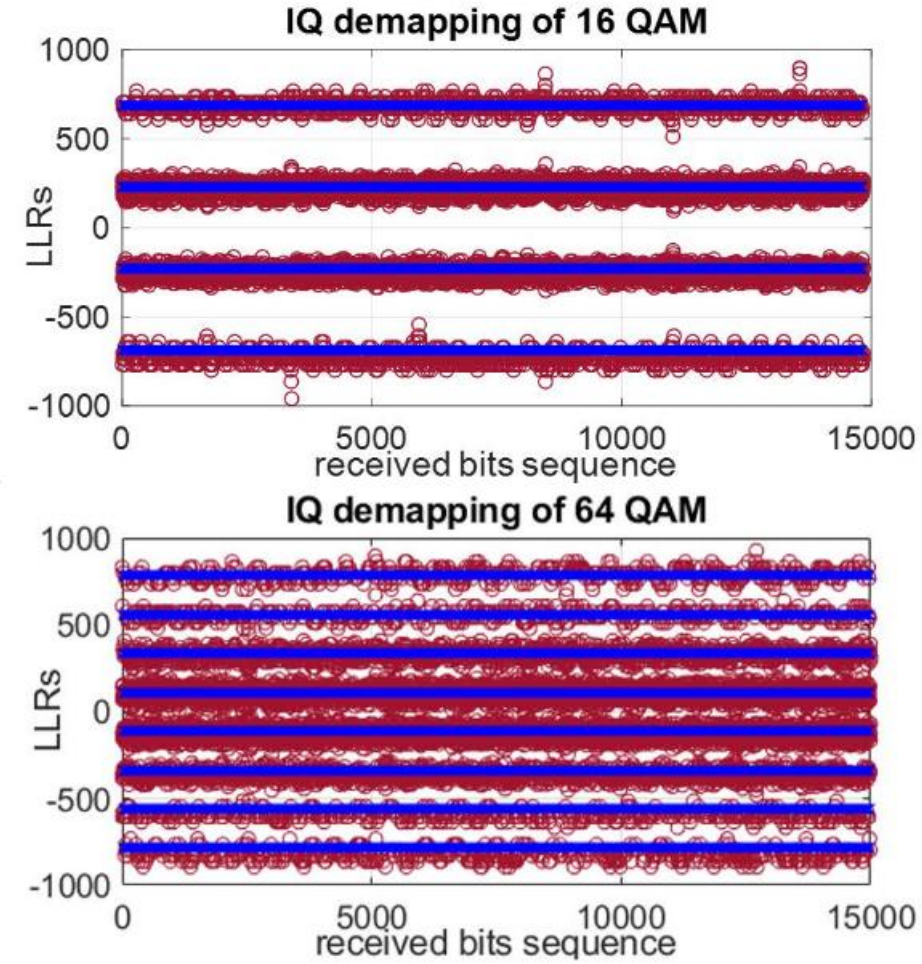

Figura 25. LLR para 16 QAM y 64 QAM [57]

3.8.4 Test de velocidad

Para medir la velocidad se ha utilizado el test de velocidad *online* de Movistar [58]. Se ha escogido este test porque es muy completo: ofrece tasas de transmisión en *uplink* y en *downlink*, además de medida de latencia y *jitter*.

Se han realizado pruebas de velocidad con los dos escenarios de implementación propuestos: con dos máquinas y con solo una. Aunque podría parecer que la implementación en dos ordenadores diferentes fuera a dar mejores resultados, ya que cada máquina tiene menos tareas que realizar, el desempeño de la red fue mejor cuando se utilizó solamente una máquina. Esto es debido a que el tiempo de procesado en el Intel Nuc es menor al tiempo de transmisión de la información de una máquina a otra.

Las mediciones de la Figura 26 se realizaron desplegando tanto el CN como el eNB en una única máquina. En ella se aprecia que la velocidad máxima es de 16Mbps en bajada y casi 8Mbps en subida. Los resultados son consistentes con la teoría. Como se aprecia en la Figura 24, cuando la calidad del enlace radio es buena, OAI permite transmitir con una modulación 16 QAM. Como el ancho de banda es de 5 MHz, la velocidad binaria máxima teórica R, según la ecuación (3), es de 20 Mbps. En la ecuación, M representa el número de símbolos utilizados en la modulación (en una modulación 16 QAM se definen 16 símbolos distintos).

$$R\ (bps) = BW\ (Hz) \log_2 M \tag{3}$$

Estos resultados permiten navegar por la red de manera cómoda. Sin embargo, el problema se encuentra en la latencia. Se ven 64 ms de latencia, lo cual es demasiado para el caso de uso a desarrollar. Se recuerda que el propósito de este proyecto es el procesamiento en tiempo real de la información de un AWS DeepRacer, por lo que reducir al mínimo la latencia entre la estación base y el vehículo es crucial.

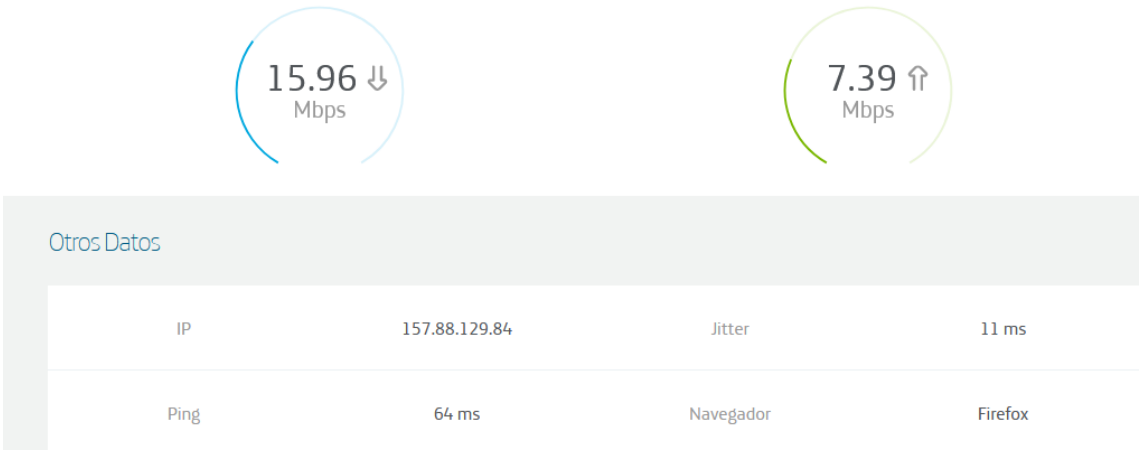

Figura 26. Test de velocidad

4. Reconocimiento de señales

4.1 Introducción

El segundo gran objetivo de este trabajo consiste en el desarrollo de un *software* que permita a un coche AWS DeepRacer reconocer señales de tráfico en tiempo real y reaccionar en consecuencia: si ve un *stop* debe parar, si en una intersección se le indica que es obligatorio girar a la derecha, que lo cumpla, etc.

Para ello, Ignacio Royuela en [9] desarrolló un *software* capaz de comunicar un ordenador con un AWS DeepRacer, de manera que este fuera capaz de procesar todos los datos necesarios. La arquitectura de la red se detalla en la Figura 27. En ella, se observan tres niveles: el servidor de aplicaciones, los vehículos y el bróker MQTT:

- El **servidor de aplicaciones**, simulado en el Intel NUC, contiene el código Python capaz de controlar el vehículo. Se definen variables globales como el acelerador o la constante de Stanley, necesaria para el cálculo de la trayectoria.
- El **vehículo** se comunica a la vez con el servidor de aplicaciones y con el bróker MQTT. Este envía las imágenes captadas por su cámara al servidor, y la información del vehículo (nivel de batería, niveles de calibración, etc.) al bróker.
- El **bróker MQTT**, simulado en el portátil ThinkPad, es quien media la comunicación entre el administrador y el vehículo. Permite enviar comandos al vehículo, con el fin de controlar el funcionamiento del mismo.

Originalmente, el único propósito de la red era la implementación de conducción autónoma en el vehículo. Se ha aprovechado que el servidor de aplicaciones ya separaba la imagen de la cámara del vehículo en fotogramas para introducir una función *sign_detection().* De esta manera se ha podido modularizar el *software,* separando los ficheros necesarios para la conducción autónoma del reconocimiento de señales.

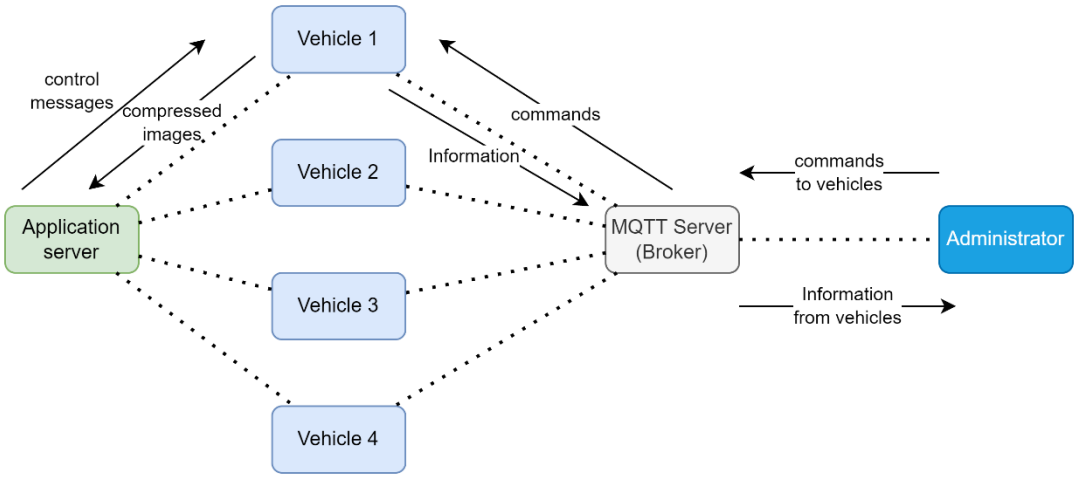

Figura 27. Arquitectura propuesta en el TFM de I. Royuela [9]

4.2 Implementación del algoritmo

El algoritmo utilizado se basa en el desarrollado en [59], un repositorio existente de GitHub. En él, Hoàng Lê Hái Thanh implementa en Python un algoritmo de *machine learning* capaz de reconocer señales de un vídeo. En este proyecto va a aprovechar el trabajo realizado en [59], adaptándolo a reconocimiento de señales en tiempo real. En esta sección se va a analizar el *software* original, se van a explicar las modificaciones realizadas y se comprobarán los resultados obtenidos.

4.2.1 Estructura del *software* original

El *software* original se componía de tres *scripts* escritos en Python y de una carpeta llamada *dataset.* A continuación, se explica el propósito de cada uno de los elementos del proyecto:

- ***common.py.*** Este archivo contiene funciones útiles necesarias para el correcto funcionamiento del programa. Entre ellas se encuentran comprobaciones de la versión de Python o la definición de los formatos aceptados para las imágenes que compondrán el *dataset.*
- ***classification.py.*** Este fichero contiene todo el modelo SVM, el cual se encarga del reconocimiento de las señales de tráfico. En él se encuentran las funciones que cargan el *dataset*, que aplican transformaciones a la imagen para que esta pueda ser procesada, que inicializan el modelo SVM y que lo evalúan.

- ***main.py.*** Archivo principal, el cual hay que ejecutar para lanzar la aplicación. En su versión inicial llamaba a un vídeo de ejemplo, el cual descomponía en fotogramas. Después, procesaba todos los *frames* para intentar encontrar señales en ellos. Si encuentra una señal, dibuja un rectángulo verde alrededor de la señal, además de escribir el nombre de esta. En la Figura 28 se muestra un ejemplo de funcionamiento sobre este vídeo de prueba.
- ***dataset.*** Carpeta que actúa como base de datos. Está compuesta por diferentes subcarpetas, numeradas del 0 al 12, de las cuales cada una corresponde a una señal de tráfico. Por ejemplo, todos los *STOP* se encuentran en la carpeta 1.

Figura 28. Ejemplo de reconocimiento en el vídeo de ejemplo

4.2.2 Modificaciones realizadas

4.2.2.1 Modificaciones sobre el dataset

El propósito de este trabajo es el de reconocer señales de tráfico en tiempo real. Por tanto, es necesario alterar ligeramente el código inicial. La primera modificación fue el aumento del *dataset*. El autor original solo tuvo en cuenta el vídeo de ejemplo para confeccionar el *dataset*. En la Figura 29 se muestra una visión general de la carpeta 2 del *dataset* original, la cual contiene imágenes de señales que obligan a girar a la izquierda. Es obvio que las imágenes están obtenidas del propio vídeo de prueba. Esto provoca un problema de *overfitting* [60]. Este efecto surge cuando un modelo de

machine learning está diseñado para resolver un problema muy concreto, y no es capaz de generalizar y resolver problemas similares. En este caso, el modelo SVM funciona muy bien sobre el vídeo de ejemplo, pero no es capaz de reconocer otras señales del mismo tipo.

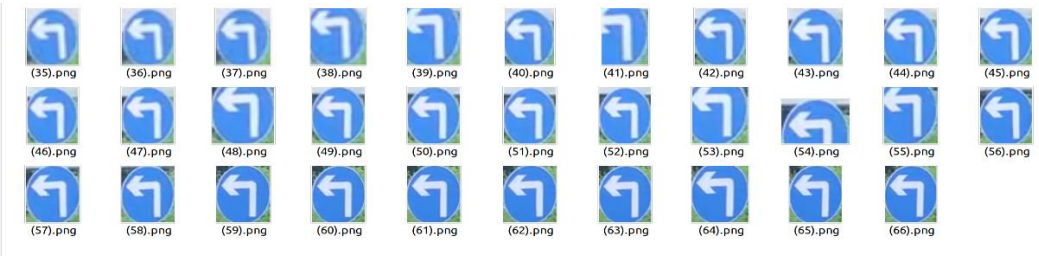

Figura 29. Ejemplo del dataset original

La solución a este problema es inmediata: se debía aumentar el *dataset* con imágenes de señales diferentes del mismo tipo. Sin embargo, el propósito de este proyecto no es generalista. El objetivo no es que el vehículo circule por carreteras reales y sea capaz de reconocer cualquier tipo de señal, sino que este debe reconocer un tipo concreto de señales dentro de un circuito establecido. Es por ello que, aunque se incluyeron imágenes de todo tipo de señales, se han incorporado también fotos de las señales que se utilizaron en la maqueta. Al igual que en [59], aquí también se ha provocado algo de *overfitting.* La Figura 30 muestra algunas de estas señales.

También, con objeto de mejorar el rendimiento, se ha reducido el número de categorías. Había señales que nunca se iban a utilizar, por lo que no era necesario sobrecargar el modelo. Eso solo hubiera incrementado innecesariamente el tiempo de entrenamiento, además de que podría dar lugar a errores. Se ha limitado el dataset a 8 categorías, dejando 7 para tipos de señales y una para “no señal”.

Finalmente, con todas estas modificaciones, se ha conseguido un dataset con un total de 2382 imágenes, pertenecientes a 8 categorías diferentes. Para conseguir las imágenes se han utilizado bases de datos públicas, como el dataset de reconocimiento de señales de tráfico alemanas [61]. Además, como ya se ha comentado, se han incluido fotos tomadas de las señales que se utilizarán en la maqueta final.

Figura 30. Señales empleadas en la maqueta

4.2.2.2 Modificaciones sobre el código

Después, fue necesario abordar la modificación más importante: conseguir que el procesado de las imágenes se hiciera en tiempo real. Para ello fue importante el *software* desarrollado en [9], el cual muestra las imágenes captadas por la cámara del vehículo en tiempo real. Entonces, aprovechando que el *software* original dividía el vídeo en fotogramas, se han podido combinar ambos códigos.

La solución implementada solo necesita entrenar al algoritmo al comienzo de la ejecución. Este proceso tarda, aproximadamente, 2,5 segundos. Este resultado se ha obtenido promediando diez entrenamientos del modelo, gracias al fragmento de código de la Figura 31. Nótese que se ha condensado todo el código necesario para el entrenamiento del algoritmo en la función *training()*. Podemos comprobar que SVM es un algoritmo lo suficientemente rápido para ser aplicado en tiempo real. En la sección 4.3 se analizarán los resultados de precisión obtenidos.

```
NUM_TRAININGS = 10
avg_array = []    # Array that will contain the measures

for i in range(NUM_TRAININGS):

    t = time.time()
    model = training()
    avg_array.append(time.time() -t)


average = sum(avg_array)/NUM_TRAININGS

print('It takes ' + str(round(average, 2)) + ' seconds to train the model')
```

Figura 31. Cálculo del tiempo de entrenamiento

En la Figura 32 se muestra la función *training()*, la cual se encarga de entrenar y evaluar el modelo. El código no ha sido modificado con respecto al original escrito por Hoàng Lê Hái Thanh en [59]. El algoritmo comienza barajando de manera aleatoria las imágenes del dataset. Después, se realiza un *deskew*, es decir, se aplica una rotación a las imágenes para que todas estén centradas y alineadas. Posteriormente, se obtiene el HoG, la característica necesaria para introducir en el algoritmo. Después, se realiza la separación de datos entre entrenamiento y test. Finalmente, se evalúa el modelo y se guarda en el archivo *data_svm.dat.*

Finalmente, se tuvo que cambiar la manera en que se reconocían las señales. Primero, el *software* convierte el fotograma original en una imagen binaria, es decir, compuesta solo por blanco y negro. Se ejemplifica este proceso en la Figura 34 a). Después, se filtra la imagen eliminando las componentes de pequeño tamaño, asociadas al ruido. Más tarde, se comprueba si alguno de los contornos resultantes en la imagen se asemeja a una circunferencia. Si es así, se llama a la función *findLargestSign()*, la cual es capaz de reconocer la circunferencia más grande (si hay más de una señal en el fotograma se analiza la más grande). Posteriormente, se comprueba si ese círculo encontrado se asemeja a una señal conocida. En ese caso, se le asigna la etiqueta correspondiente. Esta etiqueta es la que determina con qué señal se asocia la imagen capturada en el fotograma [59].

```python
def training():

    print('Loading data from data.png ... ')
    data, labels = load_traffic_dataset()
    print(data.shape)
    print('Shuffle data ... ')
    # Shuffle data
    rand = np.random.RandomState(10)
    shuffle = rand.permutation(len(data))
    data, labels = data[shuffle], labels[shuffle]

    print('Deskew images ... ')
    data_deskewed = list(map(deskew, data))

    print('Defining HoG parameters ...')
    # HoG feature descriptor
    hog = get_hog()

    print('Calculating HoG descriptor for every image ... ')
    hog_descriptors = []
    for img in data_deskewed:
        hog_descriptors.append(hog.compute(img))
    hog_descriptors = np.squeeze(hog_descriptors)

    print('Spliting data into training (90%) and test set (10%)... ')
    train_n=int(0.9*len(hog_descriptors))
    data_train, data_test = np.split(data_deskewed, [train_n])
    hog_descriptors_train, hog_descriptors_test = np.split(hog_descriptors, [train_n])
    labels_train, labels_test = np.split(labels, [train_n])


    print('Training SVM model ...')
    model = SVM()
    model.train(hog_descriptors_train, labels_train)

    print('Evaluating model...')
    evaluate_model(model, hog_descriptors_train, hog_descriptors_test, labels_test)


    print('Saving SVM model ...')
    model.save('data_svm.dat')
    return model
```

Figura 32. Función de entrenamiento del modelo

El código original no era capaz de ejecutarse: había un error en el cálculo de los contornos de las imágenes binarias. Ese error seguramente se debía a incompatibilidades entre las versiones de Python y OpenCV utilizadas en el trabajo original y en el presente proyecto. Para solucionarlo fue necesario modificar la función *localization()* del archivo *main.py*. En la Figura 33 se muestra la modificación que fue necesario aplicar para que el algoritmo funcionase correctamente. La primera línea obtiene los contornos de las imágenes binarias. La segunda línea permite que los contornos puedan ser utilizados en cualquier implementación de la biblioteca OpenCV. Finalmente, la tercera línea ordena los contornos en función de su área de manera descendente (gracias al parámetro `reverse = True`).

```
contours = cv2.findContours(binary_image, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
contours = imutils.grab_contours(contours)
contours = sorted(contours, key = cv2.contourArea, reverse=True)[:5]
```

Figura 33. Cálculo de los contornos

Figura 34. a) Imagen binaria b) Resultado final

4.3 Pruebas realizadas

Para realizar las pruebas correspondientes se ha montado una maqueta con un circuito por el cual el vehículo es capaz de circular. Como se aprecia en la Figura 35, esta maqueta está compuesta por diferentes piezas separables, permitiendo no solo almacenarla de una manera cómoda, sino también abriendo la posibilidad de diseñar un circuito más grande en el futuro. Además, se ha montado la maqueta en un laboratorio con unas condiciones lumínicas muy uniformes. En la sección 4.4 se comentará por qué es importante este detalle.

En esta maqueta se ha podido comprobar el funcionamiento de todo el trabajo completo: se tiene el vehículo mandando la información de sus sensores a un ordenador que hace las veces de un servidor MEC gracias a la red LTE desplegada con OAI.

En la Figura 36 se ven los ordenadores que se utilizaron para soportar todo el trabajo. El ordenador central fue el encargado de alojar el CN de la red LTE, además del *software* de reconocimiento de señales. Es por ello que, con objeto de balancear la carga computacional, se ha introducido un segundo ordenador que gestiona el eNB. Finalmente, el portátil actuó como *bróker* MQTT, mediando la comunicación entre el vehículo y el ordenador central.

Figura 35. Maqueta fabricada

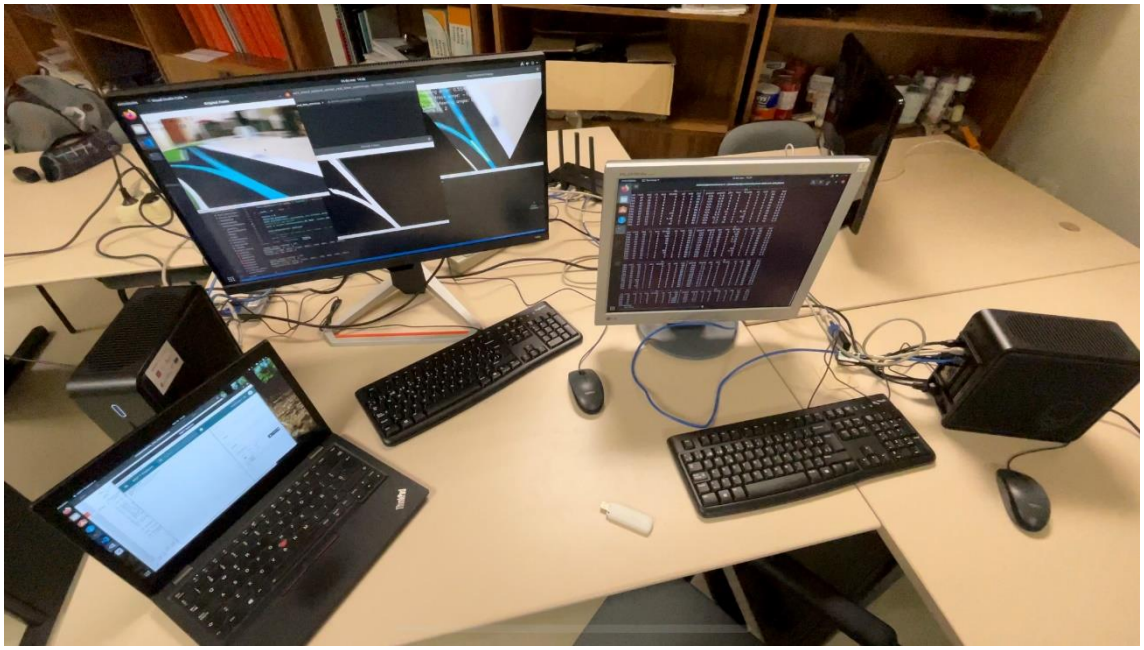

Figura 36. Arquitectura propuesta para la maqueta final

4.4 Resultados

4.4.1 Medidas de precisión

Es común, en la literatura, medir el desempeño de un algoritmo de clasificación por medio de la matriz de confusión. En ella, las columnas representan el valor esperado (cada una de las señales), y las filas, el valor predicho. De esta manera, si un dato de entrenamiento pertenece a la categoría 5 y el modelo le asigna la categoría 7, este dato se colocará en la posición (5, 7) de la matriz. Los *inputs* bien clasificados se situarán en la diagonal principal.

El *dataset* utilizado considera 7 tipos de señales, y reserva una etiqueta para "no señal". Por tanto, la matriz de confusión es una matriz cuadrada de orden 8. A continuación, en la Figura 37, se muestra la matriz de confusión normalizada por filas. Eso significa que, si en una fila aparece un 1, todas las señales de ese tipo fueron clasificadas de la misma manera. Se han redondeado los valores a dos decimales. El modelo comete errores detectando la señal de *stop* y 'prohibido girar a la izquierda'.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.04 & 0.96 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0.04 & 0 & 0 & 0 & 0.96 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Figura 37. Matriz de confusión

Ya se ha comentado que los *inputs* bien clasificados se sitúan en la diagonal principal. Entonces, el cálculo de la precisión del modelo se limita a sumar todos los valores de la diagonal principal y dividir ese resultado entre la suma de todos los campos de la matriz. Se muestra en la Figura 38 el código necesario para el cálculo de la precisión.

```
diag = np.trace(np.array(confusion_matrix))
sum = 0

for row in confusion_matrix:
    for element in row:
        sum += element

print('Accuracy: ' + str(round(100* diag/sum, 2)) + ' %')
```

Figura 38. Cálculo de la precisión

Se recuerda que en la sección 2.2.4 se comentó que en el kernel RBF de SVM había que ajustar dos parámetros: C y γ. En la aplicación del algoritmo, estos parámetros deben ser seleccionados en el archivo *classification.py*. Los valores que maximizaron la precisión fueron $C = 12$ y $\gamma = 0.02$. Tras aplicar todas las optimizaciones, el algoritmo ha alcanzado un 99.16 % de precisión.

Como se comentó en la sección 2.2.5, para entrenar el algoritmo es común separar el *dataset* en dos grupos: uno destinado para el entrenamiento y otro para testar

el rendimiento del algoritmo. En este algoritmo se destina un 90% del *datsaset* para entrenar el modelo y un 10% para comprobar la precisión de este.

Además, es posible realizar mediciones para comprobar cuánto tarda el algoritmo en reconocer una señal. Resulta interesante realizar este tipo de mediciones para evaluar de manera global el rendimiento de la aplicación. Para llevar a cabo estas métricas se ha utilizado el código mostrado en la Figura 39. Se observa que, en la línea 117, se evalúa si la variable *sign_type* es mayor que cero. Esto es debido a que se reserva la categoría '0' a objetos detectados que no son señales.

Al igual que con la precisión del algoritmo, se ha realizado una media con diez detecciones. El retardo obtenido fue de 22 ms, un valor aceptable para la aplicación propuesta.

```
112        # Traffic sign detection
113        t = time.time()
114        sign_type, result_image = detection(img, model)
115        elapsed = time.time() - t
116
117        if (sign_type > 0):
118            print('It took ' + str(elapsed) + 'seconds to recognize the signal')
```

Figura 39. Medición del tiempo de reconocimiento de una señal

4.4.2 Resultados generales

A pesar de que las métricas de precisión aseguren que el rendimiento del algoritmo es fantástico, esto no es enteramente cierto. Existe un gran problema que lastra la operación del trabajo de manera drástica. El algoritmo de reconocimiento de señales funciona perfectamente cuando el vehículo no está en movimiento. Sin embargo, cuando se han realizado pruebas en la maqueta final, el coche no es capaz de reconocer las señales.

Los dos principales factores, y por tanto aspectos a mejorar en trabajos futuros, son la resolución de la cámara y las variabilidades en la luz. Cuando el vehículo se encuentra en movimiento, especialmente en curvas, los fotogramas captados por la cámara no tienen la calidad necesaria como para que el algoritmo detecte las señales. Además, el *software* resulta ser muy sensible a variaciones en la luz. Cuando existen reflejos que enturbian las imágenes recibidas, el algoritmo tampoco es capaz de realizar correctamente su función.

Capítulo 5. Conclusiones y líneas futuras

Este trabajo está compuesto por dos partes muy diferenciadas: el despliegue de una red de datos móviles y el reconocimiento de señales. Es por ello que, en este último capítulo, se exponen las conclusiones de estas dos fases por separado. Se tratarán tanto conclusiones extraídas del trabajo, como líneas de trabajo futuras que podrían mejorar los resultados obtenidos.

5.1 Red móvil utilizando OpenAirInterface

En este proyecto se ha tratado de explorar el *software* OpenAirInterface para el despliegue de redes móviles 4G/5G. A pesar de que, en global, el resultado no es todo lo satisfactorio que prometía, se han podido extraer unas conclusiones interesantes:

- La arquitectura de la red es acertada. En la sección 3.2 se propusieron diferentes escenarios de implementación, virtualizando diferentes partes de la red. Queda presente que, para el caso de uso deseado, la combinación OAI CN + OAI eNB + UE físico es la óptima.
- La conexión establecida, tras un proceso de optimización, es estable y ofrece velocidades suficientes para aplicaciones cotidianas.
- Tanto el CN como el eNB son sencillos de configurar para adaptarlos a la arquitectura requerida (una sola máquina, dos máquinas, máquinas virtuales…)
- Gracias a la arquitectura propuesta, la red es escalable, permitiendo añadir más *eNodeB* si fuera necesario.
- Con la tarjeta BladeRF solo se pueden conseguir 25 PRB. Esto limita drásticamente el rendimiento con respecto al de una USRP.
- La base de datos de HSS, al estar contenida en un contenedor Docker, es muy rápida, sencilla de modificar y escalable.
- El gNB no es estable. No se han conseguido establecer conexiones satisfactorias 5G-NSA.

En resumen, OpenAirInterface no cumple con los requerimientos necesarios para cumplir el caso de uso propuesto. Aunque se ha conseguido establecer conexiones

estables, las tasas de transmisión son muy inferiores a las conseguidas con srsRAN por Ignacio Royuela en [9].

Sin embargo, hay determinados aspectos que se podrían mejorar. Sería interesante tener estos puntos en cuenta en futuros trabajos:

- Conseguir aumentar los PRB utilizando una BladeRF. Actualmente solo se definen ficheros de configuración con 25 PRB, y sería interesante diseñar un fichero nuevo que soporte 50 PRB o incluso 100 PRB. Modificar el ancho de banda no es una tarea trivial, y su consecución daría lugar a velocidades de transmisión mucho mayores.
- Siguiendo con el objetivo de conseguir velocidades de transmisión mayores, trabajos futuros deben buscar la manera de transmitir con una modulación 64 QAM. Tanto el estándar LTE como OpenAirInterface lo soportan, pero en las pruebas realizadas no se ha llegado a comprobar. Suponiendo 25 PRB, es decir, un ancho de banda de 5 MHz, la velocidad máxima teórica sería de 30 Mbps.
- Debido a que el software para lanzar el gNB se encontraba en desarrollo en el momento de redacción del presente trabajo, no se pudo evaluar la red 5G-NSA. Cuando se corrijan los errores del software, se debe testar el rendimiento de la red para ver si la mejora con respecto a LTE es sustancial.
- Implementar *handover*. Este proceso implica que un UE en movimiento pueda conectarse a un eNB diferente sin perder el servicio. En la aplicación requerida, el soporte de vehículos autónomos, establecer diferentes eNB puede ser de gran utilidad para aumentar el tamaño de la maqueta. Además, Ignacio Royuela en [9] tampoco consiguió implementarlo en srsRAN, por lo que sería un gran punto a favor de OAI.

5.2 Reconocimiento de señales

Para resolver la tarea del reconocimiento de señales en tiempo real se utilizó un algoritmo basado en *machine learning.* A diferencia de la red OAI, el resultado de esta

sección es muy satisfactorio. Aquí se enumeran las conclusiones extraídas de este trabajo:

- El algoritmo SVM ha resultado ser ideal para el reconocimiento de señales. Se ha conseguido una precisión muy cercana al 100% con un tiempo de entrenamiento de 2.5 segundos. Además, el modelo solo tarda 22 milisegundos en reconocer una señal.
- La aplicación no funciona correctamente cuando el vehículo se encuentra en movimiento. Las imágenes que capta la cámara están demasiado borrosas, y el algoritmo no es capaz de reconocer señales de tráfico en esas condiciones.
- Las condiciones de luz son demasiado determinantes. Los reflejos dan lugar a demasiadas confusiones, tanto pasando por alto algunas señales como inventándose otras donde no las hay. La solución más sencilla consiste en trabajar con ambientes con luz uniforme.
- Para que el vehículo reconozca de manera clara las señales, este debe circular a muy baja velocidad. Cuando el coche va demasiado rápido, las señales no llegan a ser procesadas correctamente.

Al igual que con la red OpenAirInterface, se proponen algunos aspectos a investigar en trabajos futuros:

- Se podría optimizar aún más el algoritmo. Actualmente, debido al algoritmo escogido para el reconocimiento de señales, no se tiene en cuenta el color de estas. Se podría buscar un algoritmo que, además de distinguir las formas, también se apoyase en los colores.
- Actualmente, el algoritmo solo considera siete tipos de señal de tráfico. En futuras investigaciones se debería incrementar este número, posibilitando al vehículo reconocer cualquier tipo de señal.
- Además, el vehículo debe ser capaz de reaccionar a estas señales. Por el momento, el algoritmo solamente reconoce las señales, pero no las interpreta. Debido a la sensibilidad del *software* a los reflejos de luz, implementar esta funcionalidad no es una tarea tan sencilla como se presupone.

- Se podría utilizar otro lenguaje de programación cuya ejecución sea más veloz. Actualmente se utiliza Python, un lenguaje interpretado. Se eligió este lenguaje por su sencillez, pero en aplicaciones en tiempo real suele ser más interesante apostar por lenguajes compilados, como C o C++.
- Ya se ha comentado que los reflejos suponen un problema muy importante. Sería interesante realizar algún tratado digital de los fotogramas captados por la cámara para eliminar parcial o totalmente los reflejos.
- Podría cambiarse el modo de detección de las señales. Podría ser valioso estudiar las redes neuronales convolucionales, ya que no están tan exploradas en la literatura. En teoría, una vez optimizados los pesos del modelo, el rendimiento promete ser esperanzador.

Referencias

[1] A. T. Harry Holma, «WCDMA technology and deployment status,» de *HSDPA/HSUPA for UMTS*, 2006.

[2] S. M. M. R. Indrawati, «A New Conceptual Model of Mobile Multimedia Services (MMS) and 3G Network Adoption in Indonesia,» *International Journal of Information, Science and Management,* 2014.

[3] «Banda Ancha,» [En línea]. Available: https://bandaancha.eu/articulos/cuantos-gb-datos-moviles-necesitas-9995. [Último acceso: 3 Junio 2022].

[4] M. W. J. Z. J. Z. M. H. T. X. X. Y. Wenjie Yang, «Narrowband Wireless Access for Low-Power Massive Internet of Things: A Bandwidth Perspective,» *IEEE,* vol. 24, nº 3, 2017.

[5] A. S. A. A. S. A. Nawaf Alharbe, «Application of ZigBee and RFID Technologies in Healthcare in Conjunction with the Internet of Things,» *Association of Computing Machinery,* pp. 191-195, 2013.

[6] K.-L. A. Y. C. W. Najmul Hassan, «Edge Computing in 5G: A Review,» *IEEE,* vol. 7, 2019.

[7] L. D. X. S. Z. Shancang Li, «The Internet of Things: A Survey,» *Springer Science,* 2014.

[8] «Xataka,» [En línea]. Available: https://www.xataka.com/internet-of-things/edge-computing-que-es-y-por-que-hay-gente-que-piensa-que-es-el-futuro. [Último acceso: 7 Junio 2022].

[9] I. Royuela, *Diseño e Implementación de un testbed de Edge Computing para el soporte de vehículos conectados,* Valladolid: Universidad de Valladolid, 2022.

[10] Amazon, «Amazon,» [En línea]. Available: https://www.amazon.com/-/es/53-023817/dp/B081DP5N9L. [Último acceso: 6 Junio 2022].

[11] 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].

[12] Nuand, «bladeRF2.0 micro xA9,» [En línea]. Available: https://www.nuand.com/product/bladerf-xa9/. [Último acceso: 4 Marzo 2022].

[13] Sysmocom, «Sysmocom SIM / USIM / ISIM cards,» [En línea]. Available: https://sysmocom.de/products/sim/sysmousim/index.html. [Último acceso: 2 Mayo 2022].

[14] Amazon, «AWS Deep Racer,» [En línea]. Available: https://aws.amazon.com/es/deepracer/. [Último acceso: 23 Julio 2022].

[15] Slamtec, «RPLIDAR A1,» [En línea]. Available: https://www.slamtec.com/en/Lidar/A1. [Último acceso: 4 Marzo 2022].

[16] «PcComponentes,» [En línea]. Available: https://www.pccomponentes.com/huawei-e3372h-320-modem-4g-usb?utm_source=176013&utm_medium=afi&utm_campaign=es.redbrain.shop&awc=20981_1653562145_5aaf996d2d813527f238e866eb131cf7&utm_term=deeplink&utm_content=es.redbrain.shop. [Último acceso: 26 Mayo 2022].

[17] S. M. S. M. P. K. Pattnaik, «A State of Art: Future Possibilitites of 5G with IOT and other Challenges,» de *Smart Helathcare Analytics in IOT Enabled*

Environment, 2020.

[18] ITU. [En línea]. Available: https://www.itu.int/es/ITU-T/focusgroups/imt-2020/Pages/default.aspx. [Último acceso: 10 May 2022].

[19] «Moniem-Tech,» [En línea]. Available: https://moniem-tech.com/2019/11/01/epc-core-upgrade-for-5g-en-dc-support/. [Último acceso: 7 Junio 2022].

[20] W. Telecom, «LTE Introduction,» IEEE, 2013. [En línea]. [Último acceso: 10 May 2022].

[21] R. A. Comes, LTE: Nuevas tendencias en comunicaciones móviles, Fundación Vodafone España, 2010.

[22] A. Narayanan, X. Zhang, R. Zhu, A. Hassan y S. Jin, «A Variegated Look at 5G in the Wild: Perfomance, Power and QoE Implications,» p. 625, 2021.

[23] 3GPP, System Architecture for the 5G System (3GPP TS 23.501 version 15.3.0 Release 15), 2018.

[24] E. T. T. M. Marius Corici, «An Organic 6G Core Network Architecture,» de *Conference on Innovation in Clouds, Internet and Networks (ICIN)*, 2022.

[25] C. F. T. N. K. K. F. J. ,. A. N. J. K. J. P. J. Ashkan Yousefpour, «All one needs to know about fog computing and related edge computing paradigms: A complete survey,» *Journal of Systems Architecture*, pp. 289-330, 2019.

[26] S. G. Joydeep Acharya, «Edge compression of GPS data for mobile IoT,» de *IEEE Fog World Congress (FWC)*, 2017.

[27] K.-L. A. Y. C. W. Najmul Hassan, «Edge Computing in 5G: A Review,» *IEEE Access*, vol. 7, 2019.

[28] S. Ray, «A Quick Review of Machine Learning Algorithms,» Faridabad, India, 2019.

[29] H. Reese, «Understanding the differences between AI, Machine Learning and Deep Learning,» *Tech Republic*, 2017.

[30] A. Wolfewicz, «Levity,» [En línea]. Available: https://levity.ai/blog/difference-machine-learning-deep-learning. [Último acceso: 7 Junio 2022].

[31] N. T. A. S. Amanpreet Singh, «A review of supervised machine learning algorithms,» de *3rd International Conference on Computing for Sustainable Global Development*, New Delhi, India, 2016.

[32] H. S. D. a. G. K. M. Pillutla, «Improved Face Recognition Rate Using Hog Features and SVM Classifier,» *IOSJR Journal of Electronics and Communications Engineering*, vol. 11, nº 4, pp. 34-44, 2016.

[33] «Programmer click,» [En línea]. Available: https://programmerclick.com/article/5862958789/. [Último acceso: 7 Junio 2022].

[34] S. U. N. M. K. Vairalkar, «Edge Detection of Images using Sobel Operator,» *International Journal of Emerging Technology and Advanced Engineering*, vol. 2, nº 1, 2012.

[35] B. T. Navneet Dalal, «Histograms of Oriented Gradients for Human Detection,» *IEEE*, 2005.

[36] Ashish, «Data Driven Investor,» [En línea]. Available: https://medium.datadriveninvestor.com/understanding-edge-detection-sobel-operator-2aada303b900. [Último acceso: 7 Junio 2022].

[37] W. S. Noble, «What is a Support Vector Machine?,» *Nature Biotechnology*,

vol. 24, pp. 1565-1567, 2006.

[38] K. D. A. S. Vinod Kumar Chauhan, «Problem formulations and solvers in linear SVM: a review,» *Springer Science+Business Media B.V.,* 2018.

[39] A. Saini, «Analytics Vidhya,» [En línea]. Available: https://www.analyticsvidhya.com/blog/2021/10/support-vector-machinessvm-a-complete-guide-for-beginners/. [Último acceso: 17 May 2022].

[40] D. S. C. Arti Patle, «SVM Kernel Functions for Classification,» *IEEE*, 2013.

[41] C. Q. Han Shunjie, «Parameter selection in SVM with RBF kernel function,» *IEEE,* 2012.

[42] J. E. T. Akinsola, «Supervised Machine Learning Algorithms: Classification and Comparison,» *International Journal of Computer Trends and Technology,* vol. 48, 2017.

[43] S. Chatterjee, «Detection of Myopathy and ALS Electromiograms Employing Modified Window Stockwell Transform,» *IEEE. Sensors Letters,* 2019.

[44] «OpenAirInterface,» [En línea]. Available: https://openairinterface.org. [Último acceso: 23 Mayo 2022].

[45] «Eurecom,» [En línea]. Available: https://www.eurecom.fr/en/. [Último acceso: 23 Mayo 2022].

[46] «Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Eurecom. [Último acceso: 23 Mayo 2022].

[47] X. Xiang, *Estudio de plataformas SDR para LTE-5G,* Universidad de Cantabria, 2018.

[48] O. A. Interface, «Gitlab EURECOM,» [En línea]. Available: https://gitlab.eurecom.fr/mosaic5g/mosaic5g/-/wikis/tutorials/. [Último acceso: 26 May 2022].

[49] G. Artemis, «Github,» 2022. [En línea]. Available: https://github.com/gcoUVa/Artemis/tree/base-station-oai/base-station-OAI. [Último acceso: 23 Julio 2022].

[50] «Packages Ubuntu,» [En línea]. Available: https://packages.ubuntu.com/bionic/linux-lowlatency. [Último acceso: 26 Mayo 2022].

[51] ETSI, «Universal Mobile Telecommunications System (UMTS)». 01 2012.

[52] S. P. Díaz, *Despliegue de prototipo de red LTE con OpenAirInterface y NextEPC: extensión de red y ampliación de funcionalidades,* Universidad Politécnica de Madrid, 2019.

[53] Huawei, «Amazon,» [En línea]. Available: https://www.amazon.es/Externa-Amplificador-Seguridad-Conector-Unidades/dp/B081ZS3SNX/ref=sr_1_9?__mk_es_ES=%C3%85M%C3%85%C5%BD%C3%95%C3%91&crid=1FHHFP0T8M4LD&keywords=huawei%2Blte%2Bantena%2B10dbi&qid=1654191641&sprefix=huawei%2Blte%2Bantena%2B10dbi%2Caps. [Último acceso: 2 Junio 2022].

[54] UPM, «Arquitectura de Protocolos en la Red de Acceso UMTS,» [En línea]. Available: http://web.dit.upm.es/~david/TAR/trabajos2002/02-Arquitectura-red-acceso-UMTS-Carlos-Diaz-Motero-res.pdf. [Último acceso: 23 Julio 2022].

[55] OpenAirInterface, «GitLab,» [En línea]. Available: https://gitlab.eurecom.fr/oai/openairinterface5g/wikis/T/enb. [Último acceso: 31 Mayo 2022].

[56] 3GPP, «RLC Protocol Specification (3G TS 25.322 version 1.1.0)». 1996.

[57] T. S. T. H. Khodr A. Saaifan, «NR MIMO Feature Implementation into OpenAirInterface,» *WSA*, 2021.

[58] Movistar, «Test velocidad,» [En línea]. Available: https://www.movistar.es/particulares/test-de-velocidad/. [Último acceso: 8 Junio 2022].

[59] H. L. H. Thanh, «Traffic Sign Detection - GitHub,» [En línea]. Available: https://github.com/hoanglehaithanh/Traffic-Sign-Detection. [Último acceso: 8 Junio 2022].

[60] X. Ying, «An Overview of Overfitting and its Solutions,» *Journal of Physics: Conference Series*, 2019.

[61] I. f. Neuroinformatik, «INI benchmark,» [En línea]. Available: https://benchmark.ini.rub.de/. [Último acceso: 23 Julio 2022].

[62] P. P. S. a. D. S. Shah, «A Review of Machine Learning and Deep Learning Applications,» *IEEE*, 2018.

[63] D. A. P. a. D. M. Schnyer, «Support Vector Machine,» de *Machine Learning: Methods and Applications to Brain Disorders*, 2020.

[64] S. W. X. H. H. L. Ivan Stojmenovic, «An overview of Fog computing and its security issues,» *Wiley Online Library*, 2015.

[65] Y.-C. C. Y.-H. T. a. J.-S. Y. S. Singh, «Mobile edge Fog computing in 5G era: Architecture and Implementation,» *Proc. Int. Comput. Symp. (ICS)*, 2016.

[66] Eclipse, «Mosquitto,» [En línea]. Available: https://mosquitto.org. [Último acceso: 29 Mayo 2022].

[67] Z. B. Pavel Mach, «Mobile Edge Computing: A Survey on Architecture and Computation Offloading,» *IEEE Communications, Surveys and Tutorials*, vol. 19, nº 3, 2017.

[68] GCO, «Github del GCO,» [En línea]. Available: https://github.com/gcoUVa/Artemis/tree/main/4g-network. [Último acceso: 7 Junio 2022].

[69] G. V. K. A. J. C. Y. F. W. F. Fabio Giust, «MEC Deployments in 4G and Evolution Towards 5G,» *ETSI White Paper*, February 2018.