

PROYECTO FIN DE GRADO

TÍTULO: PREDICCIÓN DE LA DURACIÓN DE SESIONES DE COMUNICACIONES V2V
MEDIANTE APRENDIZAJE PROFUNDO

AUTOR/A: DANIEL LINARES CASTILLO

TITULACIÓN: GRADO EN INGENIERÍA DE SISTEMAS DE TELECOMUNICACIÓN

TUTOR/A: JUAN MORENO GARCÍA-LOYGORRI

DEPARTAMENTO: INGENIERÍA AUDIOVISUAL Y COMUNICACIONES

VºBº TUTOR/A

Miembros del Tribunal Calificador:

PRESIDENTE/A: MARÍA CRISTINA GUERRERO GARCÍA

TUTOR/A: JUAN MORENO GARCÍA-LOYGORRI

SECRETARIO/A: PEDRO GARCÍA DEL PINO

Fecha de lectura: 17 DE MARZO DE 2025

Calificación:

El Secretario/La Secretaria,

Resumen

En la actualidad, la Inteligencia Artificial (IA) y el aprendizaje automático o *Machine Learning* (ML) están revolucionando el sector de las comunicaciones entre vehículos. Estos avances tecnológicos permiten optimizar la gestión del tráfico, mejorar la seguridad y hacer posible la conducción autónoma. Las comunicaciones entre vehículos (V2V, *Vehicle-to-Vehicle*), con la infraestructura (V2I, *Vehicle-to-Infrastructure*) y con el entorno en general (V2X, *Vehicle-to-Everything*) son fundamentales en el desarrollo de sistemas de transporte inteligente. La aplicación de modelos predictivos basados en ML y aprendizaje profundo o *Deep Learning* (DL) facilitan la optimización de la conectividad en entornos urbanos complejos.

El objetivo principal del proyecto es desarrollar un modelo capaz de predecir el tiempo de duración de las sesiones de comunicación entre vehículos (V2V) mediante la aplicación de técnicas avanzadas de ML y DL. Este proyecto busca optimizar la eficiencia y fiabilidad de las comunicaciones vehículo a vehículo (V2V), fundamentales en los sistemas de transporte inteligente y la conducción autónoma.

El proyecto se ha centrado en la simulación de escenarios urbanos de la ciudad de Madrid, utilizando el simulador de tráfico SUMO (*Simulation of Urban Mobility*), para generar entornos realistas de movilidad vehicular. Los datos obtenidos de estas simulaciones se emplearon para construir modelos predictivos que estimen la duración de las sesiones de comunicación en las comunicaciones V2V.

A nivel tecnológico, se han empleado herramientas como SUMO y Python (con la librería *TraCI*) para la generación y extracción de datos en tiempo real. Se hizo uso de software de código abierto, lo que redujo los costes económicos. Ambientalmente, se analizaron escenarios con y sin semáforos, evidenciando cómo la regulación del tráfico influye en la eficiencia de las comunicaciones V2V y la fluidez del tráfico. Se integraron sensores de tráfico estratégicamente ubicados en intersecciones clave para recolectar datos como velocidad, ruta, carril, y distancia a los cruces, así como el número de vehículos que se encontraban en un radio de 50 metros de cada punto de medición.

Posteriormente, se modeló la capa física de las comunicaciones, considerando los factores necesarios para reflejar condiciones realistas del entorno urbano. Esto permitió la generación de un dataset consolidado, incorporando información clave para el análisis y la predicción de la duración de las sesiones de comunicación.

Se han desarrollado y evaluado diversos modelos predictivos para estimar la duración de las sesiones de comunicación: regresión lineal múltiple, Máquinas de Soporte Vectorial o *Support Vector Regression* (SVR), árboles de decisión, bosques aleatorios o *random forest*, Red Neuronal Artificial (ANN). Cada modelo ha sido entrenado y evaluado utilizando diferentes métricas de rendimiento, las cuales se explican en detalle en la memoria. Esto permitió determinar su precisión y eficacia.

Abstract

Today, Artificial Intelligence (AI) and Machine Learning (ML) are revolutionising the vehicle-to-vehicle communications sector. These technological advances are optimising traffic management, improving safety and enabling autonomous driving. Communications between vehicles (V2V, Vehicle-to-Vehicle), with the infrastructure (V2I, Vehicle-to-Infrastructure) and with the environment in general (V2X, Vehicle-to-Everything) are fundamental in the development of intelligent transport systems. The application of predictive models based on ML and Deep Learning (DL) facilitates the optimisation of connectivity in complex urban environments.

The main objective of the project is to develop a model capable of predicting the time duration of vehicle-to-vehicle (V2V) communication sessions by applying advanced ML and DL techniques. This project aims to optimise the efficiency and reliability of vehicle-to-vehicle (V2V) communications, which are fundamental in intelligent transport systems and autonomous driving.

The project has focused on the simulation of urban scenarios in the city of Madrid, using the SUMO traffic simulator to generate realistic vehicular mobility environments. The data obtained from these simulations were used to build predictive models that estimate the duration of communication sessions in V2V communications.

At the technological level, tools such as SUMO and Python (with the TraCI library) were used to generate and extract data in real time. Open source software was used, which reduced economic costs. Environmentally, scenarios with and without traffic lights were analysed, showing how traffic regulation influences the efficiency of V2V communications and traffic flow. Strategically placed traffic sensors were integrated at key intersections to collect data such as speed, route, lane, and distance to junctions, as well as the number of vehicles within a 50-metre radius of each measurement point.

Subsequently, the physical communications layer was modelled, taking into account the factors necessary to reflect realistic conditions of the urban environment. This allowed the generation of a consolidated dataset, incorporating key information for the analysis and prediction of the duration of communication sessions. Several predictive models have been developed and evaluated to estimate the duration of communication sessions: multiple linear regression, Support Vector Regression (SVR), decision trees, random forest, Artificial Neural Network (ANN). Each model has been trained and evaluated using different performance metrics, which are explained in detail in the report. This allowed us to determine their accuracy and efficiency.

Índice de figuras

<i>Figura 1: Sistemas de transporte inteligente</i>	<i>5</i>
<i>Figura 2: Comunicación V2X.....</i>	<i>6</i>
<i>Figura 3: Comunicación V2V</i>	<i>8</i>
<i>Figura 4: Rango de frecuencias.....</i>	<i>9</i>
<i>Figura 5: Fenómenos LoS y NLoS</i>	<i>12</i>
<i>Figura 6: Escenario de propagación NLoS</i>	<i>14</i>
<i>Figura 7: Efectos del canal sobre la señal de propagación</i>	<i>17</i>
<i>Figura 8: Diferencias entre TCP y UDP</i>	<i>20</i>
<i>Figura 9: Comando para convertir formatos de archivos</i>	<i>29</i>
<i>Figura 10: Comando para añadir estructuras.....</i>	<i>30</i>
<i>Figura 11: Comandos para generar tráfico aleatorio</i>	<i>31</i>
<i>Figura 12: Contenido del archivo de configuración</i>	<i>32</i>
<i>Figura 13: Mapa de la Puerta de Alcalá</i>	<i>32</i>
<i>Figura 14: Comandos para generar el tráfico en Puerta de Alcalá</i>	<i>33</i>
<i>Figura 15: Comandos para aumentar el tráfico en Puerta Alcalá</i>	<i>33</i>
<i>Figura 16: Mapa de Sanchinarro</i>	<i>34</i>
<i>Figura 17: Mapa de Las Rosas</i>	<i>35</i>
<i>Figura 18: Comandos para generar el tráfico en Las Rosas.....</i>	<i>36</i>
<i>Figura 20: Comandos para generar el tráfico en A-2</i>	<i>37</i>
<i>Figura 19: Mapa de la A-2</i>	<i>37</i>
<i>Figura 21: Mapa de ampliado de A-2, Guadalajara</i>	<i>38</i>
<i>Figura 22: Mapa de A-2 ampliado, Valdeňoches</i>	<i>39</i>
<i>Figura 23: Información extraída de cada vehículo.....</i>	<i>40</i>
<i>Figura 24: Modelo de Regresión Lineal Múltiple</i>	<i>45</i>
<i>Figura 25: Modelo SVR.....</i>	<i>46</i>
<i>Figura 26: Código del Modelo SVR</i>	<i>46</i>
<i>Figura 27: Modelo Árbol de Decisión</i>	<i>47</i>
<i>Figura 28: Código del Modelo Árbol de Decisión.....</i>	<i>47</i>
<i>Figura 29: Modelo Bosques Aleatorios</i>	<i>48</i>
<i>Figura 30: Código del Modelo Bosques Aleatorios.....</i>	<i>48</i>
<i>Figura 31: Modelo ANN</i>	<i>49</i>
<i>Figura 32: Código del Modelo ANN</i>	<i>49</i>
<i>Figura 33: Comparación Regresión Lineal Múltiple</i>	<i>51</i>
<i>Figura 34: Comparación SVR.....</i>	<i>52</i>
<i>Figura 35: Comparación Árboles de Decisión</i>	<i>53</i>
<i>Figura 36: Comparación Bosques Aleatorios</i>	<i>54</i>
<i>Figura 37: Comparación ANN</i>	<i>55</i>

Índice de tablas

<i>Tabla 1: Comparación entre DSRC y C-V2X</i>	<i>11</i>
<i>Tabla 2: Coeficiente de pérdidas por el entorno</i>	<i>19</i>
<i>Tabla 3: Comparación entre protocolos.....</i>	<i>21</i>
<i>Tabla 4: Modelo empírico pérdidas según el escenario.....</i>	<i>42</i>
<i>Tabla 5: Variables empleadas en los modelos</i>	<i>45</i>
<i>Tabla 6: Comparación de resultados de los modelos.....</i>	<i>56</i>
<i>Tabla 7: Presupuesto del proyecto</i>	<i>57</i>

Lista de acrónimos

3GPP	<i>3rd Generation Partnership Project</i>
ACK	<i>Acknowledgment</i>
ADAS	<i>Advanced Driver Assistance Systems</i>
ANN	<i>Artificial Neural Network</i>
BSW	<i>Blind Spot Warning</i>
CNN	<i>Convolutional Neuronal Networks</i>
C-V2X	<i>Cellular – Vehicle to Everything</i>
DL	<i>Deep Learning (en el contexto de inteligencia artificial)</i>
DL	<i>Downlink (en el contexto de telecomunicaciones)</i>
DSRC	<i>Dedicated Short Range Communication</i>
EEBL	<i>Emergency Electronic Brake Light</i>
IEEE	<i>Institute of Electrical and Electronics Engineers</i>
FCW	<i>Forward Collision Warning</i>
IA	<i>Inteligencia Artificial</i>
IMA	<i>Intersection Movement Assist</i>
IP	<i>Internet Protocol</i>
ITS	<i>Intelligent Transportation Systems</i>
KPI	<i>Key Performance Indicator</i>
LOS	<i>Line-of-Sight</i>
LSTM	<i>Long Short-Term Memory</i>
LTE	<i>Long Term Evolution</i>
MAE	<i>Mean Absolute Error</i>

ML	<i>Machine Learning</i>
MSE	<i>Mean Squared Error</i>
NLOS	<i>Non-Line-of-Sight</i>
OFDM	<i>Orthogonal Frequency-Division Multiplexing</i>
OFDMA	<i>Orthogonal Frequency-Division Multiple Access</i>
OSM	<i>OpenStreetMap</i>
PC5	<i>Interfaz Directa</i>
PDU	<i>Protocol Data Unit</i>
PKI	<i>Public Key Infrastructure</i>
Pr	<i>Potencia recibida</i>
Pt	<i>Potencia transmitida</i>
R²	<i>Coeficiente de Determinación</i>
RNN	<i>Recurrent Neuronal Networks</i>
SC-FDMA	<i>Single-Carrier Frequency Division Multiple Access</i>
SNR	<i>Signal-to-Noise Ratio</i>
SUMO	<i>Simulation of Urban Mobility</i>
SVM	<i>Support Vector Machine</i>
SVR	<i>Support Vector Regression</i>
SYN	<i>Synchronize</i>
TCP	<i>Transmission Control Protocol</i>
UDP	<i>User Datagram Protocol</i>
UL	<i>Uplink</i>
Uu	<i>Interfaz de Red</i>

V2I	<i>Vehicle-to-Infrastructure</i>
V2N	<i>Vehicle-to-Network</i>
V2P	<i>Vehicle-to-Pedestrian</i>
V2V	<i>Vehicle-to-Vehicle</i>
V2X	<i>Vehicle-to-Everything</i>
VANET	<i>Vehicular Ad-Hoc Network</i>
WAVE	<i>Wireless Access in Vehicular Environments</i>

Índice de contenidos

Resumen	i
Abstract	iii
Índice de figuras	v
Índice de tablas	vi
Lista de acrónimos.....	vii
1. Introducción	1
1.1 Marco y motivación del proyecto	1
1.2 Objetivos técnicos y académicos	1
1.3 Estructura de la memoria	2
2. Marco tecnológico.....	5
2.1 Sistemas de Transporte Inteligente (ITS) y Comunicaciones V2X.....	5
2.2 Comunicaciones V2V.....	7
2.2.1 Aplicaciones y Beneficios.....	7
2.2.2 Tecnologías de comunicaciones más utilizadas	8
2.3 Estudio del canal físico.....	12
2.3.1 Entorno de propagación	12
2.3.3 Parámetros de los modelos de propagación	13
2.3.4 Modelos de propagación	17
2.4 Tecnologías de transmisión	20
2.5 Herramientas empleadas.....	21
2.5.1 OpenStreetMap	21
2.5.2 SUMO.....	21
2.5.3 TraCI.....	22
2.5.4 Machine learning y deep learning.....	22
3. Especificaciones y restricciones de diseño	27
3.1 Especificaciones Iniciales	27
3.2 Metodología.....	27
3.3 Solución Propuesta.....	28
4. Descripción de la solución propuesta	29
4.1 Obtención de mapas y creación de escenarios	29
4.2 Simulaciones de los escenarios	39
4.3 Dataset.....	43
4.4 Modelos predictivos.....	44
5. Resultados	51
6. Impacto del proyecto.....	57
7. Conclusiones	61

7.1	Conclusiones	61
7.2	Trabajos Futuros.....	61
Referencias		63
Anexos		71
A.	Archivo de configuración SUMO	73
B.	Código para conseguir información de los vehículos (Conseguir_información_vehiculos.py)	76
C.	Código para calcular la potencia de los vehículos (calculo_potencia_vehiculos_cercanos.py)	79
D.	Código para generar el Dataset	82
E.	Código del modelo de Regresión Lineal Múltiple	85
F.	Código del modelo de SVR.....	87
G.	Código del modelo de Árbol de Decisión	89
H.	Código del modelo de Bosques Aleatorios	91
I.	Código del modelo de ANN.....	93
J.	Dataset	95

1. Introducción

1.1 Marco y motivación del proyecto

El avance de las tecnologías de comunicación inalámbrica ha impulsado el desarrollo de sistemas de transporte inteligentes (*ITS*, por sus siglas en inglés), donde la comunicación vehículo a vehículo (*V2V*, *Vehicle-To-Vehicle*), vehículo a infraestructura (*V2I*, *Vehicle-To-Infrastructure*) o la comunicación vehículo a todo (*V2X*, *Vehicle-To-Everything*) desempeña un papel crucial. Esta tecnología permite que los vehículos compartan información en tiempo real, como su posición, velocidad y dirección, con el fin de mejorar la seguridad vial, la eficiencia del tráfico y reducir el impacto ambiental. La implementación de *V2V* es fundamental para el despliegue de vehículos autónomos y sistemas de conducción asistida, ya que facilita la prevención de accidentes, la gestión del tráfico y la optimización de las rutas.

Sin embargo, la estabilidad y disponibilidad de las comunicaciones *V2V* representan un desafío significativo, especialmente en entornos urbanos complejos. Factores como la alta densidad vehicular, la presencia de obstáculos físicos (edificios, puentes, árboles, parques...), las interferencias electromagnéticas y la velocidad relativa de los vehículos pueden afectar negativamente la calidad de la conexión y, por consiguiente, a la sesión de comunicación. Estos obstáculos hacen necesario el desarrollo de soluciones que permitan predecir y gestionar de manera eficiente la duración de las sesiones de comunicación entre vehículos.

Ante este contexto, surge la necesidad de aplicar técnicas avanzadas de *Machine Learning* (*ML*) y *Deep Learning* (*DL*) para modelar y predecir fenómenos complejos derivados de la comunicación *V2V*. Estas tecnologías permiten analizar grandes volúmenes de datos, identificar patrones y generar modelos predictivos capaces de estimar con precisión el tiempo de duración de las sesiones de comunicación.

1.2 Objetivos técnicos y académicos

Desde el punto de vista técnico, este proyecto tiene como objetivo el diseño e integración de un sistema de recopilación de datos, utilizando sensores estratégicamente ubicados en puntos clave. Estos sensores permitirán capturar información relevante sobre el tráfico y las condiciones de comunicación entre los vehículos.

Para obtener datos representativos de entornos urbanos reales, se han simulado escenarios complejos utilizando el simulador *SUMO* (*Simulation of Urban Mobility*). Este software facilita la generación de datos precisos sobre la movilidad vehicular, lo que permite analizar el comportamiento de las comunicaciones en distintas condiciones de tráfico.

Con el fin de estimar con precisión la duración de las sesiones de comunicación *V2V*, se han implementado algoritmos de *machine learning* y *deep learning*. Estos modelos buscan identificar patrones en los datos recopilados y mejorar la fiabilidad de las predicciones en entornos dinámicos y cambiantes.

Otro aspecto fundamental del proyecto es el análisis del impacto de diversos factores físicos en la estabilidad de las comunicaciones V2V. Entre ellos se consideran la potencia de transmisión, la relación señal a ruido (*SNR, Signal-to-Noise Ratio*), el ruido y el fenómeno de desvanecimiento lento (*slow fading*), importantes en la calidad de la conexión.

Finalmente, para evaluar la precisión de las predicciones realizadas, se ha llevado a cabo una comparación del rendimiento de distintos modelos predictivos. Para ello, se han utilizado métricas de evaluación estandarizadas como el error absoluto medio (*MAE*), el error cuadrático medio (*MSE*) y el coeficiente de determinación (R^2), cuyos detalles se describen en apartados posteriores.

1.3 Estructura de la memoria

En el siguiente capítulo, dedicado al marco tecnológico, se presenta una revisión exhaustiva de las tecnologías de comunicación vehicular, con un enfoque en la comunicación V2V, los desafíos actuales en entornos urbanos y el uso de técnicas de *machine learning* y *deep learning* en la predicción de eventos relacionados con la conectividad vehicular.

Posteriormente, en el capítulo sobre especificaciones y restricciones de diseño, se ofrece un resumen de los objetivos principales del proyecto, detallando las especificaciones que se han tenido en cuenta durante su desarrollo. Además, se describe la metodología empleada.

En el capítulo dedicado a la descripción de la solución propuesta, se presentan los escenarios de simulación desarrollados con el simulador *SUMO*, así como la recopilación de datos mediante detectores y el uso de librerías como *TraCI (Traffic Control Interface)*. Esta librería permite obtener información en tiempo real de información sobre la simulación. También se explica el procesamiento de los datos y la implementación de los modelos de *ML* y *DL*, describiendo su diseño, los algoritmos utilizados, el proceso de entrenamiento y la selección de métricas de evaluación.

A continuación, en el capítulo sobre resultados y análisis, se presentan los resultados de rendimiento obtenidos por cada modelo predictivo, comparando su eficacia y analizando los datos mediante gráficos y métricas de precisión para identificar el modelo más adecuado.

Más adelante, se analiza el impacto del proyecto, abordando tanto los aspectos económicos como su repercusión en distintos ámbitos clave. En primer lugar, se examinan los costos asociados a su desarrollo y ejecución, detallando los recursos empleados. Además, se evalúa su impacto social, económico, en salud y seguridad, así como su influencia en el ámbito ambiental y tecnológico, destacando su relevancia dentro de los sistemas de transporte inteligente. Este análisis permite contextualizar la importancia del proyecto y sus posibles aplicaciones en escenarios reales.

En el capítulo de conclusiones y líneas de trabajo futuras, se sintetizan los hallazgos más relevantes del estudio, evaluando el grado de cumplimiento de los objetivos planteados y el impacto de los resultados obtenidos.

Por último, el documento concluye con la enumeración de las referencias bibliográficas utilizadas en su desarrollo y redacción. También se incluyen anexos con información complementaria relevante, como la configuración de los ficheros y el código fuente empleado en las simulaciones.

2. Marco tecnológico

2.1 Sistemas de Transporte Inteligente (ITS) y Comunicaciones V2X

Los ITS representan un conjunto de soluciones tecnológicas destinadas a mejorar la seguridad vial, la eficiencia del tráfico y la sostenibilidad ambiental mediante la integración de sistemas de comunicación y gestión del tráfico. Estos sistemas buscan optimizar la movilidad, mejorar la seguridad de los usuarios y reducir el impacto ambiental generado por el transporte [1].

Los ITS combinan tecnologías avanzadas de procesamiento de datos, telecomunicaciones, sensores y sistemas de control para ofrecer servicios innovadores como la gestión adaptativa del tráfico, sistemas de peaje electrónicos, información en tiempo real a los conductores, y vehículos autónomos y conectados. La Figura 1 muestra la conectividad que ofrecen los ITS [2].



Figura 1: Sistemas de transporte inteligente

Dentro de los ITS, las comunicaciones V2X son fundamentales, ya que permiten la interconexión entre vehículos, infraestructuras y otros elementos del entorno. Las comunicaciones V2X son esenciales para el desarrollo de la movilidad autónoma y conectada, ya que permiten la cooperación entre vehículos e infraestructuras para optimizar el flujo vehicular y evitar las colisiones, garantizando la seguridad [3]. Además, contribuyen a la reducción de emisiones contaminantes mediante una conducción más eficiente y la gestión inteligente del tráfico.

A pesar de los beneficios, las comunicaciones V2X enfrentan desafíos importantes debido a las particularidades de los entornos vehiculares. Factores como la alta movilidad, la densidad del tráfico y las interferencias electromagnéticas afectan la estabilidad y calidad de las conexiones. Por ello, el diseño de redes de comunicación vehicular debe basarse en estudios detallados del comportamiento del canal de propagación y en simulaciones que consideren múltiples escenarios y situaciones potenciales.

Estas comunicaciones se dividen principalmente en [4]:

- **V2I (Vehicle-to-Infrastructure o Vehículo a Infraestructura):** Las comunicaciones entre vehículos e infraestructuras, como semáforos, señales de tráfico y centros de control de tráfico. Facilitan la gestión eficiente del tráfico y la transmisión de información sobre las condiciones de la vía.
- **V2V (Vehicle-to-Vehicle o Vehículo a Vehículo):** Las comunicaciones directas entre vehículos, permitiendo el intercambio de información relevante en tiempo real. Es esencial para la prevención de accidentes y la mejora de la seguridad vial.
- **V2P (Vehicle-to-Pedestrian o Vehículo a Peatones):** Las comunicaciones entre vehículos y peatones para advertir sobre situaciones peligrosas.
- **V2N (Vehicle-to-Network):** Las comunicaciones entre vehículos y la red para la transmisión de datos a través de internet y servicios en la nube.

En la Figura 2 se puede ver el tipo de comunicaciones descritas anteriormente [5].



Figura 2: Comunicación V2X

Este proyecto se enfoca específicamente en las comunicaciones V2V, que permiten el intercambio directo de información entre vehículos sin necesidad de pasar por una infraestructura intermedia, como estaciones base o servidores en la nube (V2I). Este tipo de comunicación ofrece una ventaja significativa en términos de latencia reducida, lo que es

crucial para aplicaciones de seguridad y respuesta en tiempo real, como sistemas de alerta de colisión o frenado de emergencia cooperativo. Sin embargo, las comunicaciones V2V presentan grandes desafíos, como la alta variabilidad del canal de comunicación, la dependencia de la línea de visión directa (*LoS, Line of Sight*), y la dificultad de mantener conexiones estables en entornos urbanos densos, donde los edificios y otros obstáculos interfieren en la propagación de la señal.

2.2 Comunicaciones V2V

2.2.1 Aplicaciones y Beneficios

Las comunicaciones V2V permiten a los vehículos compartir información en tiempo real, lo que resulta clave para prevenir accidentes y gestionar de forma más eficiente el tráfico.

Algunas de las aplicaciones más relevantes de las comunicaciones V2V son [6]:

- **Prevención de accidentes:** Al compartir información sobre la velocidad, dirección y posición, los vehículos pueden anticiparse a posibles colisiones, al reaccionar de forma más rápida que sin esta comunicación directa.
- **Gestión eficiente del tráfico:** La comunicación entre vehículos permite optimizar rutas y evitar zonas congestionadas, con tiempos de reacción muy rápidos.
- **Conducción autónoma:** Es fundamental para la toma de decisiones seguras y eficientes en entornos urbanos.
- **Advertencias de emergencias:** Los vehículos pueden alertar sobre accidentes o condiciones peligrosas en la vía, como la presencia de hielo en la carretera, vehículos detenidos en el carril, desprendimientos de tierra, cruces con poca visibilidad o zonas con baja adherencia debido a lluvia intensa o nieve.
- **Coordinación en intersecciones:** Mejora la eficiencia en cruces sin semáforos, reduciendo los tiempos de espera.
- **Asistencia en cambios de carril:** Alerta sobre vehículos en puntos ciegos o aproximándose rápidamente.
- **Formación de pelotones:** Los vehículos pueden agruparse en pelotones (*platooning*, en inglés), una estrategia en la que varios vehículos viajan en fila manteniendo una distancia constante entre ellos, coordinados mediante comunicaciones V2V [7]. Esta técnica mejora la eficiencia del tráfico, reduce el consumo de combustible y disminuye las emisiones contaminantes debido a la reducción de la resistencia aerodinámica. Además, al sincronizar la aceleración y el frenado entre los vehículos, se mejora la seguridad vial, minimizando el riesgo de colisiones y optimizando el flujo vehicular, sobre todo en autopistas.

2. *Marco tecnológico*

La implementación de tecnologías V2V contribuye significativamente a la reducción de accidentes, el ahorro de tiempo y el consumo eficiente de energía, mejorando la experiencia de conducción y la seguridad vial.

La Figura 3 muestra la comunicación V2V en un entorno de carretera [8].



Figura 3: Comunicación V2V

2.2.2 Tecnologías de comunicaciones más utilizadas

Cuando se trata de las tecnologías vehiculares se puede apreciar un nexo común. Las tecnologías más empleadas son las de medio alcance, en concreto: *DSRC (Dedicated Short Range Communication)*, ondas milimétricas y *C-V2X (Cellular-Vehicle-To-Everything)* [9].

En este caso me centraré en explicar estas tecnologías:

- **DSRC**

DSRC es una tecnología de comunicación inalámbrica diseñada específicamente para facilitar la comunicación V2V y entre vehículos e infraestructura (V2I). Su propósito principal es mejorar la seguridad vial y la eficiencia del tráfico mediante el intercambio de información en tiempo real. Opera en un espectro de frecuencia dedicado de 5.9 GHz, con un ancho de banda de 10 MHz. También es conocida como *WAVE (Wireless Access in Vehicular Environments)* en Estados Unidos y como *ETSI ITS-G5* en Europa.

Esta tecnología se basa en el estándar IEEE 802.11p, una versión adaptada del Wi-Fi para entornos vehiculares, y se complementa con otros estándares como IEEE 1609.x, que gestionan aspectos de seguridad, control de canales y aplicaciones [10]. Esta combinación permite alcanzar tiempos de respuesta extremadamente bajos, inferiores a 50 milisegundos,

algo crucial para aplicaciones de seguridad donde cada fracción de segundo puede prevenir accidentes. Además, soporta tasas de transferencia de datos que oscilan entre 3 y 27 Mbps, lo cual es suficiente para transmitir información clave, como la ubicación GPS, velocidad, aceleración y dirección de los vehículos [11].

Una de las características más destacadas de es su capacidad para ofrecer cobertura de 360 grados con un alcance de hasta 1 kilómetro. Esto permite a los vehículos detectar y comunicarse con otros en su entorno, brindando a los conductores información valiosa para prevenir accidentes. Además, emplea *OFDM (Orthogonal Frequency Division Multiplexing)* [12]. Entre sus casos de uso más importantes se encuentran [13]:

- *Emergency Electronic Brake Light (EEBL)*, que alerta sobre frenadas bruscas de vehículos delante
- *Blind Spot Warning (BSW)*, que advierte sobre vehículos en el ángulo muerto
- *Forward Collision Warning (FCW)*, que avisa de posibles colisiones frontales
- *Intersection Movement Assist (IMA)* ayudan a prevenir accidentes en intersecciones peligrosas.

La seguridad en las comunicaciones de DSRC es fundamental para su funcionamiento. Por ello, implementa un robusto sistema de Infraestructura de Clave Pública (*PKI*) que garantiza la autenticidad e integridad de los mensajes intercambiados [14]. Este sistema de cifrado previene ataques como la suplantación de identidad, la manipulación de datos y la retransmisión de mensajes falsos. Así, DSRC ofrece un entorno de comunicación seguro y confiable, clave para la adopción de tecnologías de transporte inteligente. En la Figura 4 aparecen las frecuencias que emplean los principales países [15].

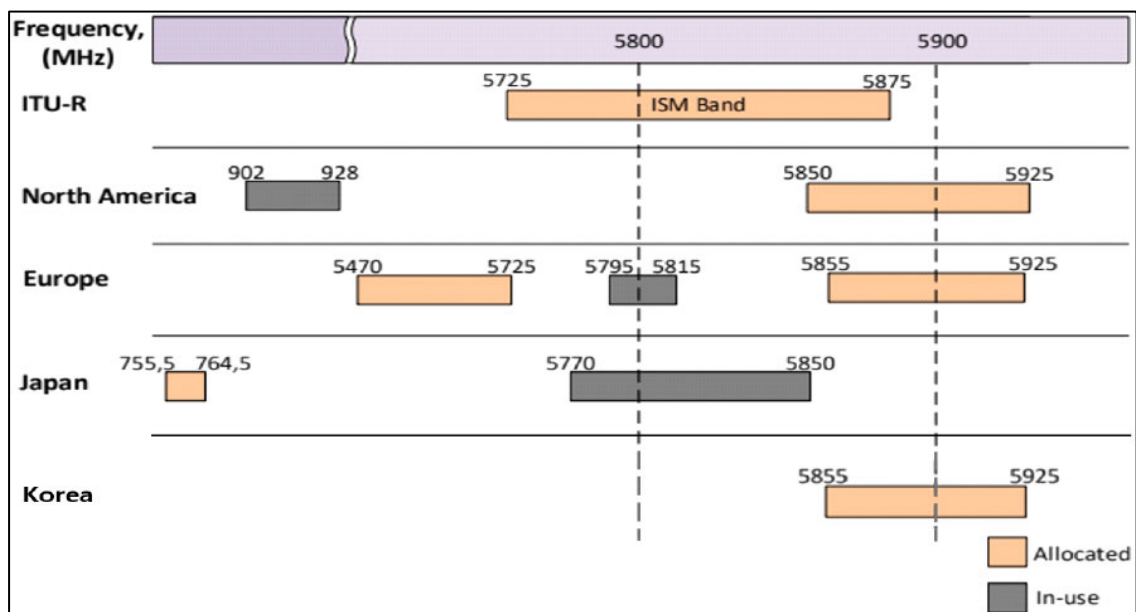


Figura 4: Rango de frecuencias

- **C-V2X**

C-V2X es una tecnología de comunicación diseñada para permitir la interacción entre vehículos (V2V), vehículos e infraestructura (V2I), vehículos a peatones (V2P) y vehículos a redes (V2N) utilizando redes celulares. A diferencia de *DSRC*, C-V2X aprovecha la infraestructura de redes móviles existentes, como *LTE* y futuras redes 5G, para proporcionar una comunicación más amplia y eficiente entre los participantes del tráfico [16].

Una de las principales ventajas de C-V2X es su capacidad para integrarse con redes móviles, lo que permite una mayor cobertura al estar dentro de *3GPP* (*Third Generation Partnership Project*) [17].

Esta tecnología utiliza dos modos de comunicación: el modo directo (*PC5*) para la comunicación inmediata entre dispositivos, y el modo de red (*Uu*) para la comunicación a través de la infraestructura de la red celular [18].

El modo directo permite una comunicación de baja latencia (<10 ms) permitiendo una comunicación con el resto de usuarios de la vía sin depender de la infraestructura de red celular. Emplea un espectro de 5.9 GHz empleando una modulación *SC-FDMA* (*Single Carrier Frequency Division Multiple Access*) para el enlace ascendente o *Uplink* (UL) y para el enlace descendente o *Downlink* (DL).

La interfaz de red aprovecha las redes 4G-*LTE* y 5G empleando una modulación *OFDMA* (*Orthogonal Frequency Division Multiple Access*) para *downlink* y una modulación *SC-FDMA* para *uplink*.

C-V2X también se destaca por su capacidad de soportar una amplia gama de aplicaciones de seguridad y eficiencia del tráfico. Entre ellas se incluyen advertencias de colisión, gestión de semáforos y señales, información sobre condiciones de la carretera y notificaciones de emergencias. Gracias a su integración con redes celulares, C-V2X puede proporcionar servicios adicionales como navegación en tiempo real, actualizaciones sobre tráfico y clima, y servicios de entretenimiento, mejorando la experiencia del conductor.

La adopción de C-V2X está impulsada por su compatibilidad con las futuras redes 5G, que ofrecerán mayor capacidad, menor latencia y una mayor densidad de dispositivos conectados. Esto permite aplicaciones más avanzadas, como la conducción autónoma y la gestión de vehículos en tiempo real.

La Tabla 1 contiene las principales características de las tecnologías *DSRC/WAVE* y C-V2X [19].

Tabla 1: Comparación entre DSRC y C-V2X

Características	DSRC/WAVE	C-V2X
Estándar	IEEE 802.11p	3GPP LTE & 5G
Frecuencia	5.9 GHz	5.9 GHz (PC5), Celular (Uu)
Ancho de banda	10 MHz	10 MHz o 20MHz
Velocidad	27 Mbps	Uplink/Downlink: 50/150 Mbps
Latencia	<50 ms	<100ms
Alcance	300 – 1000 metros	hasta 2 Km
Tecnología	OFDM	OFDMA (DL), SC-FDMA (UL)

La diferencia entre la tecnología DSRC y C-V2X suele variar en función de la región en la que se vaya a utilizar (Europa, Estados Unidos o Japón).

- **mmWaves**

Las ondas milimétricas están emergiendo como una tecnología prometedora para las comunicaciones V2V. Las ondas milimétricas son ondas electromagnéticas con longitudes de onda entre 10 mm y 1 mm, correspondientes a frecuencias entre 300 MHz y 30 GHz [20]. En el contexto de las comunicaciones V2V, se están explorando principalmente las bandas de 60 GHz y 73 GHz [21].

Principales ventajas de las ondas milimétricas [22]:

- Alta velocidad de transmisión de datos: Pueden alcanzar velocidades de varios gigabits por segundo, crucial para el intercambio rápido de información entre vehículos.
- Baja latencia: Importante para aplicaciones de seguridad en tiempo real como alertas de colisión.
- Alta resolución: Permite una detección más precisa de objetos y vehículos cercanos.

Sin embargo, las ondas milimétricas también presentan desafíos importantes:

- Alcance limitado: Tienen dificultades para propagarse a largas distancias y penetrar obstáculos sólidos.
- Sensibilidad a interferencias: Pueden verse afectadas por condiciones atmosféricas extremas.

- Infraestructura más robusta: Requieren más estaciones base para una cobertura efectiva.

Para superar estas limitaciones, se están investigando técnicas avanzadas como la formación de haces (*beamforming*) y el uso de antenas direccionales.

Las ondas milimétricas son particularmente prometedoras para aplicaciones V2V en escenarios de conducción autónoma. Aunque su implementación requerirá de desarrollar soluciones innovadoras para poder utilizar su potencial en entornos vehiculares dinámicos.

2.3 Estudio del canal físico

En este apartado se definen los principales parámetros de propagación y se detallan las características de los modelos empleados en las comunicaciones entre vehículos, siendo el último de ellos el que he emplee al desarrollar este proyecto. Los fenómenos que se describen en este apartado están contemplados en la recomendación ITU-R P.1411-12 [23].

2.3.1 Entorno de propagación

Cuando hablamos del canal de propagación hay que tener en cuenta el tipo de entorno donde nos encontramos, los principales entornos son: urbano, interurbano, rural y autovía. Sobre ellos se formulan los distintos modelos que se verán en el siguiente epígrafe [24].

Cuando se hace referencia al entorno de propagación se tiene en cuenta los conceptos *LoS* (*Line of Sight*) y *NLoS* (*Non-Line of Sight*) que son cruciales en la propagación de las señales. Ambos fenómenos se pueden apreciar en la Figura 5 [25].

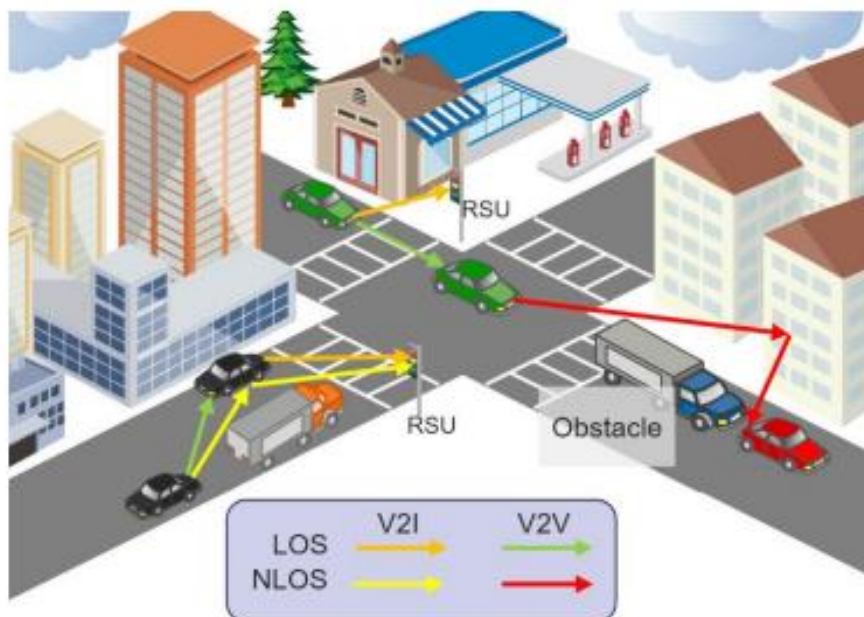


Figura 5: Fenómenos LoS y NLoS

El termino *LOS* hace referencia a la condición de que la señal viaja directamente desde el transmisor al receptor sin la presencia de obstáculos en el camino, como se dice en términos coloquiales, cuando el transmisor y el receptor tienen visión directa uno de otro. Esto favorece una señal con mayor calidad y menor atenuación.

Por el otro lado, tenemos el término *NLOS*, el cual hace referencia a la situación donde la línea de visión entre el trasmisor y el receptor está bloqueada por obstáculos físicos como edificios, árboles... Esto influye en la calidad de la señal al sufrir atenuaciones en su camino del transmisor al receptor. Esto provoca una serie de fenómenos asociados como: reflexión, difracción y dispersión, los cuales se profundizarán en el siguiente epígrafe [26].

1.3.3 Parámetros de los modelos de propagación

A lo largo de este epígrafe se desarrollarán cuáles son los principales parámetros que se deben tener en cuenta a la hora de formular cada modelo.

- **Dispersión**

La dispersión (*scattering*) es el fenómeno por el cual una onda electromagnética es desviada en múltiples direcciones al encontrarse con pequeñas partículas o irregularidades en el medio de propagación. Este efecto provoca que la señal se debilite y se distribuya en diferentes direcciones [27].

Este fenómeno puede causar pérdida de señal y reducción de la calidad de la comunicación debido a que genera múltiples trayectorias para la señal, lo que puede provocar interferencia y degradación en la calidad de la señal.

Este fenómeno se puede apreciar con claridad en la Figura 6 donde la señal transmitida por la antena del dispositivo móvil se refleja sobre los árboles (*scattering* en la Figura 6) permitiendo de esta manera que esta señal se propague en varias direcciones para llegar a la antena receptora.

- **Reflexión**

La reflexión es el cambio en la dirección de una onda electromagnética cuando incide sobre una superficie de separación entre dos medios con diferentes propiedades dieléctricas y de conductividad. La ley de reflexión establece que el ángulo de incidencia es igual al ángulo de reflexión, medidos con respecto a la normal a la superficie. Este fenómeno se produce tanto en entornos donde no hay línea de visión directa (*NLoS*) y en entornos con línea de visión (*LoS*).

La reflexión produce un modelo llamado Modelo de dos rayos, el cual se suele emplear en comunicaciones vehiculares con línea de visión (*LoS*) por su reflexión producida en el suelo. Este modelo se profundizará más adelante.

En la Figura 6 se puede apreciar cómo se produce este el efecto (*Reflection* en la Figura 6) cuando la señal transmitida desde el dispositivo móvil se refleja en el edificio y en vehículo, provocando el fenómeno de reflexión, permitiendo que la señal se propague hasta la antena receptora.

- **Difracción**

La difracción es el fenómeno que permite que las ondas electromagnéticas se curven alrededor de los bordes de un obstáculo o se propaguen a través de aperturas, incluso cuando el receptor no está en línea directa de visión con el transmisor (*NLoS*). Por el principio de *Huygens-Fresnel*: “Cada punto de un frente de onda puede ser considerado como fuente secundaria de onda que se expanden en todas direcciones con una velocidad igual a la velocidad de propagación de la onda primaria” [28].

Este fenómeno se recrudce en entornos urbanos don de los edificios, señales, árboles, vehículos... obstaculizan la que haya una visión directa entre el transmisor y el receptor. Este fenómeno es importante en entornos urbanos o en entornos con obstáculos porque permite que la señal alcance a receptores que están bloqueados por edificios u obstáculos.

En la Figura 6 se puede comprobar como el fenómeno de la difracción (*Diffraction* o *Shadowing* en la Figura 6) permitiendo que la señal transmitida desde el dispositivo móvil se propague, debido a este fenómeno, para llegar a la antena receptora.

Por resumir, los fenómenos descritos anteriormente se pueden apreciar en la Figura 6. Estos fenómenos se producen en los escenarios donde no hay visión directa, *NLoS*, se producen los

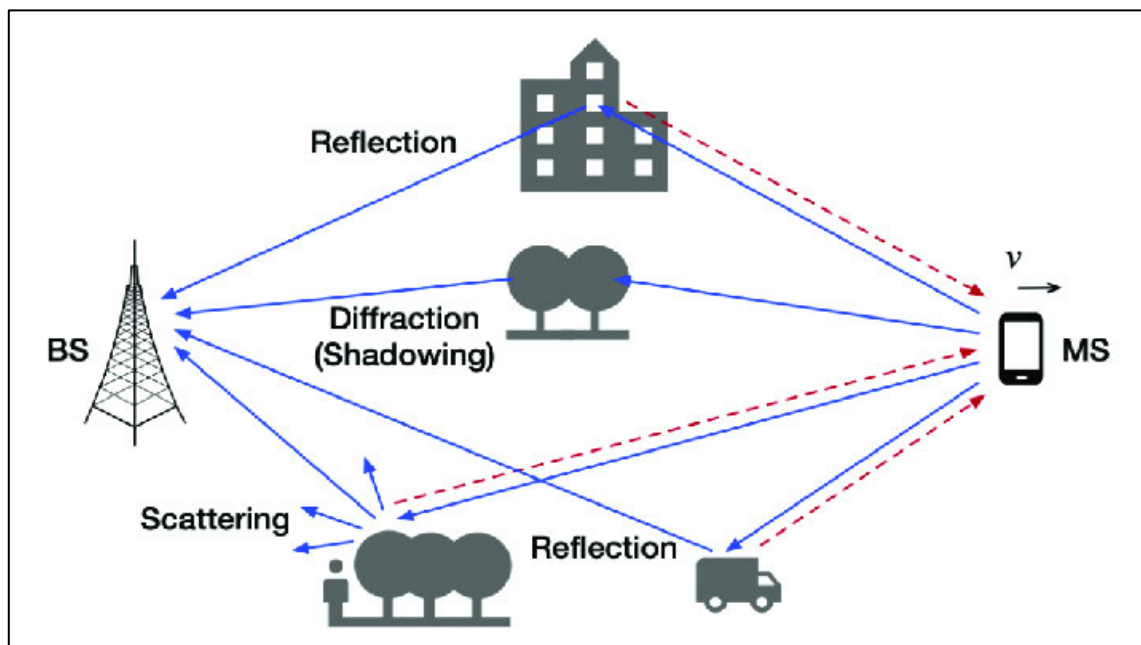


Figura 6: Escenario de propagación NLoS

tres fenómenos ya descrito: en uno de ellos se puede apreciar el fenómeno de reflexión (*Reflection*), el fenómeno de difracción (*Diffraction* o *Shadowing*) y el fenómeno de dispersión (*Scattering*).

- **Pérdidas por trayectoria**

Las pérdidas por trayectoria o *Path Loss* se define como la reducción en la densidad de potencia de una onda electromagnética a medida que se propaga por el espacio. En las comunicaciones inalámbricas se define como la diferencia entre potencia transmitida y la potencia recibida expresada en decibelios (dB). El entorno predominante para los Sistemas de Transporte Inteligente (ITS) es el denominado "cañón urbano" (*street canyon*), se caracteriza por la presencia de numerosos elementos propios de entorno urbanos, tales como una alta concentración de edificios de distintos tamaños y formas, peatones, árboles, vehículos, entre otros. Esta configuración provoca que la propagación de la señal ocurra principalmente a lo largo de las calles. El modelo básico utilizado para describir este fenómeno se encuentra especificado en la recomendación vigente cuando se desarrolló este proyecto, ITU-R P.1411–12, titulada "Datos de propagación y métodos de predicción para la planificación de los sistemas de radiocomunicaciones de exteriores de corto alcance y redes de radiocomunicaciones de área local en la gama de frecuencias de 300 MHz a 100 GHz. Debido a la complejidad de las comunicaciones en entornos urbanos es necesario extender este modelo fundamental mediante el desarrollo de modelo más avanzados.

El *Path Loss* se origina por varios factores como:

- Distancia: Cuanto mayor sea la distancia, mayor será la atenuación de la señal.
- Frecuencia: Cuanto mayor sea la frecuencia mayor serán las pérdidas.
- Entorno: Los obstáculos del entorno como árboles, edificios... y las condiciones atmosféricas pueden causar atenuaciones.

- **Desvanecimiento**

El desvanecimiento o *fading* es un fenómeno que afecta las señales inalámbricas debido a cambios rápidos en las condiciones del canal de comunicación. En redes vehiculares (*VANETs*), el desvanecimiento es causado principalmente por la alta movilidad de los vehículos, que provoca variaciones en la intensidad de la señal recibida. Esto se traduce en pérdidas de paquetes y afecta el rendimiento general de la red.

Este fenómeno que afecta a la variación de la intensidad o amplitud de una señal de radiofrecuencia al propagarse en un canal inalámbrico, provocada por fenómenos como la reflexión, difracción y dispersión al interactuar con obstáculos (edificios, vehículos, árboles, etc.). Estos cambios pueden ser rápidos o lentos y se manifiestan en variaciones de la potencia de la señal recibida. En las redes vehiculares (*VANETs*), la alta movilidad de los vehículos

incrementa estas variaciones, ocasionando pérdidas de paquetes y afectando el rendimiento de la red [29].

Hay varios tipos de desvanecimientos:

- Desvanecimientos a Gran Escala (*Large-Scale Fading*): Este tipo ocurre por la pérdida de trayectoria y el sombreado debido a grandes obstáculos como edificios.
- Desvanecimientos a Pequeña Escala (*Small-Scale Fading*): Este tipo de desvanecimientos ocurren debido a interferencias de múltiples trayectorias de la señal en distancias cortas.
- Desvanecimiento Selectivo en Frecuencia (*Frequency-Selective Fading*): Este tipo afecta a diferentes frecuencias de la señal de manera distinta, debido a multitrayectoria y a diferencias en el tiempo de llegada.
- Desvanecimiento Selectivo en el Tiempo (*Time-Selective Fading*): Este desvanecimiento ocurre cuando el canal cambia rápidamente respecto al tiempo, ocurre normalmente por movimientos del transmisor y/o receptor. A este tipo de desvanecimiento se le conoce como efecto Doppler o Dispersión Doppler. En la Expresión 2.1 se puede ver su ecuación.

$$f_d = \frac{v}{c} f_c \quad (2.1)$$

Donde f_d corresponde con el desplazamiento Doppler; v es la velocidad relativa entre el transmisor y el receptor; c es la velocidad de la luz; f_c corresponde con la frecuencia de portadora.

- **Shadowing**

El ensombrecimiento o *Shadowing*, también conocido como desvanecimiento lento o *slow/large-scale fading* es un fenómeno que ocurre en las comunicaciones inalámbricas cuando obstáculos físicos obstruyen la trayectoria directa entre el transmisor y el receptor, se produce NLoS [30].

Este fenómeno se produce cuando elementos del entorno como edificios, árboles y otros elementos bloquean la línea de visión provocando atenuaciones en la señal. Esto provoca variaciones en la potencia de la señal recibida.

Se suele caracterizar típicamente mediante la distribución log-normal, que describe como varía la intensidad de la señal con la distancia desde el transmisor (en dB). Este fenómeno contribuye al desvanecimiento a gran escala, que representa la variación de la potencia de la señal debido a los efectos producidos en el entorno de propagación.

En la Figura 7 muestra el efecto que se produce en la potencia de la señal debido a las pérdidas por propagación. Los efectos descritos anteriormente como el multitrayecto (*multipath*), las

pérdidas por trayectoria (*Path Loss*) y el ensombrecimiento (*shadowing*) son visibles en esta figura [31].

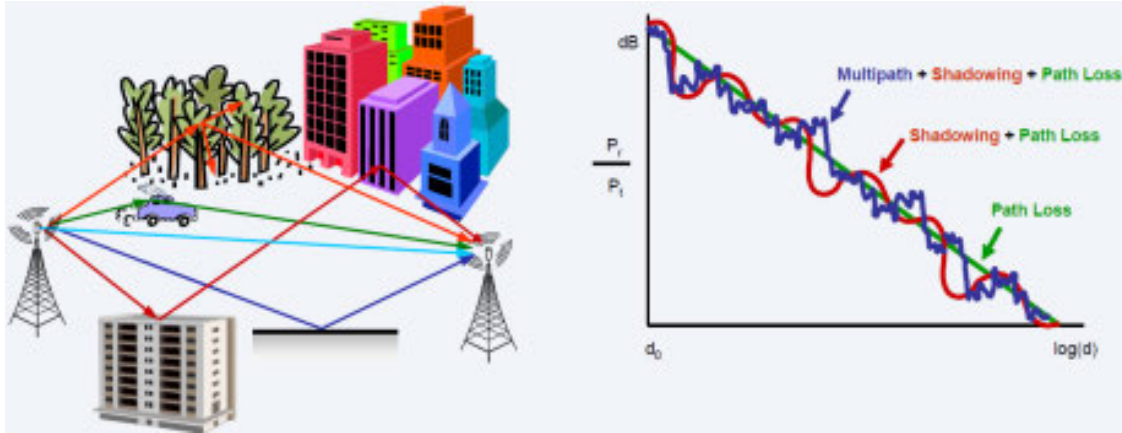


Figura 7: Efectos del canal sobre la señal de propagación

2.3.4 Modelos de propagación

Los modelos de propagación son fundamentales para comprender el comportamiento de las señales en entornos vehiculares. Estos modelos permiten predecir el comportamiento de la señal entre el transmisor y el receptor teniendo en cuenta las condiciones del ambiente y los obstáculos. A continuación, se muestran los principales modelos de propagación existentes:

- **Modelo de espacio libre**

El modelo de pérdidas en espacio libre es el modelo básico empleado para calcular la pérdida de potencia de la señal en entornos sin obstáculos. Este modelo calcula la pérdida de potencia provocada por la distancia entre el transmisor y el receptor.

En la Expresión 2.12 se describe la ecuación de *Friis* para calcular las pérdidas de potencia en espacio libre:

$$P_r(d) = P_t G_t G_r \frac{\lambda^2}{(4\pi)^2 d^2} \quad (2.2)$$

Donde P_r es la potencia transmitida; G_t corresponde a la ganancia de la antena transmisora; G_r corresponde a la ganancia de la antena receptora; λ es la longitud de onda en metros correspondiente a la frecuencia de trabajo y d es la distancia entre el transmisor y el receptor [32].

La Expresión 2.13 describe la fórmula de las pérdidas en espacio libre más conocida:

$$PL(dB) = 20\log_{10}(d) + 20\log_{10}(f) + 32.44 \quad (2.3)$$

Donde PL es la pérdida en decibelios, d es la distancia entre el transmisor y el receptor en kilómetros, f corresponde a la frecuencia de la señal transmitida en MHz y 32.44 es la constante que regula las unidades en Kilómetros y MHz [33].

- **Modelos deterministas**

Este tipo de modelos emplean las condiciones del entorno para calcular de forma más precisa las pérdidas producidas en la propagación de la señal.

Estos modelos destacan por modelar los fenómenos que se producen en la propagación como la reflexión, la difracción y la dispersión, siendo muy útiles para entornos urbanos y para la comunicación V2V.

Los modelos más conocidos son: modelo de trazado de rayos (*Ray Tracing*) y el modelo de dos rayos (*Two-Ray Ground Reflection Model*).

El modelo de trazado de rayos considera las condiciones del entorno para calcular la potencia recibida. Este modelo tiene en cuenta los fenómenos que se producen en la propagación como la reflexión, dispersión y difracción, empleando una mayor carga computacional y mayor complejidad en los cálculos. La Expresión 2.14 describe el cálculo que emplea este modelo:

$$P_r = P_t + G_t + G_r - \sum L_i \quad (2.4)$$

Donde la P_r y P_t corresponde con la potencia recibida y la potencia transmitida respectivamente en dB; G_t y G_r hacen referencia a la ganancia de la antena transmisora y receptora; $\sum L_i$ hace referencia a la pérdida total acumulada por la reflexión, difracción y dispersión.

El modelo de dos rayos emplea tanto la señal directa como la que se refleja en el suelo para calcular la potencia recibida. Este modelo carece de complejidad al solo depender de las alturas de las antenas, la distancia entre los vehículos y únicamente la presencia de dos ondas. La Expresión 2.15 describe el cálculo empleado:

$$P_r = P_t G_t G_r \frac{h_t^2 h_r^2}{d^4} \quad (2.5)$$

Donde h_t y h_r corresponden con la altura de la antena transmisora y receptora respectivamente [34].

- **Modelos empíricos**

Los modelos empíricos se desarrollan a partir de mediciones y análisis estadísticos realizados en entornos reales. Por este motivo, son más sencillos de implementar y sus cálculos son más sencillos. Son empleados principalmente en entornos urbanos, suburbanos y rurales.

Por todo lo anterior, son los modelos empleados en las comunicaciones V2V. Los principales modelos empíricos son los siguientes:

➤ **Log-distancia**

El modelo Log_distancia o *Log-distance* es un modelo de propagación ampliamente utilizado para predecir las pérdidas de la señal. Emplea la fórmula de pérdidas en espacio libre añadiendo un coeficiente de pérdidas (n), el cual varía según las condiciones del entorno. La Expresión 2.18 muestra la fórmula que emplea el (2.6) log-distancia [35]:

$$PL_{LD}(d) = PL_0(d_0) + 10n \log_{10} \left(\frac{d}{d_0} \right) \text{ dB}, \quad d > d_0$$

Donde PL_0 hace referencia a las pérdidas por propagación a una distancia de referencia d_0 en dB; d corresponde con la distancia entre las antenas transmisora y receptora.

En la Tabla 2 se hace referencia a los valores del coeficiente de pérdidas para cada entorno [36]:

Tabla 2: Coeficiente de pérdidas por el entorno

Entorno	Coeficiente de pérdidas (n)
Espacio Libre	2
Área urbana	2.7 – 3.5
Área suburbana	3 - 5
En interiores LoS	1.6 – 1.8
En interiores NLoS	4 - 6

➤ **Log-normal**

El modelo Log-normal o también conocido como modelo de *shadowing* (*Log-normal shadowing Path-Loss model*) es un modelo de propagación que parte del modelo anterior, Log-distancia, para predecir las pérdidas de señal tanto en entornos interiores como en exteriores. A diferencia del modelo anterior, este modelo implementa una variable aleatoria gaussiana, X_σ , que modela el efecto de *shadowing* expresando las variaciones aleatorias en las pérdidas por propagación debido a los obstáculos del entorno.

La Expresión 2.19 muestra la fórmula que emplea este modelo para calcular las pérdidas por propagación [37]:

$$PL(d) = PL_0(d_0) + 10n \log_{10} \left(\frac{d}{d_0} \right) + X_\sigma \text{ (dB)} \quad (2.7)$$

Donde, de manera equivalente a la Expresión 2.18, d hace referencia a la distancia entre la antena transmisora y receptora; PL_0 hace referencia a las pérdidas por propagación respecto a una distancia de referencia d_0 ; X_σ es la variable aleatoria gaussiana de media cero y desviación estándar σ .

Este modelo es una extensión del modelo Log-distancia con mayor precisión al incorporar el efecto de *shadowing*, permitiendo recrear escenarios más realistas, siendo muy útil para entornos urbanos.

2.4 Tecnologías de transmisión

En este apartado me centraré en explicar la capa de transporte y el tipo de *PDU* (*Protocol Data Unit* por sus siglas en inglés) que se puede implementar. Los protocolos de transporte más utilizados son los protocolos de transporte estándar *UDP* (*User Datagram Protocol*) y *TCP* (*Transmission Control Protocol*) [38].

TCP es un protocolo de comunicación orientado a la conexión que garantiza la entrega de datos entre dos dispositivos en el orden que fueron enviados [39]. Establece una conexión mediante un proceso denominado: “*three way handshake*” para transmitir la información. Estas vías son: *SYN* (Sincronización), *SYN-ACK* (Sincronización y Reconocimiento) y *ACK* (Reconocimiento) [40]. Por el contrario, UDP es un protocolo sin conexión que prioriza la velocidad sobre la fiabilidad enviando los datos directamente sin establecer una conexión previa [41].

La Figura 8 permite distinguir la diferencia en el envío de datos de ambos protocolos [42].

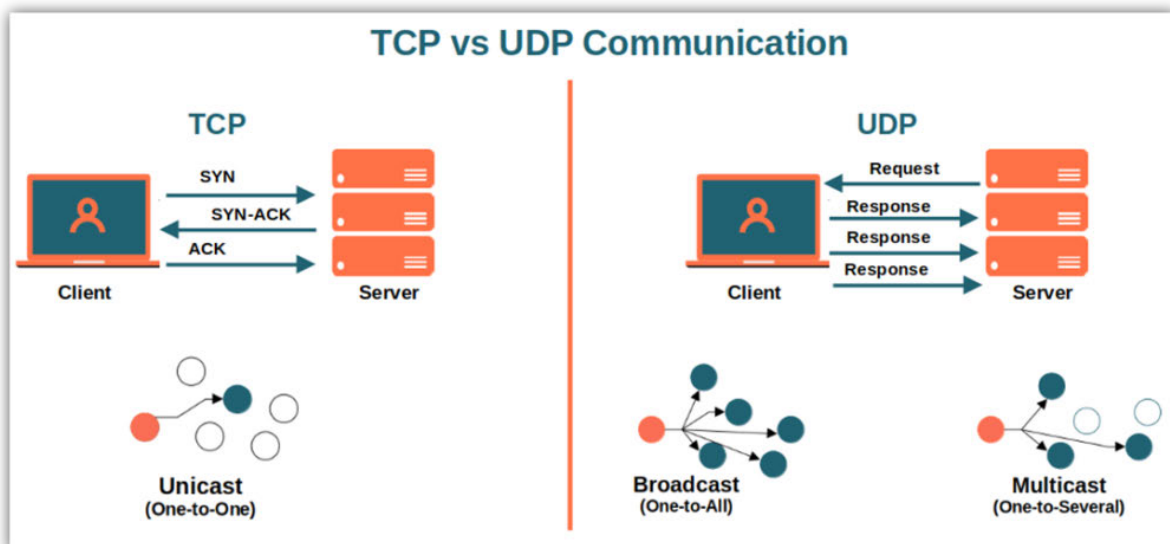


Figura 8: Diferencias entre TCP y UDP

En la Tabla 3 se pueden visualizar las principales diferencias entre ambos protocolos [43]:

Tabla 3: Comparación entre protocolos

Características	TCP	UDP
Orientación	Conexión establecida	Sin conexión
Fiabilidad	Garantiza entrega y orden	No garantiza entrega ni orden
Velocidad	Más lento, Control de flujo/congestión	Más rápido, Sin verificación
Sobrecarga	Alta (ACKs)	Baja, Sin confirmaciones
Latencia	Alta	Baja
Reenvío de datos	Sí, en caso de pérdida	No, en ningún caso
Tamaño de cabecera	20 bytes	8 bytes
Control de flujo/congestión	Ambos	No realiza ningún control

Las comunicaciones V2V requieren baja latencia por lo que *UDP* es el protocolo más adecuado al ofrecer alta velocidad y tiempos de respuesta rápidos.

2.5 Herramientas empleadas

2.5.1 OpenStreetMap

OpenStreetMap (OSM) es una plataforma colaborativa de código abierto que permite la creación y distribución de mapas geográficos de todo el mundo. Esta herramienta se basa en la contribución de usuarios que añaden información geoespacial como calles, edificios, zonas verdes, y elementos de infraestructura. Gracias a su flexibilidad y nivel de detalle, *OSM* es ampliamente utilizada para proyectos de simulación, análisis geoespacial y desarrollo de aplicaciones de movilidad.

En este proyecto, *OpenStreetMap* se ha empleado para la obtención de mapas realistas de la ciudad de Madrid. Estos mapas fueron posteriormente convertidos a un formato compatible con SUMO para generar entornos urbanos detallados y precisos [44].

2.5.2 SUMO

SUMO se trata de un simulador de tráfico de código abierto diseñado para modelar el flujo de vehículos en entornos urbanos, suburbanos o autopistas. Se trata de un simulador de DLR, el centro aeroespacial alemán responsable del desarrollo de este simulador, que permite la creación de escenarios realistas mediante la integración de mapas reales y la simulación de

vehículos con distintos comportamientos y características. Es esencial para generar los datos de movilidad empleados en este proyecto [45].

Permite analizar factores como la velocidad, la prioridad en intersecciones y la presencia de semáforos que afectan el tráfico, permitiendo modificar la presencia de estos. *SUMO* también posibilita la simulación de diferentes tipos de transporte, incluyendo coches, autobuses, camiones, peatones, etc., por lo que resulta una herramienta muy útil para analizar la movilidad urbana.

Este simulador consta de una serie de detectores que pueden posicionarse en cualquier parte del mapa y permiten conocer el tipo de vehículo que circula por ellos y poder extraer una serie de información en tiempo real empleando la interfaz *TraCI*, fundamental en la elaboración de este proyecto.

2.5.3 TraCI

TraCI es una interfaz de comunicación que permite controlar, monitorear y modificar en tiempo real simulaciones de tráfico en *SUMO* mediante *scripts* en *Python*, facilitando la recopilación de datos como la velocidad, posición y trayectoria de los vehículos, información esencial para el análisis de las comunicaciones V2V [46].

Además, *TraCI* permite modificar variables durante la simulación, posibilitando ajustar la velocidad de los vehículos, modificar rutas, activar o desactivar semáforos y simular eventos imprevistos, como accidentes o condiciones meteorológicas adversas.

En el contexto de este proyecto, *TraCI* es fundamental en la recopilación de datos esenciales para el análisis de las comunicaciones V2V, permitiendo evaluar la influencia de factores como la distancia entre vehículos, la velocidad relativa y la densidad del tráfico en la estabilidad de las conexiones. Al integrarse con técnicas de *ML* y *DL*, esta interfaz facilita el desarrollo de modelos predictivos capaces de optimizar las comunicaciones V2V y mejorar la eficiencia de los sistemas de transporte inteligente.

2.5.4 Machine learning y deep learning

Las técnicas de *ML* y *DL* se utilizan para modelar y predecir la duración de las sesiones de comunicación V2V.

Machine learning, o aprendizaje automático, es una rama de la inteligencia artificial (IA) que permite a los sistemas aprender y mejorar su rendimiento a partir de datos sin ser programados explícitamente. En lugar de seguir reglas predefinidas, los modelos de *ML* identifican patrones en los datos y hacen predicciones basadas en ellos [47].

A diferencia de los algoritmos tradicionales, que requieren instrucciones explícitas para cada tarea, los modelos de *ML* pueden generalizar a partir de ejemplos previos. Esto los hace

especialmente útiles en aplicaciones donde el volumen de datos es grande y complejo, como en la predicción en comunicaciones vehiculares.

Hay tres tipos principales de aprendizaje automático [48]:

1. Aprendizaje Supervisado.

El aprendizaje supervisado es un enfoque en el que el modelo aprende a partir de un conjunto de datos etiquetado, es decir, un conjunto en el que cada entrada cuenta con una respuesta esperada o etiqueta de salida. Este tipo de aprendizaje se emplea en tareas donde es posible definir claramente la relación entre los datos de entrada y la salida deseada, permitiendo que el modelo generalice y realice predicciones sobre nuevos datos no vistos. Los algoritmos más utilizados en el aprendizaje supervisado se encuentran la Regresión Lineal, utilizada en problemas de predicción numérica; los Support Vector Machines (SVM), eficaces en problemas de clasificación con datos no lineales; los Árboles de Decisión y Random Forest, que combinan múltiples árboles para mejorar la precisión; y las Redes Neuronales Artificiales (ANN, *Artificial Neural Networks*), capaces de modelar relaciones complejas entre los datos.

2. Aprendizaje No Supervisado.

El aprendizaje no supervisado es un enfoque en el que el modelo trabaja con datos no etiquetados, es decir, sin una salida predefinida. En lugar de aprender a partir de ejemplos con respuestas correctas, el modelo debe identificar patrones, estructuras o relaciones ocultas en los datos de forma autónoma. Este tipo de aprendizaje es especialmente útil cuando no se dispone de etiquetas previas o cuando se busca descubrir información relevante sin intervención humana. Los algoritmos más utilizados en el aprendizaje no supervisado se encuentran K-means, un método de agrupamiento basado en la proximidad de los datos; DBSCAN, un algoritmo robusto para detectar agrupaciones de diferentes formas y tamaños; y Autoencoders, una arquitectura basada en redes neuronales profundas utilizada para la reducción de dimensionalidad y la detección de anomalías.

3. Aprendizaje por Refuerzo

El aprendizaje por refuerzo es un enfoque en el que un agente aprende a tomar decisiones óptimas mediante un proceso de prueba y error. En este modelo, el agente interactúa con un entorno y recibe recompensas cuando toma decisiones correctas y penalizaciones cuando comete errores. Con el paso del tiempo, el agente ajusta su estrategia para maximizar la recompensa acumulada, logrando un comportamiento cada vez más eficiente. Los algoritmos más utilizados en el aprendizaje por refuerzo se encuentran Q-Learning, un método basado en tablas que aprende la mejor acción en cada estado del entorno; Deep Q-Networks (DQN), que combina Q-Learning con redes neuronales profundas para manejar espacios de estados más complejos.

Deep Learning, o aprendizaje profundo, es una evolución de las ANN que se caracteriza por utilizar redes neuronales profundas con múltiples capas ocultas, lo que le permite modelar

relaciones más complejas en los datos con una mayor precisión. Estas redes pueden extraer automáticamente, reduciendo el preprocesamiento manual y mejorando el rendimiento [49].

Los tipos de redes más empleados son:

- Redes Neuronales Convolucionales (*CNN, Convolutional Neural Networks*), utilizadas principalmente en visión por ordenador.
- Redes Neuronales Recurrentes (*RNN, Recurrent Neural Networks*), diseñadas para el procesamiento de secuencias y datos temporales.
- Redes *LSTM (Long Short-Term Memory)*, una variante de las *RNN* optimizadas para manejar dependencias a largo plazo en los datos.

Los algoritmos empleados en este proyecto son algoritmos de *ML* que se basan en el aprendizaje supervisado. Los algoritmos son los siguientes:

- Regresión Lineal Múltiple: Modelo estadístico utilizado para identificar relaciones lineales entre múltiples variables independientes y una variable dependiente.
- Máquinas de Soporte Vectorial o *Support Vector Regression (SVR)*: Algoritmo de aprendizaje supervisado que utiliza hiperplanos para realizar predicciones precisas, incluso en escenarios complejos.
- Árboles de Decisión o Decision Tree: Modelo predictivo basado en reglas que permite dividir el conjunto de datos en ramas para facilitar la toma de decisiones.
- Bosques Aleatorios o Random Forest: Conjunto de árboles de decisión que mejora la precisión mediante la combinación de múltiples árboles y el uso de técnicas de *bagging (Bootstrap Aggregating)*, las cuales mejoran la precisión y estabilidad al emplear múltiples predicciones en paralelo. En este modelo en concreto, se generan múltiples árboles de decisión utilizando muestras del conjunto de datos original. A continuación, cada árbol emplea una selección aleatoria de características en cada nodo, lo que introduce una mayor variabilidad, mejorando la diversidad de los modelos.

Las predicciones finales se realizan combinando los resultados de todos los árboles, ya sea por voto mayoritario (clasificación) o por promedio (regresión).

- Redes Neuronales Artificiales (*ANN*): Modelos inspirados en el cerebro humano capaces de aprender patrones no lineales complejos. Su estructura básica consta de una serie de neuronas organizadas por capas. La primera capa es la capa de entrada, estas primeras neuronas reciben los datos del problema. Después, los datos pasan a las capas ocultas donde se procesa la información mediante funciones de activación no lineales. Por último, la capa de salida genera el resultado final.

Estas técnicas permiten analizar grandes volúmenes de datos, identificar patrones complejos y predecir la duración de las sesiones de comunicación.

Las tecnologías descritas en este apartado permiten un análisis integral del comportamiento de las comunicaciones V2V, proporcionando resultados realistas y aplicables a entornos urbanos complejos.

3. Especificaciones y restricciones de diseño

Todo desarrollo se debe plantear como un problema de ingeniería a solventar. Por lo tanto, existirán unas especificaciones iniciales, que determinarán la metodología y la solución. Las especificaciones estarán relacionadas con los objetivos del proyecto.

En este apartado se plantean las restricciones que se han tenido en cuenta en el desarrollo de este proyecto.

Este proyecto se enfoca en predecir la duración de las sesiones de comunicación V2V mediante la implementación de algoritmos de *machine learning* y *deep learning*. A continuación, se detallan las especificaciones y restricciones de diseño.

3.1 Especificaciones Iniciales

El desarrollo de este proyecto parte de una serie de especificaciones iniciales que han definido la metodología y las soluciones técnicas implementadas. Estas especificaciones están alineadas con los objetivos técnicos y académicos del proyecto:

- **Precisión en la predicción:** Desarrollar modelos capaces de predecir la duración de las sesiones en la comunicación entre vehículos (V2V).
- **Realismo en los escenarios:** Simular entornos urbanos realistas mediante el uso de datos geoespaciales de *OpenStreetMap (OSM)* e integrarlos en *SUMO*.
- **Obtención de datos relevantes:** Extraer datos esenciales como velocidad, posición, carril, ruta, potencia recibida (P_r), potencia de transmisión (P_t), potencia de ruido (P_{noise}) y relación señal-ruido (SNR).
- **Optimización del tráfico y la seguridad:** Contribuir a la mejora de la eficiencia del tráfico y la seguridad vial mediante la predicción de la estabilidad de las comunicaciones vehiculares.

3.2 Metodología

Para alcanzar los objetivos planteados, se ha seguido una metodología estructurada en varias fases:

1. **Generación de escenarios realistas:** Se han utilizado mapas extraídos de *OpenStreetMap (OSM)* para crear entornos urbanos detallados y precisos. Estos mapas fueron adaptados al simulador *SUMO* para la generación de datos de tráfico realistas.
2. **Simulación de tráfico:** Se implementaron escenarios de tráfico en *SUMO*, incluyendo condiciones con y sin semáforos, para analizar el impacto de la regulación del tráfico sobre las comunicaciones V2V.

3. *Captura de datos en tiempo real*: Utilizando la interfaz *TraCI*, se recolectaron datos clave sobre el comportamiento de los vehículos en la simulación, como la velocidad, posición, distancia a intersecciones, carril y ruta de cada vehículo.
4. Modelado físico de la comunicación: Se integraron parámetros físicos como la potencia de transmisión, atenuación de la señal, ruido y *SNR* para calcular la duración de las sesiones de comunicación.
5. Generación del *dataset*: Se consolidaron los datos obtenidos en un conjunto de datos estructurado, incorporando cálculos del tiempo de comunicación y condiciones del entorno.
6. Desarrollo de modelos predictivos: Se entrenaron y evaluaron diversos modelos de *ML* y *DL*, incluyendo regresión lineal múltiple, *Support Vector Regression (SVR)*, árboles de decisión, Random Forest y redes neuronales artificiales (*ANN*).
7. Evaluación de resultados: Los modelos fueron validados mediante métricas como el error absoluto medio (*MAE*), el error cuadrático medio (*MSE*) y el coeficiente de determinación (R^2) para determinar su eficacia.

3.3. Solución Propuesta

La solución propuesta integra simulaciones de tráfico realistas con modelos de *Machine Learning* y *Deep Learning* para predecir la duración de las sesiones de comunicación V2V. Este enfoque permite analizar cómo factores físicos y dinámicos del tráfico afectan la estabilidad de las comunicaciones.

La combinación de herramientas como: *OpenStreetMap*, *SUMO*, librería *TraCI* y modelos predictivos ha permitido analizar el comportamiento de los vehículos en entornos urbanos, suburbanos y autopistas con una alta densidad de vehículos. Estos resultados contribuyen a optimizar la gestión del tráfico y mejorar la seguridad vial mediante la anticipación de fallos en la comunicación entre vehículos, con el fin de evitar posibles accidentes.

4. Descripción de la solución propuesta

Este apartado se pretende explicar los pasos que se han seguido para completar el proyecto, explicando las herramientas utilizadas y como se fueron solucionando los problemas para poder obtener el tiempo que dura la sesión de comunicación en V2V.

Como ya se ha definido en capítulos anteriores, la solución propuesta se basa en el diseño e implementación de un sistema de predicción para la duración de sesiones de comunicación V2V utilizando modelos de *machine learning* y *deep learning*. Este sistema emplea el estándar *DSRC*, IEEE 802.11p, como base para la comunicación, aprovechando su baja latencia en entornos urbanos y autopistas saturadas, empleando el protocolo *UDP* para la transmisión, por su baja latencia y su alta velocidad en el envío de datos y su capacidad de transmitir en Broadcast.

El proceso que se ha seguido se divide en las siguientes fases:

1. Obtención de mapas y creación de escenarios.
2. Simulaciones de los escenarios creados en *SUMO* empleando la librería *TraCI* para obtener datos relevantes.
3. Generar el *Dataset*.
4. Modelos predictivos empleados.

En los siguientes epígrafes se describen las fases en las que se ha dividido el proyecto.

4.1 Obtención de mapas y creación de escenarios

En este epígrafe se explica en detalle las herramientas empleadas para la obtención de los mapas y la creación de los escenarios que se han empleado durante el desarrollo del proyecto.

Se escogió Madrid como la ciudad para desarrollar este proyecto. Durante la investigación inicial, descubrí *OpenStreetMap (OSM)*, que me permitió exportar las partes necesarias del mapa. Para establecer los escenarios, seleccioné varias áreas de Madrid y las clasifiqué en tres tipos de entornos: urbanos, suburbanos y autopistas, combinando algunos de estos entornos en un solo escenario. Esta herramienta permite exportar los mapas en el formato *osm*. Sin embargo, este formato no permite trabajar con *SUMO* por lo que hay que cambiarle el formato a *net.xml*.

Para ello, a través del *cmd (command)* en la carpeta donde se encuentra el proyecto se ejecuta el comando que aparece en la Figura 9 desde el terminal [50]:

```
netconvert --osm-files PuertaAlcala.osm --output-file PuertaAlcala.net.xml --geometry.remove  
--roundabouts.guess --ramps.guess --junctions.join --tls.guess-signals
```

Figura 9: Comando para convertir formatos de archivos

Donde *netconvert* es una herramienta que trae SUMO que permite convertir el formato de los archivos, *--osm-files PuertaAlcala.osm* especifica el nombre del archivo *osm* que contiene los datos geoespaciales, *--output-file PuertaAlcala.net.xml* define el nombre del archivo de salida en el nuevo formato *net.xml*, *--geometry.remove* simplifica la geometría de las carreteras eliminando nodos innecesarios para optimizar las simulaciones, *--roundabouts.guess* detecta las rotondas en el mapa de entrada y las convierte a elementos funcionales en SUMO, *--ramps.guess* identifica rampas en el mapa y las ajusta para que se puedan emplear en la simulación, *--junctions.join* simplifica los nodos de las intersecciones, *--tls.guess-signals* representa los semáforos (*Traffic Light Systems, TLS*) en el nuevo formato.

A continuación, se añaden las estructuras que hay en el mapa con el comando que se muestra en la Figura 10:

```
polyconvert --net-file PuertaAlcala.net.xml --osm-files PuertaAlcala.osm --type-file typemap.xml  
-o PuertaAlcala.poly.xml
```

Figura 10: Comando para añadir estructuras

Donde *polyconvert* es una herramienta que trae SUMO que permite añadir polígonos, edificios, parques... al nuevo mapa, *--net-file PuertaAlcala.net.xml* especifica el archivo de red generado anteriormente con el comando *netconvert*, *--osm-files PuertaAlcala.osm* identifica el archivo de entrada que contiene la información de edificios, parques, zonas verdes... *--type-file typemap.xml* identifica los datos del archivo *osm* haciendo un mapeo que asocia las categorías de los elementos para poder implementarlos en las simulaciones donde *typemap.xml* contiene las categorías para hacer el mapeo, *-o PuertaAlcala.poly.xml* especifica el archivo de salida donde se guardan las estructuras generadas por este comando.

Una vez tenemos el mapa en el formato *net.xml* y las estructuras, el siguiente paso es generar los vehículos que recorrerán el mapa siguiendo unas rutas aleatorias. Para ello se ejecutan los comandos de la Figura 11 correspondientes al escenario de Puerta de Alcalá con semáforos [51]:

```
python randomTrips.py -n PuertaAlcala.net.xml -r bus_routes.rou.xml -o bus_trips.xml -e 1000 -p 30
--vehicle-class bus --trip-attributes="accel=\"0.8\"""

python randomTrips.py -n PuertaAlcala.net.xml -r truck_routes.rou.xml -o truck_trips.xml -e 1000 -p
15 --vehicle-class truck --trip-attributes="color=\"179,223,183\"""

python randomTrips.py -n PuertaAlcala.net.xml -r delivery_routes.rou.xml -o delivery_trips.xml -e
1000 -p 30 --vehicle-class delivery --trip-attributes="color=\"115,211,230\"""

python randomTrips.py -n PuertaAlcala.net.xml -r passenger_routes.rou.xml -o passenger_trips.xml
1000 -p 0.9 --vehicle-class passenger --trip-attributes="color=\"255,255,255\"""

python randomTrips.py -n PuertaAlcala.net.xml -r trailer_routes.rou.xml -o trailer_trips.xml -e 1000
-p 150 --vehicle-class trailer --trip-attributes="color=\"223,179,180\" accel=\"0.5\"""
```

Figura 11: Comandos para generar tráfico aleatorio

Donde *randomTrips.py* es un *script* de *Python* que también trae *SUMO* y permite generar rutas aleatorias, *-n PuertaAlcala.net.xml* especifica el archivo de red sobre el que se van a generar dichas rutas; *-r bus_routes.rou.xml* define, para el primer comando, el archivo de rutas para los autobuses; *-o bus_trips.xml* es el archivo de salida donde se guardaran las rutas, *-e 1000* corresponde al periodo de tiempo que se generaran los viajes (1000 segundos), *-p 30* indica cada cuantos segundos se genera un viaje (cada 30 segundos), *--vehicle-class bus* especifica el tipo de vehículos que se generan (autobuses para el primer comando), *--trip-attributes="accel=\"0.8\""* añade una aceleración de 0.8 m/s^2 para los autobuses.

El atributo *--trip-attributes="color=\"223,179,180\""* que aparece en el resto de los comandos le asigna un color a cada tipo de vehículos para que sea más sencillo distinguirlos durante la simulación.

El resto de los comandos generan rutas para camiones (*truck*), vehículos de reparto (*delivery*), vehículos de pasajeros (*passenger*) y trailers o vehículos con remolque (*trailer*).

Por último, antes de iniciar la simulación hay que crearse un nuevo archivo con el formato *sumo.cfg* el cual hace referencia a la configuración que tendrá la simulación. A este nuevo archivo hay que añadirle todos los archivos generados anteriormente, proporcionándole un tiempo de inicio y final. La Figura 12 muestra el contenido del nuevo archivo para la configuración del escenario de Puerta de Alcalá. El contenido completo del archivo de configuración se encuentra en el anexo:

```

<input>
  <net-file value="PuertaAlcala.net.xml"/>
  <route-files value="bus_routes.rou.xml,passenger_routes.rou.xml,trailer_routes.rou.xml,delivery_routes.rou.xml,truck_routes.rou.xml"/>
  <additional-files value="PuertaAlcala.poly.xml"/>
</input>

<output>
  <fcd-output value="vehicle_data.xml"/>
</output>

<time>
  <begin value="0"/>
  <end value="1000"/>
</time>

```

Figura 12: Contenido del archivo de configuración

- **Puerta de Alcalá**

El primer caso seleccionado fue el centro de Madrid, específicamente la calle Alcalá, desde Gran Vía hasta la Puerta de Alcalá. Aquí se consideraron dos escenarios: uno con los semáforos existentes y otro sin ellos, lo que permitió analizar la evolución del tráfico en ambos casos, representando un entorno urbano en ambas simulaciones. En la Figura 13 se puede apreciar el tramo mencionado desde la aplicación SUMO:



Figura 13: Mapa de la Puerta de Alcalá

Las configuraciones del resto de escenarios que se muestran a continuación emplean los mismos comandos y la misma configuración que los empleados para el escenario de Puerta de Alcalá, siguiendo el mismo orden descrito anteriormente. La única modificación que se realizó fue en el intervalo de generación de los distintos vehículos para ajustarlo mejor a las características que requería cada escenario.

Para el caso de sin semáforos se crearon dos escenarios distintos: uno con tráfico normal y otro generando un alto flujo de vehículos para tener un escenario más realista del centro de Madrid en hora punta para ver las diferencias entre ambos escenarios:

➤ Tráfico normal

En la Figura 14 se muestran los comandos empleados para crear este escenario:

```
python randomTrips.py -n PuertaAlcala.net.xml -r bus_routes.rou.xml -o bus_trips.xml -e 1000 -p 15
--vehicle-class bus --trip-attributes="accel=\"0.8\"""

python randomTrips.py -n PuertaAlcala.net.xml -r truck_routes.rou.xml -o truck_trips.xml -e 1000 -p
40 --vehicle-class truck --trip-attributes="color=\"179,223,183\"""

python randomTrips.py -n PuertaAlcala.net.xml -r delivery_routes.rou.xml -o delivery_trips.xml -e
1000 -p 30 --vehicle-class delivery --trip-attributes="color=\"115,211,230\"""

python randomTrips.py -n PuertaAlcala.net.xml -r passenger_routes.rou.xml -o passenger_trips.xml
1000 -p 1.2 --vehicle-class passenger --trip-attributes="color=\"255,255,255\"""

python randomTrips.py -n PuertaAlcala.net.xml -r trailer_routes.rou.xml -o trailer_trips.xml -e 1000
-p 150 --vehicle-class trailer --trip-attributes="color=\"223,179,180\" accel=\"0.5\"""
```

Figura 14: Comandos para generar el tráfico en Puerta de Alcalá

➤ Tráfico lento

La Figura 15 muestra los comandos que se han utilizado para agregarle más tráfico al escenario:

```
python randomTrips.py -n PuertaAlcala.net.xml -r bus_routes.rou.xml -o bus_trips.xml -e 1000 -p 15
--vehicle-class bus --trip-attributes="accel=\"0.8\"""

python randomTrips.py -n PuertaAlcala.net.xml -r truck_routes.rou.xml -o truck_trips.xml -e 1000 -p
40 --vehicle-class truck --trip-attributes="color=\"179,223,183\"""

python randomTrips.py -n PuertaAlcala.net.xml -r delivery_routes.rou.xml -o delivery_trips.xml -e
1000 -p 30 --vehicle-class delivery --trip-attributes="color=\"115,211,230\"""

python randomTrips.py -n PuertaAlcala.net.xml -r passenger_routes.rou.xml -o passenger_trips.xml
1000 -p 0.8 --vehicle-class passenger --trip-attributes="color=\"255,255,255\"""

python randomTrips.py -n PuertaAlcala.net.xml -r trailer_routes.rou.xml -o trailer_trips.xml -e 1000
-p 150 --vehicle-class trailer --trip-attributes="color=\"223,179,180\" accel=\"0.5\"""
```

Figura 15: Comandos para aumentar el tráfico en Puerta Alcalá

En este nuevo escenario se le añaden más vehículos, pudiéndose observar como el tráfico aumenta considerablemente, creando retenciones más prolongadas en el tiempo para poder comprobar como este fenómeno actúa en la capa de las comunicaciones. Generando un escenario mucho más realista del centro de Madrid.

- **Sanchinarro**

Otro escenario fue Sanchinarro, incluyendo un pequeño tramo de autopista hasta el Corte Inglés. Al igual que en el caso anterior, se diseñaron escenarios con y sin semáforos, representando un entorno urbano. La Figura 16 muestra el mapa de Sanchinarro con la zona que se ha empleado para la simulación:

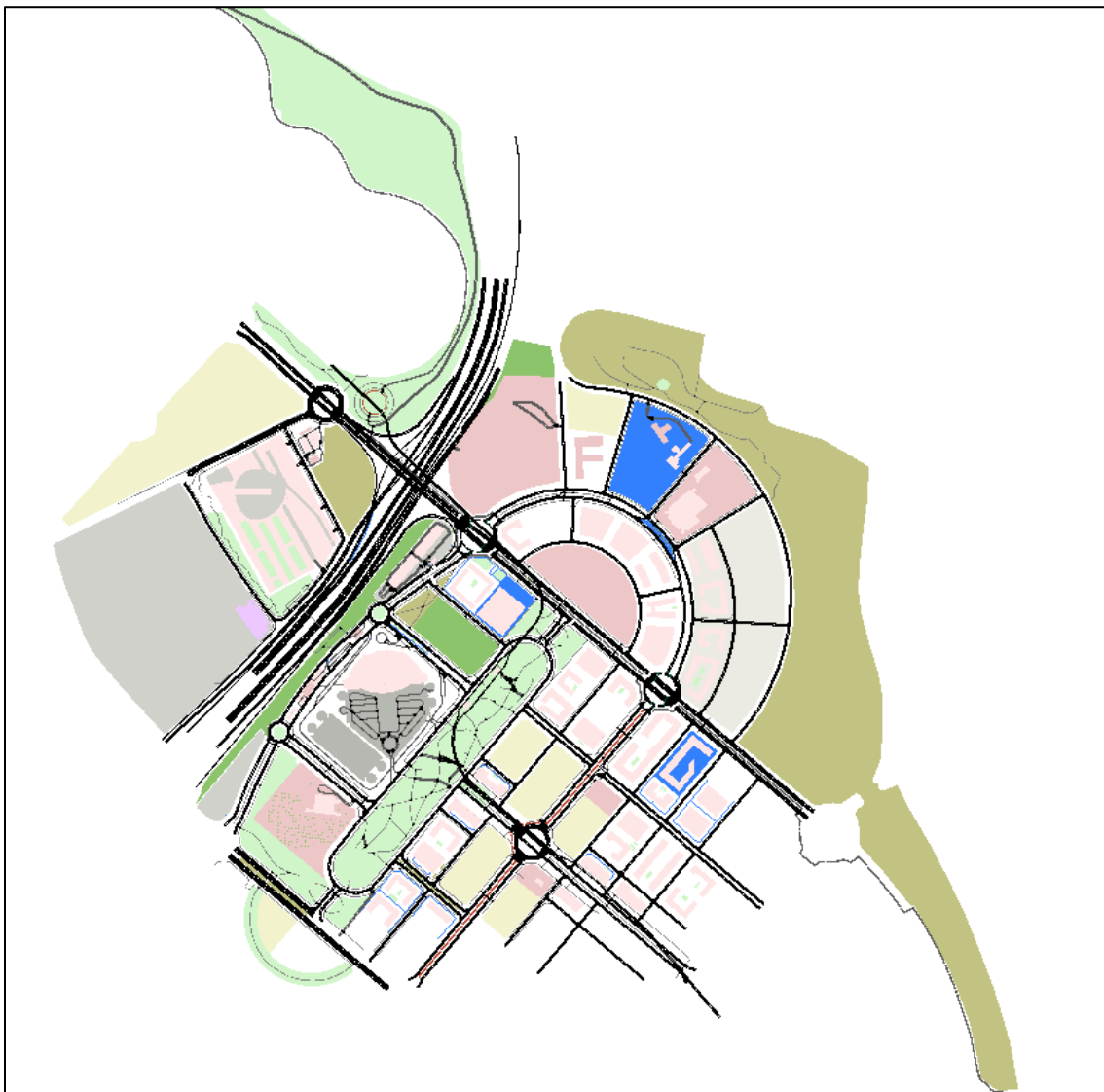


Figura 16: Mapa de Sanchinarro

En esta configuración se emplea los comandos de la Figuras 9 y 10 cambiando el nombre del archivo PuertaAlcala por SanchinarroCI (*Sanchinarro Corte Inglés*) para identificar este nuevo escenario con el mapa descargado de *OpenStreetMap (OSM)*.

El comando de la Figura 11 se mantiene igual que en el escenario de Puerta de Alcalá, pero sustituyendo el nombre de PuertaAlcala por SanchinarroCl.

- **Las Rosas**

La zona de Las Rosas, concretamente el tramo que conecta Las Rosas con Vicálvaro fue seleccionado por tener un entorno suburbano con varias rotondas que conectan tramos de carretera con incorporaciones a áreas suburbanas.

La Figura 17 muestra el mapa de Las Rosas - Vicálvaro, en concreto, el tramo que conecta ambos barrios mediante una serie de rotondas que se ubican sobre la M-40 y la R-3 que se ha empleado en este nuevo escenario:



Figura 17: Mapa de Las Rosas

Al igual que en los otros escenarios, se emplearon los comandos de las Figuras 9 y 10 para cambiar el formato al mapa descargado de *OpenStreetMap (OSM)*, modificando el nombre de PuertaAlcala por LasRosas para identificar el nuevo escenario. Sin embargo, para crear el nuevo escenario se emplearon los comandos modificando el intervalo de generación de los vehículos, como se muestra en la Figura 18:

```
python randomTrips.py -n LasRosas.net.xml -r bus_routes.rou.xml -o bus_trips.xml -e 1000 -p 30 --
vehicle-class bus --trip-attributes="accel=\"0.8\"""

python randomTrips.py -n LasRosas.net.xml -r truck_routes.rou.xml -o truck_trips.xml -e 1000 -p 15 -
-vehicle-class truck --trip-attributes="color=\"179,223,183\"""

python randomTrips.py -n LasRosas.net.xml -r delivery_routes.rou.xml -o delivery_trips.xml -e 1000 -
p 15 --vehicle-class delivery --trip-attributes="color=\"115,211,230\"""

python randomTrips.py -n LasRosas.net.xml -r passenger_routes.rou.xml -o passenger_trips.xml
1000 -p 0.7 --vehicle-class passenger --trip-attributes="color=\"255,255,255\"""

python randomTrips.py -n LasRosas.net.xml -r trailer_routes.rou.xml -o trailer_trips.xml -e 1000 -p
100 --vehicle-class trailer --trip-attributes="color=\"223,179,180\" accel=\"0.5\"""
```

Figura 18: Comandos para generar el tráfico en Las Rosas

- **Autopista A-2**

Finalmente, se consideró un tramo de la autopista A-2, desde Guadalajara hasta Torija, que combina entornos urbanos y de autopista con un alto volumen de tráfico. Esto ha permitido evaluar problemas de circulación, congestión de tráfico, y el comportamiento de los vehículos y sus comunicaciones.

El objetivo de este escenario fue utilizar un mapa más amplio que los anteriores para estudiar el impacto de una autopista en las comunicaciones cercanas a una gran ciudad. Este tramo integra diversos tipos de entornos durante un periodo de hora punta, lo que genera una alta densidad de vehículos y crea un escenario más realista. Esto derivó en tráfico lento en las entradas de Guadalajara y de varias localidades atravesadas por esta autopista.

Además, se incluyeron carreteras secundarias que añadieron realismo al análisis. Se observó que la elevada densidad de vehículos provocó obstrucciones en los accesos tanto a la ciudad como a los pueblos circundantes, complicando las comunicaciones y ralentizando el tráfico.

La Figura 19 muestra el mapa del tramo de la A-2 mencionado anteriormente:

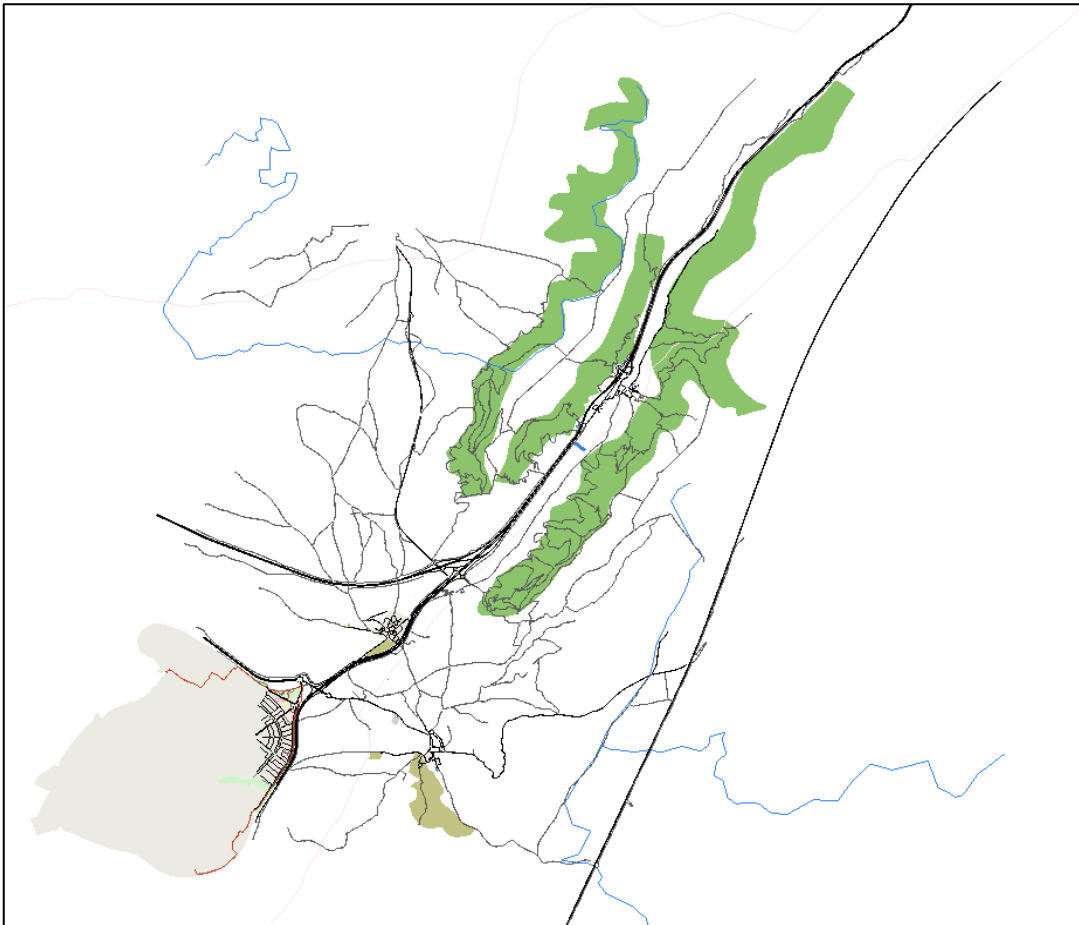


Figura 19: Mapa de la A-2

Al igual que en los otros escenarios, se hace uso de los comandos de las Figuras 9 y 10 para recrear el nuevo escenario. Este nuevo escenario se muestra en la Figura 20 para ajustarlo a los criterios de este nuevo escenario:

```
python randomTrips.py -n TramoA2.net.xml -r bus_routes.rou.xml -o bus_trips.xml -e 1000 -p 60 --
vehicle-class bus --trip-attributes="accel=\"0.8\""
```

```
python randomTrips.py -n TramoA2.net.xml -r truck_routes.rou.xml -o truck_trips.xml -e 1000 -p 7 --
vehicle-class truck --trip-attributes="color=\"179,223,183\""
```

```
python randomTrips.py -n TramoA2.net.xml -r delivery_routes.rou.xml -o delivery_trips.xml -e 1000 -
p 30 --vehicle-class delivery --trip-attributes="color=\"115,211,230\""
```

```
python randomTrips.py -n TramoA2.net.xml -r passenger_routes.rou.xml -o passenger_trips.xml
1000 -p 0.7 --vehicle-class passenger --trip-attributes="color=\"255,255,255\""
```

```
python randomTrips.py -n TramoA2.net.xml -r trailer_routes.rou.xml -o trailer_trips.xml -e 1000 -p
30 --vehicle-class trailer --trip-attributes="color=\"223,179,180\" accel=\"0.5\""
```

Figura 20: Comandos para generar el tráfico en A-2

A continuación, se muestra el nuevo escenario ampliado en varias imágenes para poder diferenciar de mejor manera las partes de las que consta este mapa.

En la Figura 21 se muestra la A-2 en su paso por la ciudad de Guadalajara ciudad observándose las entradas y salidas de esta ciudad, y la unión de las autopistas R-2 y A-2 tras su paso por Guadalajara. Se pueden distinguir una serie de carreteras secundarias que conectan esta ciudad con los pueblos de los alrededores, pudiéndose observar los entornos suburbanos y de carretera:

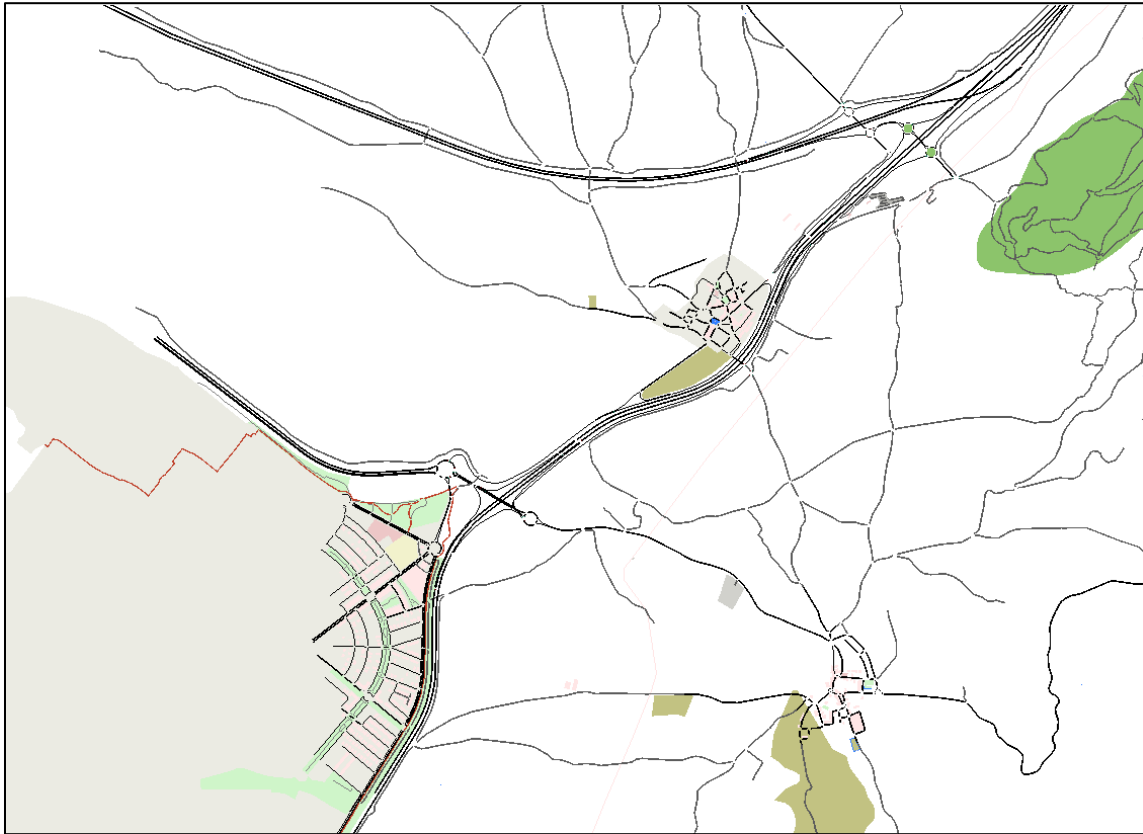


Figura 21: Mapa de ampliado de A-2, Guadalajara

En la siguiente imagen, la Figura 22, se aprecia la continuación de la A-2 después de haber dejado atrás la ciudad de Guadalajara:

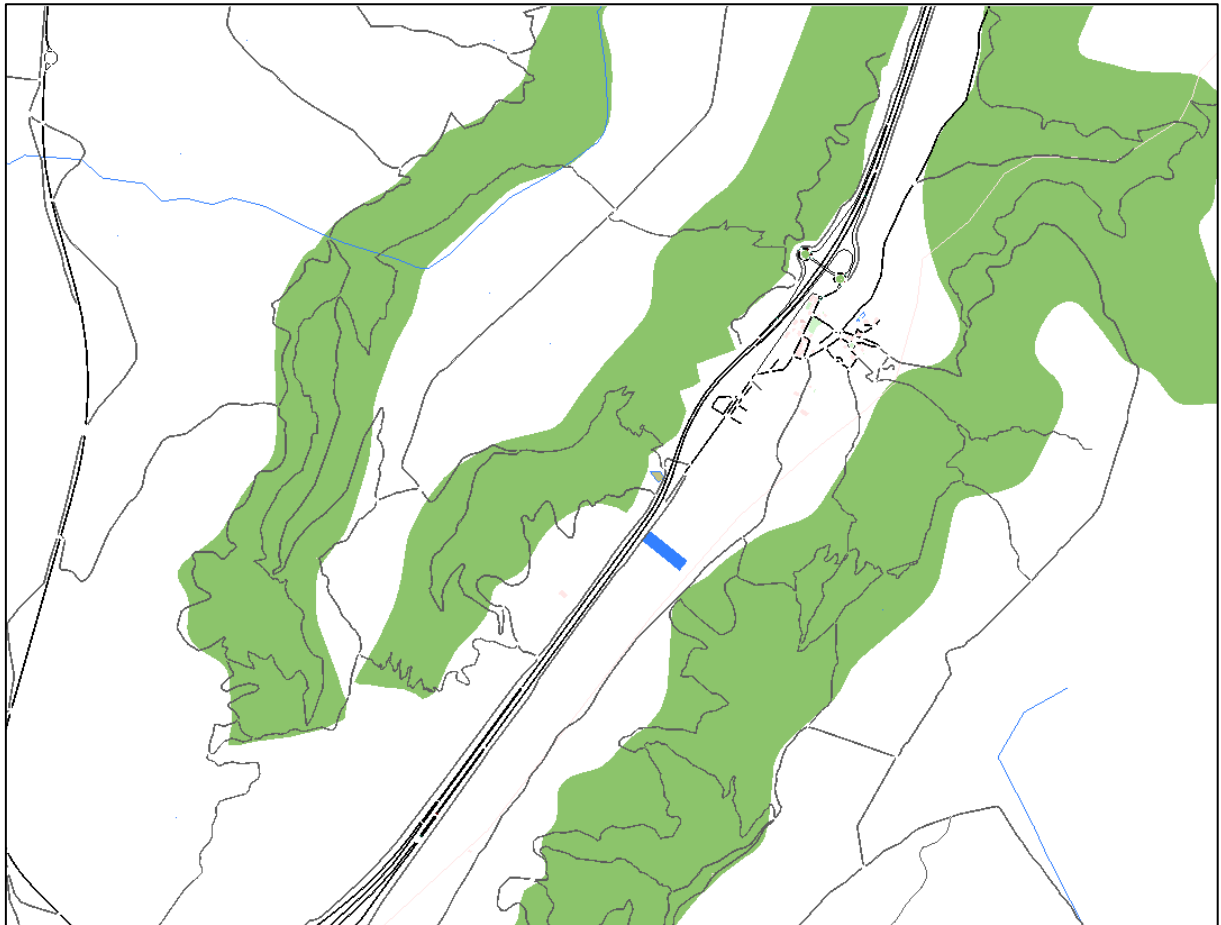


Figura 22: Mapa de A-2 ampliado, Valdenoches

En este nuevo tramo de la autopista se distinguen los entornos rurales (suburbanos) y el entorno de autopista, en su paso por la población de Valdenoches en su camino hacia Torija. Al igual que en la Figura 21, la Figura 22 muestra los diferentes entornos que componen esta simulación sobre la A-2.

Con las simulaciones creadas, se instalaron en diversos puntos del mapa una serie de detectores sobre los principales cruces para estudiar el comportamiento de los vehículos cuando se aproximan a esos puntos del mapa. Para ello se emplearon los detectores de entrada y salida, *Multi-Entry-Exit Detectors* (E3), y detectores de carriles, *Induction Loops Detectors* (E1), para detectar los vehículos [52][53].

4.2 Simulaciones de los escenarios

En este capítulo se describe el código que se ha empleado para obtener los datos de los vehículos en los escenarios descritos en el capítulo anterior.

Para obtener los datos relevantes se empleó la librería *TraCI*. Esta librería es una interfaz que permite interactuar con simulaciones de tráfico en tiempo real con el simulador *SUMO*.

Se ha hecho uso de esta librería por los siguientes motivos:

- Controlar simulaciones: Permite mantener el control sobre las simulaciones al poder observar los elementos dentro de la simulación.
- Recopilar datos: Permite extraer información detallada sobre la simulación como el nombre de los vehículos, las posiciones de cada uno, la velocidad que llevan, el carril por el que circulan y la ruta que van a realizar.

TraCI permite interactuar con los elementos de la simulación en tiempo real debido al protocolo que emplea cliente-servidor, donde *SUMO* actúa como servidor y el *script* de *Python* empleado actúa como cliente, permitiendo ejecutar el código en tiempo real durante el desarrollo de la simulación.

Para tratar los datos se emplean las librerías *ElementTree* y *Pandas* para crear y escribir archivos *XML* [54][55]. Además de poder importar archivos *CSV* entre otros que permite generar los datos necesarios de las simulaciones [56].

- **Conseguir información de los vehículos (*Conseguir_informacion_vehiculos.py*)**

Para realizar todo lo descrito anteriormente, *TraCI* junto con *SUMO* proporcionan una serie de funciones ya programadas en *Python* como las siguientes:

- La función *traci.start()* permite iniciar las instancias de *SUMO*.
- La función *traci.simulationStep()* avanza la simulación en un único paso de tiempo dentro de *SUMO*.
- La función *traci.vehicle.getPosition(veh_id)* obtiene la posición de un vehículo.
- La función *traci.vehicle.getSpeed(veh_id)* permite obtener la velocidad de un vehículo.
- La función *traci.vehicle.getLaneID(veh_id)* permite obtener el carril por el que circula el vehículo.
- La función *traci.vehicle.getRoute(veh_id)* permite conocer la ruta que va realizar el vehículo.
- La función *traci.close()* cierra la simulación iniciada desde el *script*.

Todas estas funciones y muchas más se encuentran en la documentación para el usuario que tiene *SUMO* [46]. La Figura 23 muestra algunos de las funciones mencionadas anteriormente sobre el *script* de *Python* *Conseguir_informacion_vehiculos.py* que se ha desarrollado para poder obtener la información en tiempo real:

```
# Vehículos dentro del detector
for veh_id in vehicles:
    veh_position = traci.vehicle.getPosition(veh_id)
    distance_to_detector = sqrt((veh_position[0] - det_position[0]) ** 2 + (veh_position[1] - det_position[1]) ** 2)
    detector_info["detected_vehicles"].append({
        "vehicle_id": veh_id,
        "distance_to_detector": round(distance_to_detector),
        "position_x": veh_position[0],
        "position_y": veh_position[1],
        "speed": round(traci.vehicle.getSpeed(veh_id)),
        "lane": traci.vehicle.getLaneID(veh_id),
        "route": traci.vehicle.getRoute(veh_id)
    })
```

Figura 23: Información extraída de cada vehículo

En este mismo código se define una función *find_nearby_vehicles()*, a la que se le pasa como parámetros de entrada: una lista con las posición de los detectores (*detector_positions*) y una distancia máxima de 50 metros (*max_distance=50*). Esta función es esencial puesto que devuelve un diccionario donde cada detector tiene una lista con los vehículos cercanos (vehículos que se encuentren a 50 metros o menos del detector) con información detallada sobre cada uno.

Al ejecutar este código se abre la aplicación de *SUMO* y comienza la simulación. Durante la simulación se van almacenando los datos en el diccionario donde:

- Se crea una lista para cada detector.
- Se le agregan los vehículos que cumplen las condiciones de distancia al detector.
- Para cada vehículo que cumple las condiciones, se extrae información acerca de: identificador del vehículo (*veh_id*), la distancia al detector en ese momento (*distance_to_detector*), las posiciones del vehículo (*position_x* y *position_y*), la velocidad (*speed*), el carril por el que están circulando (*lane*) y la ruta que van a realizar (*route*).
- Por último, se crea el archivo de salida *XML* donde se muestra la información extraída de la simulación.

Una vez acaba la simulación y se obtiene el nuevo archivo *XML* con la información detallada de cada detector, se desarrolló un nuevo código que se centrara en la capa física de las comunicaciones.

- **Calcular la potencia de los vehículos (*calculo_potencia_vehiculos_cercanos.py*)**

El nuevo *script* de *Python* *calculo_potencia_vehiculos_cercanos.py* realiza cálculos de la capa física de las comunicaciones considerando los elementos del entorno. Para ello empecé por definir los parámetros del canal:

- Se asigno una frecuencia de transmisión (*f*) de 5,9 GHz.
- Se estableció la potencia de transmisión (*P_t*) en 20 dBm.
- El ancho de banda del canal (*BANDWIDTH*) en 10 MHz y una temperatura estándar (*TEMPERATURE*) de 290 K (kelvin).
- Constantes para calcular la pérdida de trayectoria basados en el modelo experimental: exponente de pérdidas de trayectoria (*PATHLOSS_EXPONENT*) y pérdida a una distancia de referencia en dB (*PL_D0*) a la distancia de referencia (*D0*) de 1 metro. Los distintos entornos con sus exponentes de pérdidas se muestran en la Tabla 4 [34].

Tabla 4: Modelo empírico pérdidas según el escenario

Escenarios	Modelo de pérdidas por trayectoria
Autopista	$n = 1.77$ $PL(d_0) = 72.5 \text{ dB}$
Urbano	$n = 1.68$ $PL(d_0) = 74.2 \text{ dB}$
Suburbano	$n = 1.53$ $PL(d_0) = 78.4 \text{ dB}$

Después de definir el canal se definieron varias funciones para evaluar las condiciones del entorno. La primera función, *calculate_pathloss()*, calcula la pérdida por trayectoria en función de la distancia siguiendo el modelo experimental basado en los modelos de log-distancia y log-normal. Empelando la Expresión 4.1:

$$Pathloss = PL_{D_0} + 10 * exponente * \log_{10} \left(\frac{distancia}{D_0} \right) \quad (4.1)$$

Donde PL_{D_0} hace referencia a la constante de pérdida de trayectoria descrita anteriormente, *exponente* hace referencia a la constante de pérdidas y D_0 es la distancia de referencia.

La siguiente función, *load_obstacles()*, carga el archivo con la extensión poly.xml que contiene los obstáculos del mapa. Hace uso de *ElementTree* para analizar el archivo XML con polígonos. Además, se utiliza la librería *Shapely* para convertir los puntos definidos en polígonos para calcular si la línea de visión (LoS) de un vehículo está obstruida por algún elemento.

La función *shadowing_loss()* calcula la atenuación por desvanecimiento (*shadowing*) producida por los obstáculos. Se define la línea de visión entre el vehículo y el detector (situado sobre una intersección) añadiéndole unas pérdidas para cada obstáculo que se encuentre en la línea de visión.

Se define una última función antes de la función principal que calcula la potencia de ruido del canal, *calculate_noise()*. Además, se agrega una interferencia por la presencia de vehículos en el entorno. Esta función hace uso de la fórmula del ruido térmico que se muestra en la Expresión 4.2:

$$P_{noise} = k * T * B \quad (4.2)$$

Donde P_{noise} es la potencia de ruido, k es la constante de Boltzmann donde $k = 1.380649 * 10^{-23} \text{ (J/K)}$, T es la temperatura en Kelvin y B es el ancho de banda del canal en Hz.

Se emplean las funciones descritas anteriormente en la función principal, *process_vehicles()*. La función principal procesa los datos por cada paso de tiempo (*step*) de cada detector con la información detallada sobre los vehículos, agregando los cálculos de pérdida de trayectoria (*Path Loss*), pérdida por ensombrecimiento (*shadowing*). Además, se le añaden el cálculo de la potencia recibida (P_r) como se muestra en la Expresión 4.3:

$$P_r = P_t - \text{Pathloss} - \text{Shadowing} \quad (4.3)$$

Donde P_t y P_r corresponden con la potencia transmitida y la potencia recibida respectivamente.

Y se calcula la relación señal a ruido (SNR) con la fórmula que se muestra en la Expresión 4.4:

$$SNR = P_r - P_{noise} \quad (4.4)$$

Donde P_{noise} es la potencia de ruido.

Para calcular la potencia recibida se restan el *pathloss* y el *shadowing* porque los valores que obtienen estas variables de sus respectivas funciones son valores positivos. Razón por la cual se restan para aumentar su atenuación en lugar de sumarse.

Por último, se agregan los nuevos elementos al diccionario y se guardan en un archivo *XML*. En el siguiente apartado se explica que herramientas se han empleado para generar el *dataset*.

4.3 Dataset

Antes de describir los modelos de *machine learning* y *deep learning* empleados se genera el *dataset* que se ha empleado para estos modelos.

Para ello, se tuvo que formatear los archivos *XML* generados en los apartados anteriores para que tuvieran un formato legible. Para ello se empleó la herramienta “*XML formatter*” que permite formatear archivos de todos los tamaños, puesto que hay otras herramientas que no lo permiten. Esta herramienta es una extensión que se le puede añadir al buscador *Google Chrome*.

Una vez formateados los archivos se empleó un código en *Python* para filtrar la información y generar el *dataset* que después se implementa en los modelos de *ML* y *DL*.

En este código se implementa una función que calcula el tiempo que permanece abierta la sesión de comunicación. Para ello, se determina umbral de relación señal a ruido (SNR) de 15 dB con un margen para el enlace de 5 dB para evitar que se cierre la conexión por desvanecimientos puntuales [57]. Quedando finalmente un umbral de 10 dB.

Se establece que la sesión de comunicación se abre cuando el vehículo empieza a ser detectado (cuando la distancia al detector es inferior a 50 metros) y se cierra cuando deja de ser detectado (se aleja a más de 50 metros del detector) o cuando su relación señal a ruido (SNR) cae por debajo del umbral, en este caso cae por debajo de 10 dB.

Con todo lo anterior, el nuevo dataset consta de las siguientes columnas:

- *Time*: muestra el segundo de la simulación cuando fue detectado.
- *Detector_id*: muestra el nombre del detector.

- *Nearby_vehicle_count*: muestra los vehículos que se encuentran en un radio de 50 metros del vehículo detectado.
- *Vehicle_id*: muestra el nombre del vehículo detectado.
- *Disatnce*: muestra la distancia del vehículo detectado al detector.
- *Pos_x* y *Pos_y*: muestra la posición del vehículo en el mapa.
- *Speed*: muestra la velocidad del vehículo en m/s.
- *Lane*: muestra el carril por el que circula el vehículo detectado.
- *Route*: muestra la ruta que va a realizar el vehículo detectado.
- *Pt*: muestra la potencia de transmisión de cada vehículo (20 dBm).
- *Pr*: muestra la potencia de recepción del vehículo detectado.
- *Pniose*: muestra la potencia de ruido del vehículo detectado.
- *SNR*: muestra la relación señal a ruido del vehículo detectado.
- *Escenario*: muestra el escenario del vehículo.
- *Session_duration*: muestra el tiempo que dura la sesión de comunicación.

El dataset consta de un total de 16 columnas y 561.923 filas en las que se unifican todos los escenarios, ocupando un total de 291.415 KB o 291,4 MB.

Con el dataset creado, el siguiente paso es generar los modelos de *machine learning* y *deep learning* que van a emplear el *dataset* con el que se evaluará su rendimiento.

4.4 Modelos predictivos

En este apartado se explican cuáles han sido los modelos predictivos que se han empleado para tratar de predecir el tiempo que dura la sesión de comunicación.

Se han empleado varios modelos distintos para evaluar qué modelo es el que mejor se ajusta a las necesidades de este proyecto, empezando por modelos más sencillos y acabando con modelos de una mayor complejidad. Para todos los modelos que se mencionan a continuación se ha dividido el *dataset* en un 80% para entrenamiento y un 20% para prueba, evaluando todos los modelos con las siguientes métricas [58]:

- Error cuadrático medio (*MSE*, *Mean Square Error*).
- Error absoluto medio (*MAE*, *Mean Absolute Error*).
- Coeficiente de determinación (R^2).

Además, se emplean las librerías de *sklearn* para importar los modelos y la librería *matplotlib* para poder visualizar los resultados en una gráfica y sea más sencillo su posterior análisis. Los modelos empleados son los siguientes:

- **Regresión Lineal Múltiple**

El modelo de Regresión lineal múltiple es un modelo estadístico que estudia la relación entre una variable dependiente y dos o más variables independientes. Este modelo permite considerar múltiples factores que afectan una variable mediante relaciones lineales [59].

La Figura 24 muestra como la relación de linealidad entre los datos afecta al modelo de *Regresión Lineal* [60]:

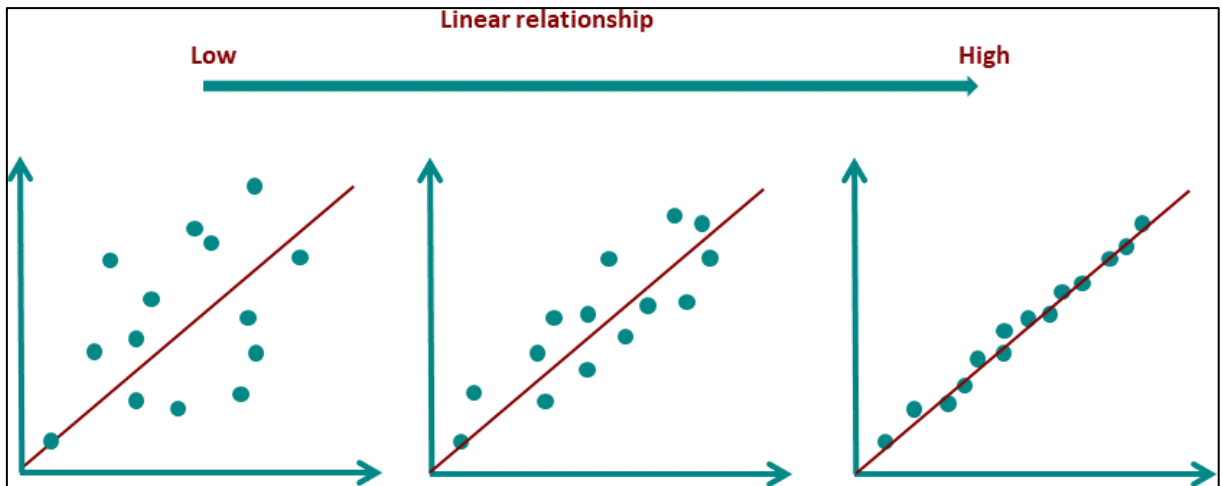


Figura 24: Modelo de Regresión Lineal Múltiple

Para el desarrollo de este proyecto se ha tomado como variable dependiente la duración de la sesión de comunicación, *sesión_duration*, y como variables independientes: *nearby_vehicle_count*, *distance*, *pos_x*, *pos_y*, *speed*, *Pt*, *Pr*, *Pnoise* y *SNR*.

La Tabla 5 muestra las variables que se han empleado en este modelo:

Tabla 5: Variables empleadas en los modelos

Variables	Descripción
<i>nearby_vehicle_count</i>	Número de vehículos cercanos
<i>distance</i>	Distancia al vehículo objetivo
<i>pos_x</i>	Posición en el eje X
<i>pos_y</i>	Posición en el eje Y
<i>speed</i>	Velocidad del vehículo
<i>Pt</i>	Potencia de transmisión
<i>Pr</i>	Potencia recibida
<i>Pnoise</i>	Potencia de ruido
<i>SNR</i>	Relación señal a ruido
<i>session_duration</i>	Duración de la sesión de comunicación

A continuación, se emplea el modelo de regresión lineal importando de *sklearn* la librería *LinearRegression* que nos permite utilizar este modelo.

- **Máquinas de Soporte Vectorial (SVR, Support Vector Regression)**

Las Máquinas de Soporte Vectorial para Regresión (SVR) son una extensión de las Máquinas de Vectores de Soporte (SVM) aplicada a problemas de regresión, donde la variable objetivo es continua en lugar de categórica. Se emplea un margen de tolerancia sobre el que se realiza una regresión lineal [61].

La Figura 25 muestra el funcionamiento del modelo SVR [62]:

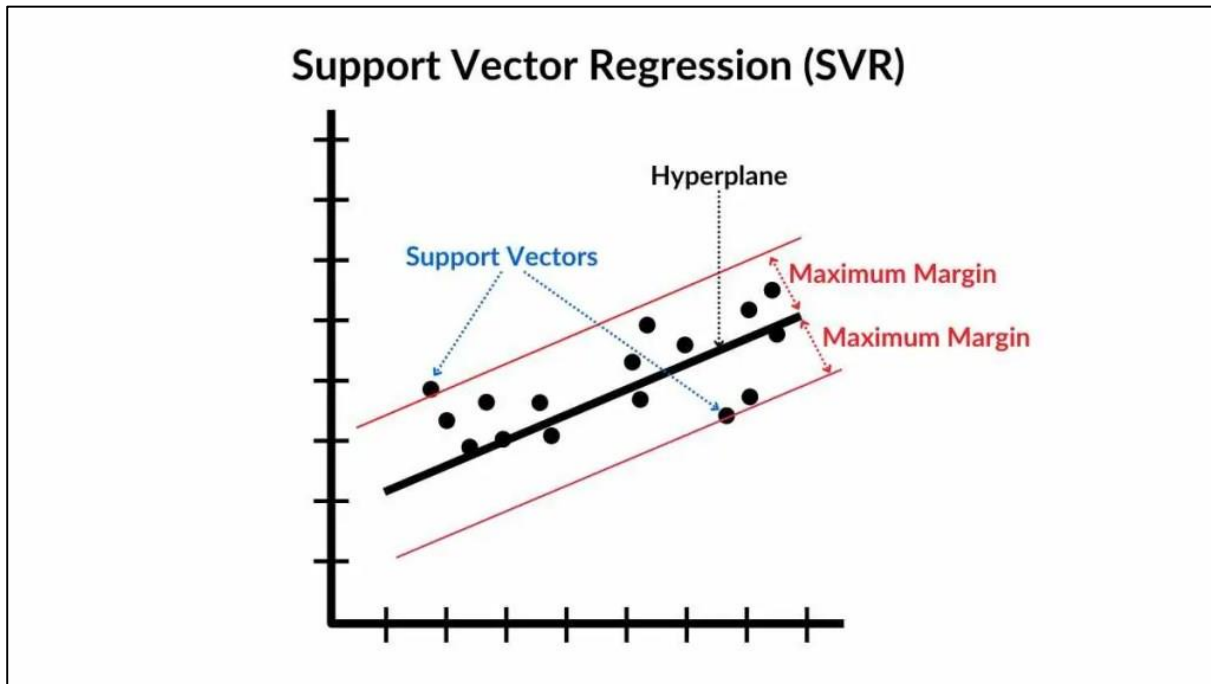


Figura 25: Modelo SVR

En este modelo se emplea la variable que calcula el tiempo que dura la sesión de comunicación para entrenar y probar el modelo. Para poder emplear este modelo, se importa de *sklearn.svm* la librería SVR. La Figura 26 muestra como se ha empleado el modelo importado de la librería *sklearn.svm*:

```
# Modelo SVM
svm_model = SVR(kernel='rbf', C=1.0, epsilon=0.1)
svm_model.fit(X_train_scaled, y_train)
```

Figura 26: Código del Modelo SVR

Donde *kernel='rbf'* indica que se emplea un *kernel* de función base radial, se trata de un *kernel* no lineal. *C=1.0* es el parámetro de regularización que controla el equilibrio entre ajustarse bien a los datos de entrenamiento y mantener el modelo lo suficientemente simple para generalizar bien. Y *epsilon=0.1* define el margen de tolerancia en el que no se penalizan los errores. Determina el ancho del "tubo" alrededor de la función de regresión.

- **Árboles de decisión (*decision trees*)**

Los Árboles de decisión son un algoritmo de aprendizaje autónomo empleado para tareas de clasificación y de regresión. Funciona dividiendo el espacio de entrada en regiones más pequeñas, buscando maximizar la separación entre clases o valores. Su estructura consiste en un gráfico tipo árbol con los siguientes elementos [63]:

- **Nodo raíz:** Representa la decisión final.
- **Nodos internos:** Representan decisiones sobre variables de entrada.
- **Ramas:** Conectan los nodos y representan los posibles resultados de cada decisión.
- **Nodos hoja:** Representan las predicciones o etiquetas de clase finales

La Figura 27 muestra la estructura de un Árbol de decisión [64]:

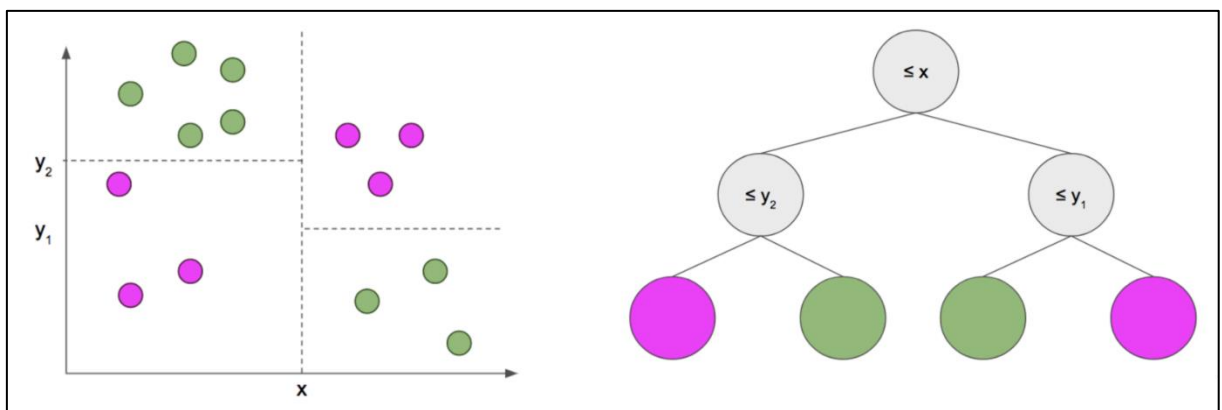


Figura 27: Modelo Árbol de Decisión

Al igual que para los modelos anteriores, se emplea la duración de la sesión como variable dependiente de las variables independientes ya mencionadas. Para emplear este modelo, se importa de *sklearn.tree* la librería *DecisionTreeRegressor*.

La Figura 28 muestra la forma en que se ha empleado la librería *DecisionTreeRegressor*:

```
# Modelo de Árbol de Decisión
tree_model = DecisionTreeRegressor(random_state=42)
tree_model.fit(X_train_scaled, y_train)
```

Figura 28: Código del Modelo Árbol de Decisión

- **Bosques Aleatorios (*Random forests*)**

Los Bosques Aleatorios son un algoritmo de aprendizaje autónomo que combina múltiples árboles de decisión para realizar tareas de clasificación y regresión. Son un conjunto de árboles de decisión donde cada árbol se construye utilizando una muestra aleatoria de los datos de entrenamiento, aportando una mayor precisión que el modelo anterior.

La Figura 29 muestra gráficamente el funcionamiento del modelo *Random Forest* [65]:

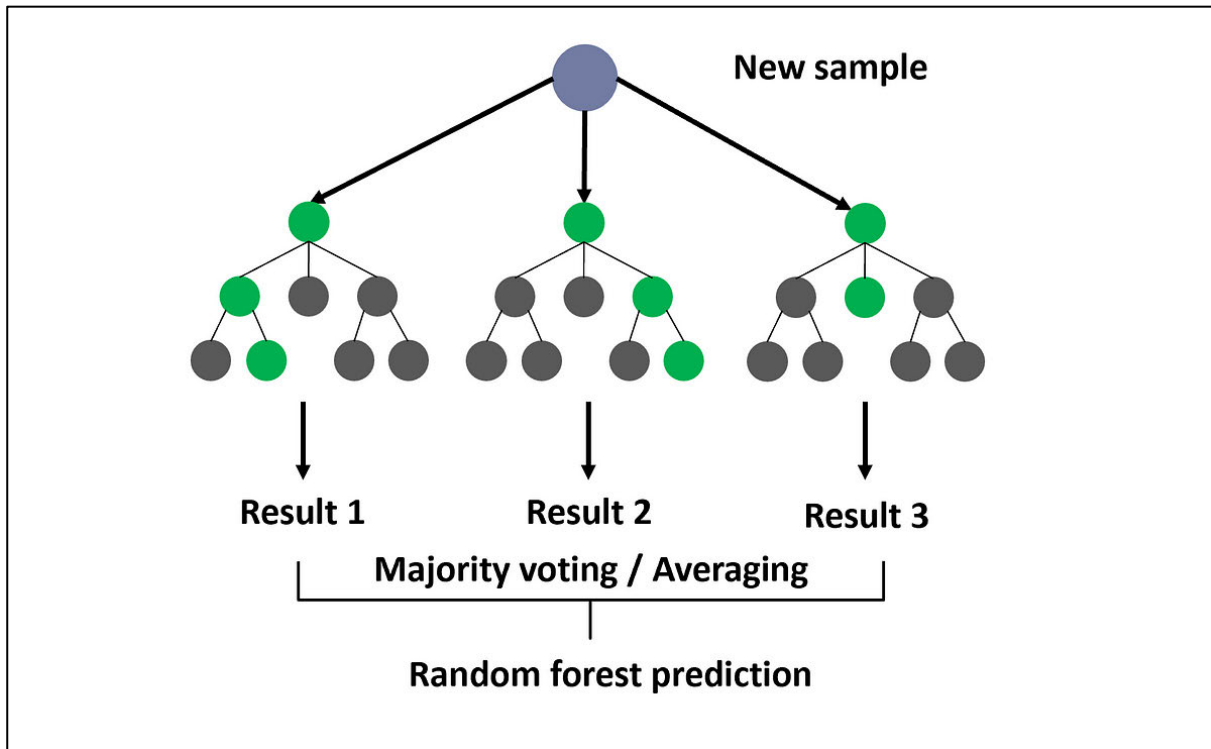


Figura 29: Modelo Bosques Aleatorios

Para poder hacer uso de este modelo se importó de *sklearn.ensemble* la librería *RandomForestRegressor*.

La Figura 30 muestra como se ha empleado la librería *RandomForestRegressor*:

```
# Modelo de Bosques Aleatorios
rf_model = RandomForestRegressor(n_estimators=100, max_depth=None, random_state=42)
rf_model.fit(X_train_scaled, y_train)
```

Figura 30: Código del Modelo Bosques Aleatorios

Donde $n_estimators=100$ indica que se han empleado un bosque aleatorio formado por 100 árboles de decisión, $max_depth=None$ indica que no se establece profundidad máxima para los árboles permitiendo que crezcan hasta su máxima profundidad.

- **Red Neuronal Artificial (ANN, Artificial Neuronal Network)**

Una Red Neuronal Artificial es un modelo computacional inspirado en la estructura y funcionamiento del cerebro humano. Consiste en una red de nodos interconectados llamados neuronas artificiales, que procesan información de entrada y generan una salida [66].

Están formadas por capas de neuronas artificiales interconectadas [67]:

- Capa de entrada: Recibe los datos.
- Capas ocultas: Procesan la información.

- Capa de salida: Produce el resultado final.

Cada neurona recibe una o más entradas, las procesa empleando la función de activación y transmite la salida a otras neuronas. La Figura 31 muestra el funcionamiento de la Red Neuronal Artificial (ANN) [68]:

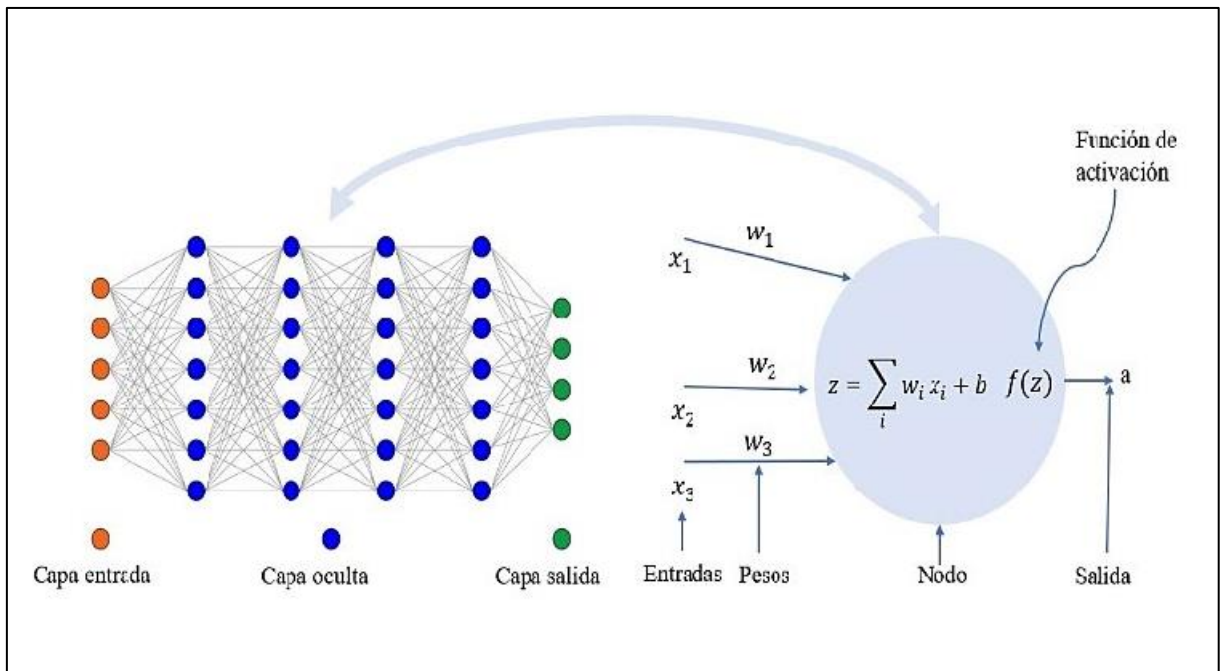


Figura 31: Modelo ANN

Para poder emplear este modelo se importó de *tensorflow.keras.models* la librería *Sequential* y de *tensorflow.keras.layers* las librerías *Dense* y *Dropout*. La Figura 32 muestra cómo se han empleado estas librerías:

```
# Construcción del modelo ANN
model = Sequential([
    Dense(128, input_dim=X_train_scaled.shape[1], activation='relu'), # Capa oculta 1
    Dropout(0.3), # Dropout para evitar sobreajuste
    Dense(64, activation='relu'), # Capa oculta 2
    Dropout(0.3), # Dropout adicional
    Dense(32, activation='relu'), # Capa oculta 3
    Dense(1, activation='linear') # Capa de salida (regresión)
```

Figura 32: Código del Modelo ANN

Donde *Sequential()* crea un modelo secuencial donde las capas se apilan una sobre otra, *Dense(128, input_dim=X_train_scaled.shape, activation='relu')* emplea 128 neuronas y *relu* es la función de activación, *Dropout(0.3)* desactiva el 30% de las neuronas durante la fase de entrenamiento para prevenir el sobreajuste y de esta manera prevenir que memorice los datos, *Dense(1, activation='linear')* se crea una neurona para la salida y se emplea una activación lineal para mostrar los resultados.

5. Resultados

En este capítulo se describen cuáles han sido los resultados obtenidos con los distintos modelos descritos en los capítulos anteriores y se realiza una comparación entre ellos para estudiar sus diferencias. Además, se describe cual es el modelo que mejor resultado ha obtenido prediciendo la duración de la sesión. Se muestran los modelos ordenados por nivel de complejidad, siendo el primer modelo el más sencillo y el último el más complejo

Para la elaboración de los resultados se ha empleado el 80% del dataset para el entrenamiento de los modelos y el 20% para su testeo.

- **Regresión Lineal Múltiple**

Este modelo emplea la regresión lineal teniendo en cuenta varias variables. Los resultados obtenidos por el modelo han sido los siguientes:

- Error Cuadrático Medio (MSE): 20852
- Error Absoluto Medio (MAE): 92
- Coeficiente de Determinación (R^2): 0.43

Según estos datos, puede analizarse como tiene un rendimiento de casi un 43% por lo que no es el mejor modelo para predecir el tiempo de una sesión de comunicación. La Figura 33 muestra una comparación entre los datos reales y los datos predichos por el modelo:

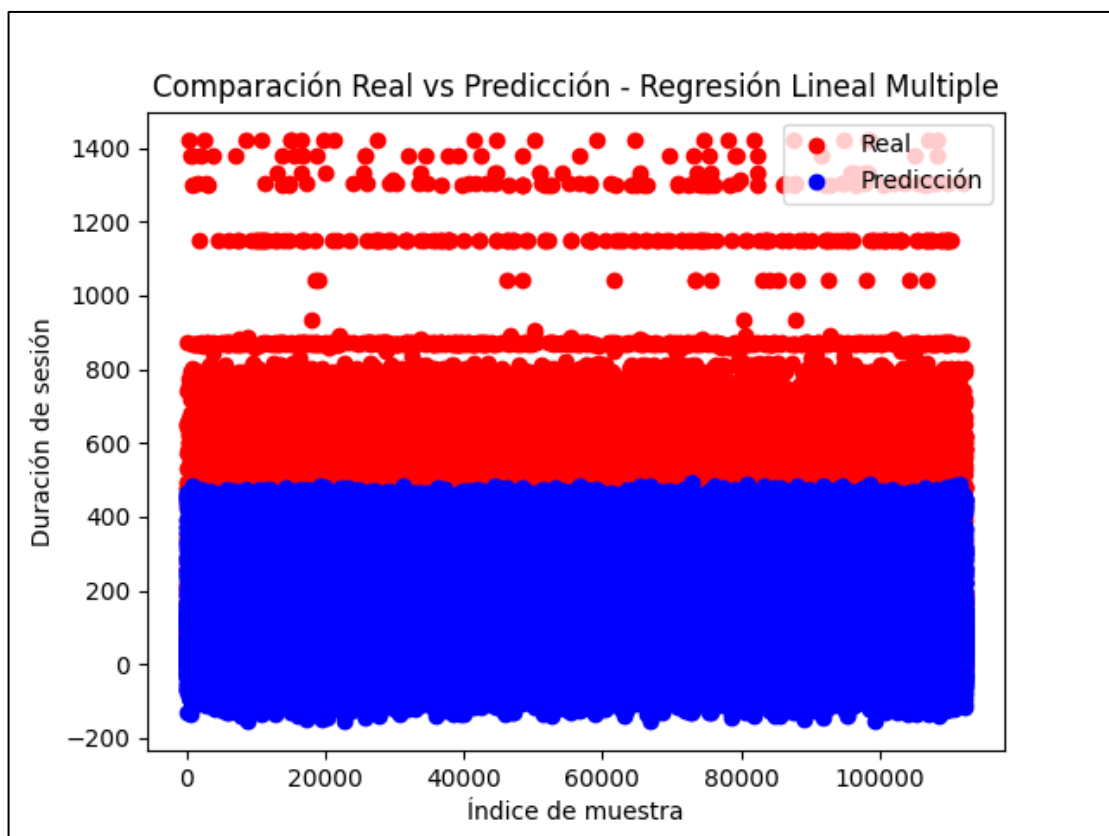


Figura 33: Comparación Regresión Lineal Múltiple

Analizando los resultados se puede apreciar que este modelo no es el más adecuado para predecir el tiempo de duración de la sesión por su bajo rendimiento y los errores obtenidos, como se puede apreciar en la Figura 33. Esta figura representa la comparación entre los valores reales y los predichos por el modelo de la duración de la sesión, donde los puntos rojos representan los valores reales y los puntos azules muestran las predicciones del modelo. El eje x (índice de muestra) indica que cada punto en la gráfica corresponde con una observación del conjunto de prueba. Mientras que el eje y (Duración de sesión) representa la variable objetivo que estamos prediciendo.

- **Máquinas de Soporte Vectorial (SVR)**

El modelo que emplea Máquinas de Soporte Vectorial emplea las mismas variables que el resto de modelos. Los resultados obtenidos han sido los siguientes:

- Error Cuadrático Medio (MSE): 19387
- Error Absoluto Medio (MAE): 66
- Coeficiente de Determinación (R^2): 0.47

Con estos datos se puede apreciar una leve mejoría respecto al anterior modelo, con un rendimiento de casi un 47%. Teniendo en cuenta este rendimiento, se puede afirmar que tampoco es el mejor modelo para predecir el tiempo de la sesión de comunicación.

La Figura 34 muestra la comparación entre los datos reales y los predichos por el modelo:

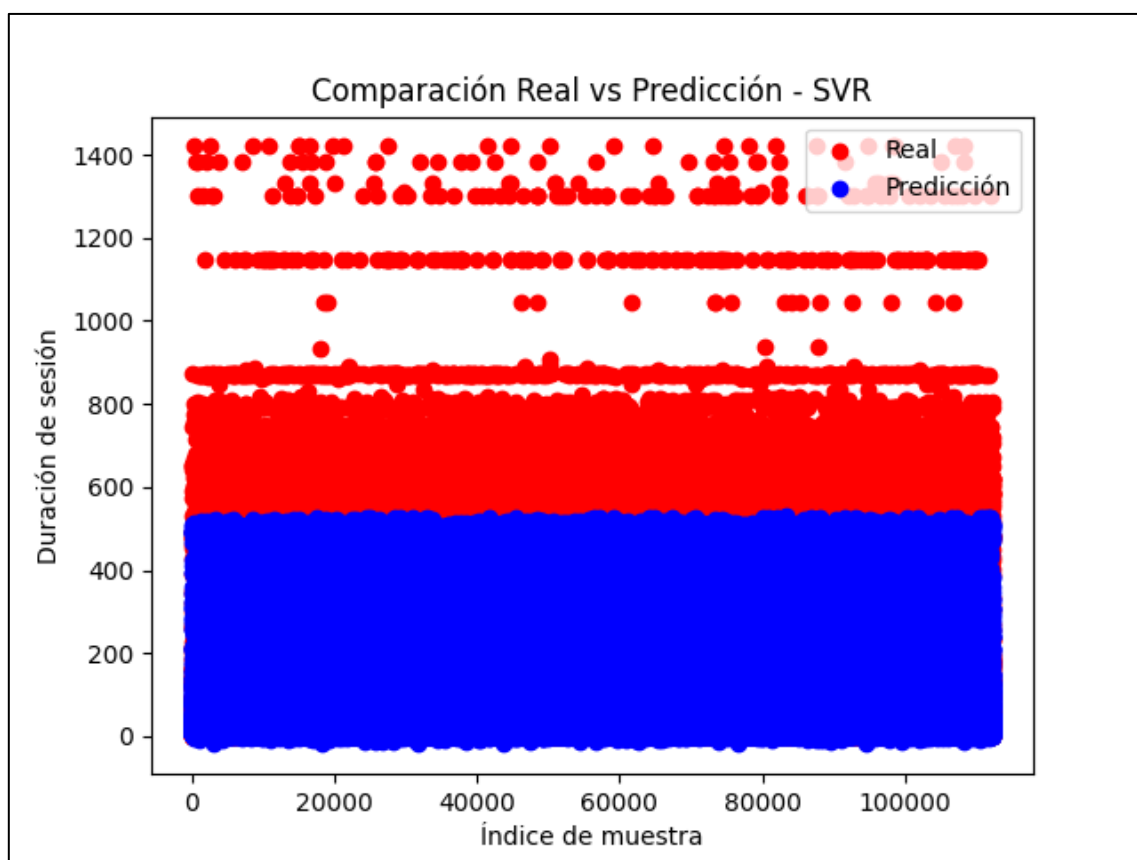


Figura 34: Comparación SVR

Analizando los resultados se puede apreciar que este modelo no es el más adecuado para predecir el tiempo de duración de la sesión por su bajo rendimiento y los errores obtenidos, como se puede apreciar en la Figura 34.

- **Árboles de Decisión (*Decision Trees*)**

Este modelo emplea un Árbol de Decisión para establecer las decisiones y mostrar los resultados. Los resultados obtenidos son los siguientes:

- Error Cuadrático Medio (MSE): 12969
- Error Absoluto Medio (MAE): 38
- Coeficiente de Determinación (R^2): 0.65

Los datos obtenidos como resultado del uso de este modelo muestran una mejoría en la predicción con un rendimiento de alrededor del 65%.

Estos resultados pueden apreciarse en la Figura 35 comparando los datos reales del *dataset* con los predichos por este modelo:

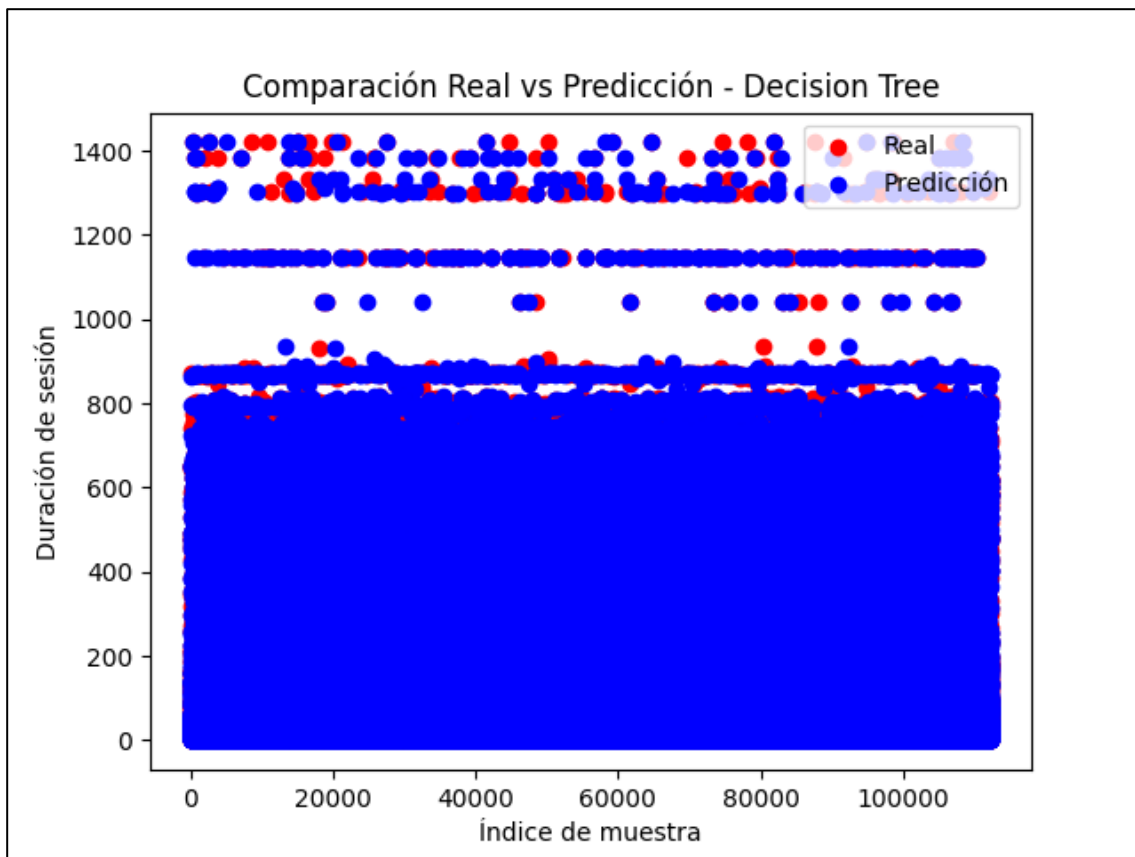


Figura 35: Comparación Árboles de Decisión

Analizando los resultados, se puede apreciar como este modelo mejora las predicciones del tiempo de duración de la sesión. Este modelo tiene un mayor rendimiento que los dos anteriores como se puede apreciar en la Figura 35.

- **Bosques Aleatorios (*Random Forest*)**

El modelo de Bosques Aleatorios emplea una estructura basada en el empleo de Árboles de Decisión para mostrar los resultados. Los resultados obtenidos han sido los siguientes:

- Error Cuadrático Medio (*MSE*): 7841
- Error Absoluto Medio (*MAE*): 34
- Coeficiente de Determinación (R^2): 0.79

Los resultados obtenidos al emplear este modelo muestran una mejoría respecto a los tres modelos anteriores, con un rendimiento del 79%.

Los resultados obtenidos se pueden apreciar en la Figura 36 donde se comparan los datos reales del *dataset* con los datos predichos por este modelo:

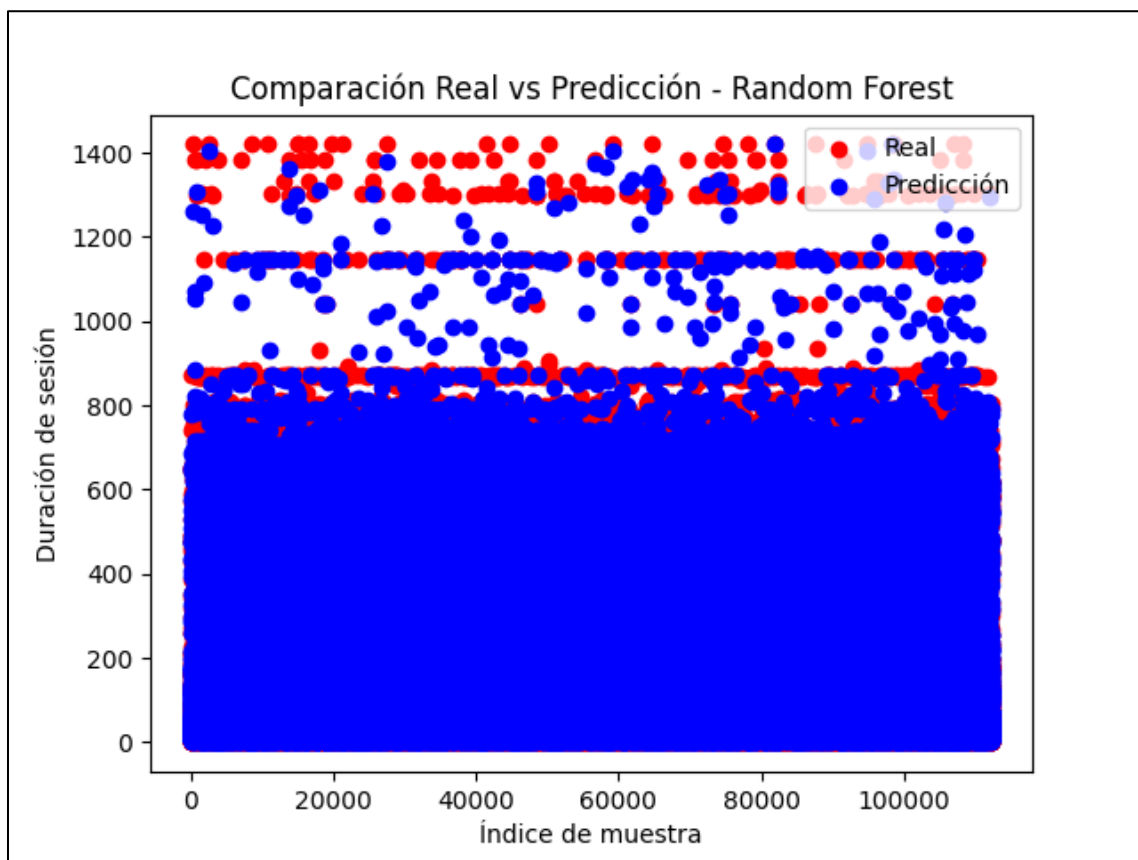


Figura 36: Comparación Bosques Aleatorios

Analizando los resultados, se puede apreciar como este modelo mejora las predicciones respecto de los modelos anteriores. Este modelo tiene un mayor rendimiento como se puede apreciar en la Figura 36.

- **Red Neuronal Artificial (*ANN*)**

La Red Neuronal Artificial es un modelo de aprendizaje automático o *ML*, que emplea una red con varias capas que imita una red neuronal para predecir el tiempo de la sesión de comunicación. Los resultados obtenidos con este modelo son los siguientes:

- Error Cuadrático Medio (MSE): 4173
- Error Absoluto Medio (MAE): 31
- Coeficiente de Determinación (R^2): 0.89

Los resultados obtenidos tras el empleo de este modelo muestran un rendimiento de casi un 90%, siendo con gran diferencia, el modelo que mejor rendimiento proporciona.

Estos resultados pueden apreciarse en la Figura 37 donde se comparan los datos reales del *dataset* con los predichos por el modelo:

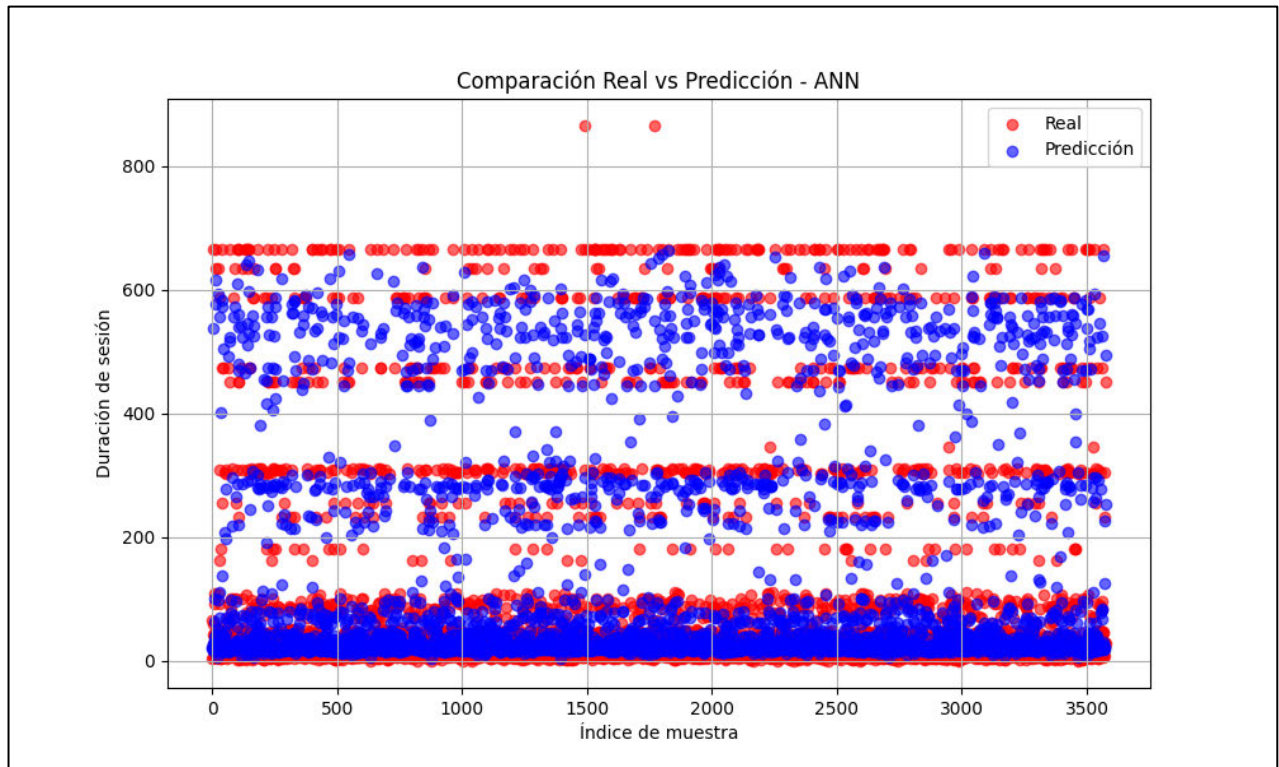


Figura 37: Comparación ANN

Analizando los resultados gráficos puede apreciarse como este modelo mejora significativamente las predicciones respecto a todos los modelos anteriores. Esta mejora en la predicción denota que este modelo es el que da el mayor rendimiento como se aprecia en la Figura 37.

En la Tabla 6 se realiza una comparación entre los modelos mostrando los resultados aproximados obtenidos de cada uno de MSE , MAE y R^2 .

Los resultados están ordenados por complejidad de los modelos, siendo el más sencillo la Regresión Lineal Múltiple y el más complejo la Red Neuronal Artificial (ANN). Se puede comprobar como a medida que aumenta la complejidad del modelo, disminuyen los errores y, por ende, mejora el rendimiento de los modelos.

Tabla 6: Comparación de resultados de los modelos

Modelo	MSE	MAE	R^2
Regresión Lineal Múltiple	20852	92	0.43
Máquinas de Soporte Vectorial (SVR)	19387	66	0.47
Árbol de Decisión	12969	38	0.65
Bosque Aleatorio	7841	34	0.79
Red Neuronal Artificial (ANN)	4173	31	0.89

Los valores de MSE y MAE corresponden con los errores que producen los modelos al predecir los resultados sobre los valores reales que se encuentran en el dataset. Los valores que muestran ambas métricas hacen referencia a la normalización que se tuvo que hacer en los datos de entrada para que estos pudieran ser tratados adecuadamente.

Para concluir con los resultados, se puede afirmar que el modelo que mejor rendimiento ofrece a este proyecto, a la hora de predecir el tiempo de las sesiones de comunicación, es la Red Neuronal Artificial (ANN). Este modelo que emplea el aprendizaje automático ofrece un coeficiente de determinación o rendimiento de 0.89 (89%). Siendo, por tanto, el modelo que menos errores introduce en los resultados de las predicciones y, por consiguiente, el modelo que mejor se adapta a las necesidades de este proyecto.

6. Impacto del proyecto

En este apartado se detalla el coste aproximado para la realización de este proyecto. La Tabla 7 muestra los costes desglosados según el nombre, las unidades empleadas, el precio por cada unidad (precio unitario) y el precio total. Los conceptos empleados son los siguientes:

- **Material:** el ordenador se considera como pieza fundamental para poder llevar a cabo el desarrollo del proyecto puesto que todas las herramientas empleadas son virtuales. Se ha presupuestado un ordenador con un coste de 1.000,00€. Teniendo en cuenta una amortización en 4 años y que el proyecto ha tenido una duración de 6 meses, se establece un coste de 125,00€ para la elaboración del proyecto.
- **Software:** este apartado describe todas las herramientas empleadas durante el desarrollo del proyecto.
 - *OpenStreetMap*: herramienta empleada para conseguir los mapas para las simulaciones con un coste de 0,00 €, al tratarse de un *software* de código libre.
 - *Simulador SUMO*: herramienta empleada para las simulaciones de los vehículos con un coste de 0,00 € por tratarse de una aplicación de código libre.
 - *Visual Studio Code*: herramienta empleada para el desarrollo y visualización del código y los resultados, tiene un coste de 0,00 € por tratarse de una aplicación de código libre.
- **Mano de obra:** se ha considerado como coste el sueldo medio de un Ingeniero de Telecomunicaciones Junior de 40 €/hora. Por lo tanto, el proyecto al tener una duración de 320 horas se obtiene un coste en salario de 12.800,00 €.

Finalmente, sumando el gasto de todos los recursos necesarios para la realización del proyecto, obtenemos un gasto total de 12.925,00€.

Tabla 7: Presupuesto del proyecto

Concepto	Descripción	Unidades	Precio unitario	Precio Total
Material	Ordenador	1	1.000,00 €	125,00 €
Software	<i>OpenStreetMap</i>	1	- €	- €
	<i>Simulador SUMO</i>	1	- €	- €
	<i>Visual Studio Code</i>	1	- €	- €
Mano de obra	Hora de trabajo como Ingeniero	320	40,00 €	12.800,00 €
TOTAL				12.925,00 €

A continuación, se redactan cuáles son las principales implicaciones que están relacionadas con este proyecto. Muestra el impacto social, de salud y seguridad, ambiental, económico, tecnológico e industrial y una posible contribución a los *ODS* (Objetivos de Desarrollo Sostenible). Las mencionadas implicaciones se desarrollan a continuación:

- **Impacto social**

El sistema diseñado tiene un impacto significativo en la seguridad vial y la reducción de accidentes. Este proyecto contribuye directamente al *ODS 11*: Ciudades y Comunidades Sostenibles, ya que fomenta una movilidad segura y accesible mediante el intercambio de información en tiempo real entre vehículos e infraestructura vial [69]. Reducir los accidentes no solo salva vidas, sino que también disminuye costos asociados a emergencias médicas y daños materiales, mejorando la calidad de vida de las comunidades afectadas. Además, refuerza la confianza en el uso de tecnologías avanzadas para el transporte.

- **Impacto en salud y seguridad**

Al disminuir los accidentes y la congestión de vehículos, el proyecto el proyecto contribuye indirectamente a mejorar la calidad del aire al reducir las emisiones de gases contaminantes producidos por vehículos detenidos en el tráfico. Esto se alinea con el *ODS 3*: salud y bienestar, ya que reduce los riesgos asociados a la contaminación y mejora la salud de las personas expuestas al ruido y la polución [70].

Asimismo, el uso de sistemas confiables de comunicación vehicular como *DSRC* garantiza la transmisión de información crítica sobre tráfico y emergencias con baja latencia, mejorando significativamente los tiempos de respuesta ante incidentes [71].

- **Impacto ambiental**

Este proyecto incorpora un enfoque sostenible al optimizar el flujo vehicular, reduciendo las emisiones de dióxido de carbono (CO_2) y otros gases contaminantes. Así, contribuye al *ODS 13*: Acción por el Clima, implementando medidas que mitigan el impacto del transporte en el cambio climático [72]. Además, el diseño del sistema promueve un uso más eficiente de los recursos energéticos en vehículos eléctricos y autónomos, disminuyendo su huella ambiental.

- **Impacto económico**

Desde una perspectiva económica, este sistema ofrece beneficios significativos al reducir los costos relacionados con accidentes y congestión. Además, abre oportunidades para nuevos modelos de negocio, como seguros vehiculares basados en datos en tiempo real o la integración de servicios adicionales en carreteras inteligentes [73].

La implementación de tecnologías avanzadas como *DSRC* en infraestructuras viales también genera empleo en sectores tecnológicos e industriales, impulsando la economía local y fomentando la inversión en investigación y desarrollo

- **Impacto tecnológico e industrial**

El proyecto impulsa la innovación tecnológica al desarrollar soluciones avanzadas de conexiones entre vehículos. Esto fomenta el crecimiento del sector de comunicaciones y contribuye al desarrollo de ciudades inteligentes, un aspecto clave en el *ODS 9: Industria, Innovación e Infraestructura* [74].

El uso de estándares como IEEE 802.11p garantiza la interoperabilidad y escalabilidad del sistema, facilitando su adopción global y fomentando la creación de nuevas tecnologías relacionadas con la movilidad autónoma.

- **Contribución a los ODS**

En resumen, este proyecto se alinea con varios ODS, entre ellos:

- ODS 3: Reducir accidentes y mejorar la salud mediante un tráfico más seguro.
- ODS 9: Promover infraestructuras resilientes y sostenibles.
- ODS 11: Incrementar la seguridad en la movilidad urbana.
- ODS 13: Mitigar el cambio climático a través de la reducción de emisiones.

7. Conclusiones

En el último apartado se van a exponer las conclusiones del proyecto, así como los trabajos futuros que se podrían llevar a cabo con el fin de mejorar la solución propuesta y proseguir a partir de las soluciones de este proyecto.

7.1 Conclusiones

En este proyecto de fin de carrera, se ha desarrollado un sistema basado en comunicaciones *DSRC* para predecir la duración de las sesiones de comunicación entre vehículos en entornos urbanos, suburbanos y autopistas con alta congestión. Para ello, se analizó la problemática de mantener una conexión estable en situaciones de alta densidad de vehículos en el entorno, empleando aspectos de la capa física de las comunicaciones como la relación señal a ruido (*SNR*) y las condiciones ambientales.

Para resolver esta problemática, se implementaron modelos de *machine learning* y *deep learning* que permitieron estimar la duración de las sesiones de comunicación. Se estableció un umbral óptimo de *SNR* de 15 dB con un margen de enlace de 5 dB, lo que garantizó la estabilidad de la conexión y evitó interrupciones causadas por desvanecimientos rápidos de la señal.

Los resultados obtenidos muestran que modelos avanzados como Bosques Aleatorios (*Random Forest*) y Redes Neuronales Artificiales (*ANN*, *Artificial Neuronal Network*) ofrecen predicciones más precisas en comparación con enfoques más simples como regresión lineal múltiple o *SVM* (*Support Vector Machine* o Maquinas de Soporte Vectorial). Además, se comprobó que la inclusión de un margen de enlace mejora notablemente la estabilidad de las predicciones en escenarios dinámicos y con alto nivel de ruido.

Desde una perspectiva práctica, el sistema desarrollado contribuye a la seguridad vial al optimizar la transmisión de datos en tiempo real, permitiendo una mejor coordinación entre vehículos e infraestructura empleando el protocolo de transmisión de datos *UDP*. Asimismo, el análisis de impacto ambiental confirma que la optimización del flujo vehicular puede reducir significativamente las emisiones de CO_2 y mejorar la calidad del aire en áreas urbanas.

7.2 Trabajos Futuros

El desarrollo de este proyecto ha permitido identificar varias áreas de mejora que pueden incrementar la precisión y utilidad del sistema en futuras investigaciones. Estas mejoras abarcan desde la optimización de los modelos de predicción hasta la exploración de nuevas tecnologías y la validación con datos reales. A continuación, se detallan las principales líneas de trabajo a considerar en futuras fases de desarrollo.

Una de las áreas clave es la mejora y optimización de los modelos de predicción, un proceso que puede evolucionar a medida que avanza la tecnología y se descubren nuevas formas de optimizar los recursos. En este proyecto, se inició con modelos básicos de *machine learning* y

se fueron incrementando en complejidad hasta llegar a un modelo más avanzado de *ML* basado en Redes Neuronales Artificiales (*ANN*). Sin embargo, aún existe un gran potencial para emplear modelos más avanzados de *deep learning*, como redes neuronales recurrentes (*RNN*, *Recurrent Neural Network*) u otras arquitecturas más sofisticadas, que permitan capturar con mayor precisión la variabilidad temporal en la conectividad vehicular.

Además, el uso de técnicas de optimización de hiper-parámetros puede mejorar significativamente el rendimiento de los modelos actuales, incrementando su precisión y capacidad de adaptación a diferentes escenarios. Para hacer que los resultados sean más realistas y representativos de condiciones reales, se recomienda complementar estos modelos con datos obtenidos de evaluaciones en escenarios reales urbanos. Esto permitiría validar las predicciones con información empírica y ajustar los modelos para que reflejen de manera más precisa el comportamiento de la conectividad vehicular en situaciones reales.

Otra línea de trabajo prometedora es la exploración de *5G-V2X*, una tecnología que ofrece ventajas en términos de latencia, estabilidad y eficiencia energética en comparación con *DSRC*. Evaluar el rendimiento del modelo en redes *5G-V2X* permitiría determinar su viabilidad en escenarios de comunicación vehicular más avanzados. Asimismo, diseñar estrategias que combinen *DSRC* y *5G* podría mejorar la robustez y la adaptación del sistema en distintos entornos, garantizando una conectividad más estable, segura y eficiente.

Por último, se plantea el desarrollo de un sistema de predicción en tiempo real. Esto implicaría la implementación de la solución en dispositivos físicos integrados en los vehículos o en plataformas en la nube para proporcionar estimaciones en tiempo real sobre la duración de las sesiones de comunicación. Además, analizar la integración con vehículos autónomos y sistemas avanzados de asistencia a la conducción (*ADAS*) permitiría optimizar la toma de decisiones en situaciones críticas, mejorando la seguridad y eficiencia del tráfico vehicular.

En conclusión, este proyecto sienta las bases para futuros avances en el ámbito de la conectividad vehicular y su aplicación en entornos de sistemas inteligentes de transporte (*ITS* por sus siglas en inglés). La mejora en la precisión de los modelos, la validación con datos reales y la integración con nuevas tecnologías contribuirán al desarrollo de sistemas de transporte más eficientes y seguros, impulsando la movilidad urbana hacia un futuro más sostenible e innovador.

Referencias

- [1] Dirección General de Tráfico (DGT), "Sistemas Inteligentes de Transporte (ITS)," [En línea]. Disponible en: <https://www.dgt.es/muevete-con-seguridad/tecnologia-e-innovacion-en-carretera/sistemas-inteligentes-de-transporte-its/>.
- [2] JDM Group Company, "¿Qué es un sistema de transporte inteligente?," *JDM Group Company Blog*, May 5, 2021. [Online]. Available: <https://jdmgroupcompany.com/que-es-un-sistema-de-transporte-inteligente/?lang=es>.
- [3] B. Ribeiro, M. J. Nicolau, y A. Santos, "Using Machine Learning on V2X Communications Data for VRU Collision Prediction," *Sensors*, vol. 23, no. 3, p. 1260, 2023. [En línea]. Disponible en: <https://doi.org/10.3390/s23031260>.
- [4] Keysight Technologies, "V2X Post," *Keysight Blog*, 3 de octubre de 2024. [En línea]. Disponible en: <https://www.keysight.com/blogs/en/inds/auto/2024/10/03/v2x-post>.
- [5] COIT, "5G y los sentidos del vehículo autónomo," *BIT Revista del Colegio Oficial de Ingenieros de Telecomunicación*, 2024. [Online]. Available: <https://bit.coit.es/5g-y-los-sentidos-del-vehiculo-autonomo/>.
- [6] Tomorrow Bio, "Cuando los coches hablan: cómo la comunicación V2V está dando forma al futuro del transporte," *Tomorrow Bio*, 20 de julio de 2023. [En línea]. Disponible en: <https://tomorrow.bio/es/post/cuando-los-coches-hablan-como-la-comunicacion-v2v-esta-dando-forma-al-futuro-del-transporte>.
- [7] CORDIS, "La conducción en pelotón da un paso más hacia su implementación," *CORDIS - Comisión Europea*, 2024. [Online]. Available: <https://cordis.europa.eu/article/id/254161-platoon-driving-inches-a-step-closer/es>.
- [8] H. Rodríguez, "El futuro de los coches autónomos y conectados," *National Geographic España*, Jan. 3, 2023. [Online]. Available: https://nationalgeographic.com.es/ciencia/actualidad/futuro-coches-autonomos-y-conectados_13619.
- [9] Data Alliance, "C-V2X y DSRC: Coche conectado - Tecnologías de vehículos autónomos," [En línea]. Disponible en: <https://www.data-alliance.net/es/c-v2x-y-dsrc-coche-conectado-tecnolog%C3%ADas-de-veh%C3%ADculos-aut%C3%B3nomos>.
- [10] D. Tian, H. Luo, J. Zhou, Y. Wang, G. Yu and HaiyingXia, "A Self-Adaptive V2V Communication System with DSRC," *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*, Beijing, China, 2013, pp. 1528-1532, doi: 10.1109/GreenCom-iThings-CPSCoM.2013.271.

- [11] A. Demba and D. P. F. Möller, "Vehicle-to-Vehicle Communication Technology," *2018 IEEE International Conference on Electro/Information Technology (EIT)*, Rochester, MI, USA, 2018, pp. 0459-0464, doi: 10.1109/EIT.2018.8500189.
- [12] J. B. Reddy, L. Deneire, L. Van der Perre, B. Gyselinckx and M. Engels, "On equalization for OFDM-dedicated short range communication (DSRC) modem," *2000 IEEE International Conference on Personal Wireless Communications. Conference Proceedings (Cat. No.00TH8488)*, Hyderabad, India, 2000, pp. 230-234, doi: 10.1109/ICPWC.2000.905809.
- [13] U.S. Department of Transportation, "Connected Vehicle Applications," *National Operations Center of Excellence*, 2024. [Online]. Available: <https://transportationops.org/connected-fleet-challenge/cvapplications>.
- [14] Ciberseguridad Euskadi, "Infraestructura de Clave Pública (PKI)," *Ciberglosario*, 2024. [Online]. Available: <https://www.ciberseguridad.eus/ciberglosario/infraestructura-de-clave-publica-pki>.
- [15] J. KIEES, "Article from JKIEES Archive," *Journal of Korean Institute of Electromagnetic Engineering and Science*, vol. 32, no. 2, pp. 110, 2024. [Online]. Available: https://www.jkiees.org/archive/view_article?pid=jkiees-32-2-110.
- [16] X. Wang, R. A. Berry, I. Vukovic and J. Rao, "A Fixed-Point Model for Semi-Persistent Scheduling of Vehicular Safety Messages," *2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)*, Chicago, IL, USA, 2018, pp. 1-5, doi: 10.1109/VTCFall.2018.8690990.
- [17] MD Elektronik, "V2X: Comunicación Vehículo a Todo," *MD Elektronik Website*, 2024. [Online]. Available: <https://www.md-elektronik.com/es/v2x/>.
- [18] M. Gonzalez-Martín, M. Sepulcre, R. Molina-Masegosa and J. Gozalvez, "Analytical Models of the Performance of C-V2X Mode 4 Vehicular Communications," in *IEEE Transactions on Vehicular Technology*, vol. 68, no. 2, pp. 1155-1166, Feb. 2019, doi: 10.1109/TVT.2018.2888704.
- [19] Keysight Technologies, "V2X Communication: Ensuring Safety in a Connected World," *Keysight Blog*, Oct. 3, 2024. [Online]. Available: <https://www.keysight.com/blogs/en/inds/auto/2024/10/03/v2x-post>.
- [20] Kim, Wooseong. "Experimental Demonstration of MmWave Vehicle-to-Vehicle Communications Using IEEE 802.11ad." *Sensors (Basel, Switzerland)* vol. 19,9 2057. 2 May. 2019, doi:10.3390/s19092057.
- [21] Hualian Xingtong, "What is Millimeter Wave?," *Hualian Xingtong Website*, 2024. [Online]. Available: <https://www.hualianxingtong.com/es/what-is-millimeter-wave>.

- [22] J. A. Ruíz-Vega, J. F. Gutiérrez-Gutiérrez, and M. O. López-Rodríguez, "Modelos de propagación para sistemas de comunicación inalámbrica," *Revista Mexicana de Ingeniería Biomédica*, vol. 38, no. 2, pp. 83-100, 2017. [Online]. Available: https://www.scielo.org.mx/scielo.php?script=sci_arttext&pid=S2007-07052017000200083.
- [23] ITU-R, *Propagation data and prediction methods for the planning of short-range outdoor radiocommunication systems and radio local area networks in the frequency range 300 MHz to 100 GHz*, Recommendation ITU-R P.1411-12, Aug. 2023. [Online]. Available: https://www.itu.int/dms_pubrec/itu-r/rec/p/R-REC-P.1411-12-202308-!!PDF-S.pdf.
- [24] H. Fernandez, L. Rubio, V. M. Rodrigo-Penarrocha y J. Reig, "Path loss characterization for vehicular communications at 700 MHz and 5.9 GHz under LOS and NLOS conditions," *IEEE Antennas and Wireless Propagation Letters*, vol. 13, pp. 931–934, 2014.
- [25] S. S. Shokry, O. Abdelfattah, y G. Mabrouk, "Virtual MIMO for vehicular communication systems," *Journal of Al-Azhar University Engineering Sector*, vol. 17, no. 65, pp. 1348–1361, Oct. 2022. [Online]. Available: https://jaes.journals.ekb.eg/article_266220_0404bec991055be03ed6bfca43dfe1db.pdf.
- [26] H. A. Fricker, Lecture 7: Propagation, dispersion and scattering, 2009 [En línea]. Disponible en: <https://topex.ucsd.edu/rs/Lec07.pdf>.
- [27] H. Iwai, R. Yoshida and H. Sasaoka, "Propagation models for simulation scenario of ITS V2V communications," *2013 Proceedings of the International Symposium on Antennas & Propagation*, Nanjing, China, 2013, pp. 689-692.
- [28] C. Huygens, *Treatise on Light* (Illustrated Edition). Echo Library, 2007.
- [29] F. Jameel, Faisal, M. A. A. Haider and A. A. Butt, "Performance analysis of VANETs under Rayleigh, Rician, Nakagami-m and Weibull fading," *2017 International Conference on Communication, Computing and Digital Systems (C-CODE)*, Islamabad, Pakistan, 2017, pp. 127-132, doi: 10.1109/C-CODE.2017.7918915.
- [30] A. Wage, *Tema 4: Modelos de Propagación Electromagnética*, 2024. [Online]. Available: https://awage.github.io/download/mpe_tema4.pdf.
- [31] E. Basar, *Wireless Communications with Reconfigurable Intelligent Surfaces*, Koç University, Turkey, Sep. 2020. [Online]. Available: <https://corelab.ku.edu.tr/>
- [32] B. Kihei, J. A. Copeland and Y. Chang, "Improved 5.9GHz V2V Short Range Path Loss Model," *2015 IEEE 12th International Conference on Mobile Ad Hoc and Sensor Systems*, Dallas, TX, USA, 2015, pp. 244-252, doi: 10.1109/MASS.2015.84.

- [33] De La Paz Vizqueira, Luis. (2023). Formulas de Friss. 10.13140/RG.2.2.22857.21603.
- [34] O. Onubogu, K. Ziri-Castro, D. Jayalath, K. Ansari and H. Suzuki, "Empirical vehicle-to-vehicle pathloss modeling in highway, suburban and urban environments at 5.8 GHz," *2014 8th International Conference on Signal Processing and Communication Systems (ICSPCS)*, Gold Coast, QLD, Australia, 2014, pp. 1-6, doi: 10.1109/ICSPCS.2014.7021126.
- [35] S. K. Choudhary y D. K. D. C. A. CITE, "Path loss prediction models for wireless communication channels and its comparative analysis," en *International Journal of Engineering, Management & Sciences (IJEMS)*, 2015.
- [36] S. K. Choudhary y D. K. D. C. A. CITE, "Path loss prediction models for wireless communication channels and its comparative analysis," en *International Journal of Engineering, Management & Sciences (IJEMS)*, 2015.
- [37] T. Abbas, K. Sjöberg, J. Karedal y F. Tufvesson, "A measurement based shadow fading model for vehicle-to-vehicle network simulations," *International Journal of Antennas and Propagation*, vol. 2015, pp. 1–12, 2015.
- [38] Y. Sahraoui, A. Ghanam, S. Zaidi, S. Bitam and A. Mellouk, "Performance evaluation of TCP and UDP based video streaming in vehicular ad-hoc networks," *2018 International Conference on Smart Communications in Network Technologies (SaCoNeT)*, El Oued, Algeria, 2018, pp. 67-72, doi: 10.1109/SaCoNeT.2018.8585447.
- [39] E. Collado, "Control de congestión en TCP," *Eduardo Collado Blog*, Dec. 6, 2023. [Online]. Available: <https://www.eduardocollado.com/2023/12/06/control-de-congestion-en-tcp/>.
- [40] J. Canfield, "Wireshark - 3 way handshake (SYN ACK)," *FreeKB*, Oct. 28, 2023. [Online]. Available: <https://freekb.net/Article?id=300>.
- [41] M. Padua, "UDP: Conozca la revolución silenciosa en la transmisión de datos," *IT Masters Mag*, Nov. 27, 2024. [Online]. Available: <https://itmastersmag.com/transformacion-digital/que-es-el-udp-y-para-que-sirve/>.
- [42] D. Dey, "TCP vs UDP: Exploring the Differences in Network Communication," *Medium*, 2024. [Online]. Available: <https://medium.com/@debopamdeycse19/tcp-vs-udp-exploring-the-differences-in-network-communication-a1543f2e70f3>.
- [43] I. Vienažindytė, "TCP vs. UDP, comparamos los dos protocolos," *NordVPN Blog*, Nov. 13, 2019. [Online]. Available: <https://nordvpn.com/es/blog/protocolo-tcp-udp/>.
- [44] OpenStreetMap, "Mapa interactivo de OpenStreetMap - Madrid, España," *OpenStreetMap*, 2024. [Online]. Available: <https://www.openstreetmap.org/#map=11/40.4407/-3.7202>.

-
- [45] Eclipse Foundation, "Simulation of Urban MObility (SUMO)," *Eclipse SUMO*, 2024. [Online]. Available: <https://eclipse.dev/sumo/>.
- [46] Deutsches Zentrum für Luft- und Raumfahrt (DLR), "TraCI (Traffic Control Interface) Documentation," *SUMO Documentation*, 2024. [Online]. Available: <https://sumo.dlr.de/docs/TraCI.html>.
- [47] SAP, "¿Qué es Machine Learning?," *SAP España*, 2024. [Online]. Available: <https://www.sap.com/spain/products/artificial-intelligence/what-is-machine-learning.html>.
- [48] DataCamp, "Casos de uso principales y algoritmos de Machine Learning," *DataCamp Blog*, 2024. [Online]. Available: <https://www.datacamp.com/es/blog/top-machine-learning-use-cases-and-algorithms>.
- [49] Universitat Carlemany, "Deep Learning: qué es, tipos y ejemplos," *Universitat Carlemany*, Nov. 13, 2023. [Online]. Available: <https://www.universitatcarlemany.com/actualidad/blog/deep-learning-que-es-tipos-ejemplos/>.
- [50] luiscano95, "tutorial.sumo - SUMO Traffic Simulation Tutorial," *GitHub*, 2016. [Online]. Available: <https://github.com/luiscano95/tutorial.sumo>.
- [51] addola, "YouTubeTutorials - SUMO Traffic Simulation Example," *GitHub*, 2019. [Online]. Available: <https://github.com/addola/YouTubeTutorials/blob/08665c306f49b7e60a9895be1f8693a810666ed8/SUMOExamples/IntersectionExample/Makefile>.
- [52] Deutsches Zentrum für Luft- und Raumfahrt (DLR), "Multi-Entry-Exit Detectors (E3) - SUMO Documentation," *SUMO Documentation*, 2024. [Online]. Available: https://sumo.dlr.de/docs/Simulation/Output/Multi-Entry-Exit_Detectors_%28E3%29.html.
- [53] Deutsches Zentrum für Luft- und Raumfahrt (DLR), "Induction Loops Detectors (E1) - SUMO Documentation," *SUMO Documentation*, 2024. [Online]. Available: https://sumo.dlr.de/docs/Simulation/Output/Induction_Loops_Detectors_%28E1%29.html.
- [54] Dive Into Python, "Abrir archivo XML y leer datos con Python," *Dive Into Python*, 2024. [Online]. Available: <https://diveintopython.org/es/learn/file-handling/xml#abrir-archivo-xml-y-leer-datos-con-python>.
- [55] Python Software Foundation, "xml.etree.ElementTree — The ElementTree XML API," *Python Documentation*, 2024. [Online]. Available: <https://docs.python.org/es/3.13/library/xml.etree.elementtree.html>.

- [56] DataCamp, "Tutorial de pandas en Python: la guía definitiva para principiantes," *DataCamp*, 2024. [Online]. Available: <https://www.datacamp.com/es/tutorial/pandas>.
- [57] Cadence, "What is Signal-to-Noise Ratio and How to Calculate It," *Cadence Blog*, 2020. [Online]. Available: <https://resources.pcb.cadence.com/blog/2020-what-is-signal-to-noise-ratio-and-how-to-calculate-it>.
- [58] DataCamp, "Modelos de Machine Learning Explicados," *DataCamp Blog*, 2024. [Online]. Available: <https://www.datacamp.com/es/blog/machine-learning-models-explained>.
- [59] IBM, "Regresión lineal múltiple," *IBM Documentation*, 2024. [Online]. Available: <https://www.ibm.com/docs/es/cognos-analytics/11.1.0?topic=tests-multiple-linear-regression>.
- [60] DATAtab, "Regresión Lineal - Explicación y Ejemplo," *DATAtab Tutorial*, 2024. [Online]. Available: <https://datatab.es/tutorial/linear-regression>.
- [61] IBM, "Máquinas de Soporte Vectorial (SVM)," *IBM Topics*, 2024. [Online]. Available: <https://www.ibm.com/es-es/topics/support-vector-machine>.
- [62] Spot Intelligence, "Support Vector Regression (SVR)," *Spot Intelligence*, May 8, 2024. [Online]. Available: <https://spotintelligence.com/2024/05/08/support-vector-regression-svr/>.
- [63] Scikit-learn, "Tree-based models," *Scikit-learn Documentation*, 2024. [Online]. Available: <https://scikit-learn.org/stable/modules/tree.html>.
- [64] Bookdown, "Árboles de decisión - Parte I," *Bookdown*, 2024. [Online]. Available: <https://bookdown.org/content/2031/arboles-de-decision-parte-i.html>.
- [65] R. Yehoshua, "Random Forests," *Medium*, Mar. 25, 2023. [Online]. Available: <https://medium.com/@roiyeo/random-forests-98892261dc49>.
- [66] GAMCO, "Red neuronal artificial (ANN)," *GAMCO Glosario*, 2024. [Online]. Available: <https://gamco.es/glosario/red-neuronal-artificial-ann/>.
- [67] IBM, "Neural network model," *IBM Documentation*, 2024. [Online]. Available: <https://www.ibm.com/docs/es/spss-modeler/saas?topic=networks-neural-model>.
- [68] Optimization of Rock Blasting Fragmentation in Open Pit Using Predictive Models - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/El-modelo-de-red-neuronal-artificial-ANN_fig1_379821656
- [69] Naciones Unidas, "Objetivos de Desarrollo Sostenible," *ONU*, 2024. [Online]. Available: <https://www.un.org/sustainabledevelopment/es/>.

-
- [70] C. Fernández Aller y R. Miñano, *Guía para trabajar la Responsabilidad Social y Ambiental (GRSA)*, Versión 2.0, ETSI Sistemas Informáticos, UPM, Mayo 2015.
- [71] IEEE, *IEEE Standard for Information Technology—Telecommunications and Information Exchange Between Systems—Local and Metropolitan Area Networks—Specific Requirements—Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications—Amendment 6: Wireless Access in Vehicular Environments*, IEEE Std 802.11p-2010, IEEE Standards Association, 2010.
- [72] D. Schrank, B. Eisele, y T. Lomax, *Urban Mobility Report 2021*, Texas A&M Transportation Institute, 2021.
- [73] R. Sivaraj y T. Ravichandran, “Impact of Intelligent Transportation Systems on Economic Growth,” *International Journal of Transport Economics*, vol. 49, no. 2, pp. 153–169, 2021.
- [74] A. Festag, “Standards for Vehicular Communication: From IEEE 802.11p to 5G,” *e & i Elektrotechnik und Informationstechnik*, vol. 132, no. 7, pp. 409–416, 2015.

Anexos

A. Archivo de configuración SUMO

Este archivo de configuración corresponde al escenario de Las Rosas:

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- generated on 2024-12-21 18:47:48 by Eclipse SUMO sumo Version 1.20.0
-->

<configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/sumoConfiguration.xsd">

    <input>
        <net-file value="LasRosas.net.xml"/>
        <route-files
value="bus_routes.rou.xml,passenger_routes.rou.xml,trailer_routes.rou.xml,deli
very_routes.rou.xml,truck_routes.rou.xml"/>
        <additional-files value="LasRosas.poly.xml"/>
    </input>

    <output>
        <fcd-output value="vehicle_data.xml"/>
    </output>

    <time>
        <begin value="0"/>
        <end value="1000"/>
    </time>

</configuration>
```

Archivo de configuración del escenario autopista A2:

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- generated on 2024-12-28 21:09:01 by Eclipse SUMO sumo Version 1.20.0
-->

<configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/sumoConfiguration.xsd">

    <input>
        <net-file value="TramoA2.net.xml"/>
        <route-files
value="bus_routes.rou.xml,passenger_routes.rou.xml,trailer_routes.rou.xml,deli
very_routes.rou.xml,truck_routes.rou.xml"/>
        <additional-files value="TramoA2.poly.xml"/>
    </input>
```

```
<output>
  <fcd-output value="vehicle_data.xml"/>
</output>

<time>
  <begin value="0"/>
  <end value="1000"/>
</time>
```

```
</configuration>
```

Archivo de configuración del escenario Puerta de Alcalá:

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- generated on 2024-12-22 15:23:07 by Eclipse SUMO sumo Version 1.20.0
-->

<configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/sumoConfiguration.xsd">

  <input>
    <net-file value="PuertaAlcala.net.xml"/>
    <route-files
value="bus_routes.rou.xml,passenger_routes.rou.xml,trailer_routes.rou.xml,deli
very_routes.rou.xml,truck_routes.rou.xml"/>
    <additional-files value="PuertaAlcala.poly.xml"/>
  </input>

  <output>
    <fcd-output value="vehicle_data.xml"/>
  </output>

  <time>
    <begin value="0"/>
    <end value="1000"/>
  </time>

</configuration>
```

Archivo de configuración del escenario Sanchinarro:

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- generated on 2024-12-12 16:27:56 by Eclipse SUMO sumo Version 1.20.0
-->

<configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/sumoConfiguration.xsd">
```

```
<input>
  <net-file value="SanchinarroCI.net.xml"/>
  <route-files
value="bus_routes.rou.xml,truck_routes.rou.xml,delivery_routes.rou.xml,trailer
_routes.rou.xml,passenger_routes.rou.xml"/>
  <additional-files value="SanchinarroCI.poly.xml"/>
</input>

<time>
  <begin value="0"/>
  <end value="1000"/>
</time>

</configuration>
```

B. Código para conseguir información de los vehículos (Conseguir_información_vehiculos.py)

```
import traci
import xml.etree.ElementTree as ET
from math import sqrt

# Inicia conexión con SUMO
sumoBinary = "sumo-gui"
sumoCmd = [sumoBinary, "-c",
"C:/Users/dlina/Documents/UNIVERSIDAD/TFG/SUMO/LasRosas/LasRosas.sumo.cfg"]

# Función para encontrar vehículos cercanos
def find_nearby_vehicles(detector_positions, max_distance=50):
    nearby_vehicles = {}
    for det_id, det_position in detector_positions.items():
        nearby_vehicles[det_id] = []
        for veh_id in traci.vehicle.getIDList():
            veh_position = traci.vehicle.getPosition(veh_id)
            distance = sqrt((veh_position[0] - det_position[0]) ** 2 +
(veh_position[1] - det_position[1]) ** 2)
            if distance <= max_distance:
                nearby_vehicles[det_id].append({
                    "vehicle_id": veh_id,
                    "distance": round(distance),
                    "position_x": veh_position[0],
                    "position_y": veh_position[1],
                    "speed": round(traci.vehicle.getSpeed(veh_id)),
                    "lane": traci.vehicle.getLaneID(veh_id),
                    "route": traci.vehicle.getRoute(veh_id)
                })
    return nearby_vehicles

try:
    traci.start(sumoCmd)

    time_data = []

    # Define IDs de detectores y sus posiciones. Para cada escenario las
    posiciones son distintas
    detector_positions = {
        "e3_0": (1060.97, 1396.98),
        "e3_1": (1024.72, 1343.64),
        "e3_2": (1089.74, 1280.83),
        "e3_3": (1072.99, 1353.32),
        "e3_4": (1075.13, 851.28),
        "e3_5": (1139.26, 714.71),
        "e3_6": (1179.52, 812.72),
        "e3_7": (1123.73, 423.37),
        "e3_8": (1103.47, 362.35),
```

```

    "e3_9": (1170.15, 339.08),
    "e3_10": (1198.80, 378.34),
}

while traci.simulation.getMinExpectedNumber() > 0:
    traci.simulationStep()
    current_time = traci.simulation.getTime()
    time_step_data = {"time": current_time, "detectors": []}

    # Obtener vehículos cercanos
    nearby_vehicles = find_nearby_vehicles(detector_positions,
max_distance=50)

    for det_id, det_position in detector_positions.items():
        vehicles = traci.multientryexit.getLastStepVehicleIDs(det_id)

        detector_info = {
            "detector_id": det_id,
            "detected_vehicles": [],
            "nearby_vehicles": nearby_vehicles[det_id],
            "nearby_vehicle_count": len(nearby_vehicles[det_id])
        }

        # Vehículos dentro del detector
        for veh_id in vehicles:
            veh_position = traci.vehicle.getPosition(veh_id)
            distance_to_detector = sqrt((veh_position[0] -
det_position[0]) ** 2 + (veh_position[1] - det_position[1]) ** 2)
            detector_info["detected_vehicles"].append({
                "vehicle_id": veh_id,
                "distance_to_detector": round(distance_to_detector),
                "position_x": veh_position[0],
                "position_y": veh_position[1],
                "speed": round(traci.vehicle.getSpeed(veh_id)),
                "lane": traci.vehicle.getLaneID(veh_id),
                "route": traci.vehicle.getRoute(veh_id)
            })

            time_step_data["detectors"].append(detector_info)

        time_data.append(time_step_data)

    # Crear XML de salida
    root = ET.Element("simulation_data")
    for time_step in time_data:
        time_elem = ET.SubElement(root, "time_step",
time=str(time_step["time"]))

        for detector in time_step["detectors"]:

```

```
        detector_elem = ET.SubElement(time_elem, "detector",
id=detector["detector_id"])

        # Contar vehículos cercanos
        ET.SubElement(detector_elem, "nearby_vehicle_count").text =
str(detector["nearby_vehicle_count"])

        # Vehículos detectados
        for vehicle in detector["detected_vehicles"]:
            detected_elem = ET.SubElement(detector_elem,
"detected_vehicle", id=vehicle["vehicle_id"])
            ET.SubElement(detected_elem, "distance_to_detector").text =
f"{vehicle['distance_to_detector']:.2f}"
            ET.SubElement(detected_elem, "x").text =
f"{vehicle['position_x']:.2f}"
            ET.SubElement(detected_elem, "y").text =
f"{vehicle['position_y']:.2f}"
            ET.SubElement(detected_elem, "speed").text =
f"{vehicle['speed']:.2f}"
            ET.SubElement(detected_elem, "lane").text = vehicle["lane"]
            ET.SubElement(detected_elem, "route").text =
str(vehicle["route"])

        # Vehículos cercanos
        for nearby in detector["nearby_vehicles"]:
            nearby_elem = ET.SubElement(detector_elem, "nearby_vehicle",
id=nearby["vehicle_id"])
            ET.SubElement(nearby_elem, "distance").text =
f"{nearby['distance']:.2f}"
            ET.SubElement(nearby_elem, "x").text =
f"{nearby['position_x']:.2f}"
            ET.SubElement(nearby_elem, "y").text =
f"{nearby['position_y']:.2f}"
            ET.SubElement(nearby_elem, "speed").text =
f"{nearby['speed']:.2f}"
            ET.SubElement(nearby_elem, "lane").text = nearby["lane"]
            ET.SubElement(nearby_elem, "route").text =
str(nearby["route"])

        # Guardar XML
        output_path =
"C:/Users/dlina/Documents/UNIVERSIDAD/TFG/SUMO/LasRosas/detector_vehicle.xml"
        tree = ET.ElementTree(root)
        tree.write(output_path)
        print(f"Archivo guardado en {output_path}")

finally:
    traci.close()
```

C. Código para calcular la potencia de los vehículos (calculo_potencia_vehiculos_cercanos.py)

```
import math
import xml.etree.ElementTree as ET
import random
from shapely.geometry import LineString, Polygon

# Parámetros del canal
FREQUENCY = 5.9e9 # Frecuencia en Hz (5.9 GHz para V2V)
TRANSMIT_POWER = 20 # Potencia de transmisión fija en dBm
D0 = 1 # Distancia de referencia en metros
PL_D0 = 78.4 # Pérdida a la distancia de referencia en dB (urbano)
PATHLOSS_EXPONENT = 1.53 # Exponente de pérdida de trayectoria (urbano)
BANDWIDTH = 10e6 # 10 MHz
TEMPERATURE = 290 # Temperatura estándar K

# Función para calcular la pérdida de trayectoria
def calculate_pathloss(distance):
    if distance < D0:
        distance = D0
    pathloss = PL_D0 + 10 * PATHLOSS_EXPONENT * math.log10(distance / D0)
    return pathloss

# Cargar obstáculos desde el archivo .poly
def load_obstacles(file_path):
    tree = ET.parse(file_path)
    root = tree.getroot()
    obstacles = []
    for poly in root.findall("poly"):
        points_attr = poly.get("points")
        if points_attr:
            points = [tuple(map(float, point.split(","))) for point in
points_attr.strip().split()]
            obstacles.append(Polygon(points))
    return obstacles

# Calcular shadowing por obstáculos
def shadowing_loss(obstacles, start, end):
    line_of_sight = LineString([start, end])
    loss = sum(random.uniform(5, 10) for o in obstacles if
line_of_sight.intersects(o))
    return loss if loss > 0 else random.uniform(0, 5)

# Función para calcular el ruido
def calculate_noise(num_vehicles):
    k = 1.38e-23 # Constante de Boltzmann en J/K
    noise_power = k * TEMPERATURE * BANDWIDTH
    noise_dbm = 10 * math.log10(noise_power * 1000)
    vehicle_interference = min(10, num_vehicles * 0.5)
```

```
    return noise_dbm - vehicle_interference

# Procesar vehículos y realizar cálculos
def process_vehicles(detectors_file, output_file):
    obstacles = load_obstacles(poly_file)
    tree = ET.parse(detectors_file)
    root = tree.getroot()

    for time_step in root.findall("time_step"):
        for detector in time_step.findall("detector"):
            nearby_vehicle_count =
int(detector.find("nearby_vehicle_count").text)

            # Procesar vehículos detectados
            for vehicle in detector.findall("vehicle"):
                vehicle_position = (float(vehicle.find('x').text),
float(vehicle.find('y').text))
                distance = float(vehicle.find("distance_to_detector").text)

                # Cálculos
                pathloss = calculate_pathloss(distance)
                shadowing = shadowing_loss(obstacles, vehicle_position, (0,
0))

                noise_dbm = calculate_noise(nearby_vehicle_count)
                power_received = TRANSMIT_POWER - pathloss - shadowing
                snr = power_received - noise_dbm

                # Añadir resultados al XML
                ET.SubElement(vehicle, "Pt").text = f"{TRANSMIT_POWER:.2f}
dBm"
                ET.SubElement(vehicle, "Pr").text = f"{power_received:.2f}
dBm"
                ET.SubElement(vehicle, "Pnoise").text = f"{noise_dbm:.2f} dBm"
                ET.SubElement(vehicle, "SNR").text = f"{snr:.2f} dB"

            # Procesar vehículos cercanos
            for vehicle in detector.findall("nearby_vehicle"):
                vehicle_position = (float(vehicle.find('x').text),
float(vehicle.find('y').text))
                distance = float(vehicle.find("distance").text)

                # Cálculos
                pathloss = calculate_pathloss(distance)
                shadowing = shadowing_loss(obstacles, vehicle_position, (0,
0))

                noise_dbm = calculate_noise(nearby_vehicle_count)
                power_received = TRANSMIT_POWER - pathloss - shadowing
                snr = power_received - noise_dbm

                # Añadir resultados al XML
```

```
ET.SubElement(vehicle, "Pt").text = f"{TRANSMIT_POWER:.2f}
dBm"
ET.SubElement(vehicle, "Pr").text = f"{power_received:.2f}
dBm"
ET.SubElement(vehicle, "Pnoise").text = f"{noise_dbm:.2f} dBm"
ET.SubElement(vehicle, "SNR").text = f"{snr:.2f} dB"

# Guardar el XML modificado
tree.write(output_file)
print(f"Resultados guardados en: {output_file}")

# Ejecutar la simulación
detectors_file =
"C:/Users/dlina/Documents/UNIVERSIDAD/TFG/SUMO/Obtener_datos/LasRosas/Capa_Fis
ica/detector_vehicle.xml"
poly_file =
"C:/Users/dlina/Documents/UNIVERSIDAD/TFG/SUMO/Obtener_datos/LasRosas/Capa_Fis
ica/LasRosas.poly.xml"
output_file =
"C:/Users/dlina/Documents/UNIVERSIDAD/TFG/SUMO/Obtener_datos/LasRosas/Capa_Fis
ica/comm_res_emp_mod_10_Pnoise_LR_Sub.xml"

process_vehicles(detectors_file, output_file)
```

D. Código para generar el Dataset

```
import xml.etree.ElementTree as ET
import pandas as pd

# Lista de archivos XML y sus escenarios correspondientes
xml_files = [
    ("C:/Users/dlina/Documents/UNIVERSIDAD/TFG/SUMO/Modelo_ML/comm_results_empirical_model_10_Pnoise_CSPA_form.xml", "CSPA"), #CSPA -> Con Semaforos Puerta Alcala
    ("C:/Users/dlina/Documents/UNIVERSIDAD/TFG/SUMO/Modelo_ML/comm_res_emi_model_10_Pnoise_LR_Sub_form.xml", "LR"), # LR -> Las Rosas
    ("C:/Users/dlina/Documents/UNIVERSIDAD/TFG/SUMO/Modelo_ML/comm_results_empirical_model_10_Pnoise_SSMTPA_form.xml", "SSMTPA"), #SSMTPA -> Sin Semaforos Mas Trafico Puerta Alcala
    ("C:/Users/dlina/Documents/UNIVERSIDAD/TFG/SUMO/Modelo_ML/comm_results_empirical_model_10_Pnoise_SSS_form.xml", "SSS"), # SSS -> Sin Semaforos Sanchinarro
    ("C:/Users/dlina/Documents/UNIVERSIDAD/TFG/SUMO/Modelo_ML/comm_results_empirical_model_10_Pnoise_SSTNPA_form.xml", "SSTNPA"), # SSTNPA -> Sin Semaforos Traficio Normal Puerta Alcala
    ("C:/Users/dlina/Documents/UNIVERSIDAD/TFG/SUMO/Modelo_ML/comm_res_emi_model_10_Pnoise_A2_Hig_form.xml", "A2") # A2 -> Autopista A-2
]

# Umbral para determinar la duración de la sesión
SNR_THRESHOLD = 15 # dB
link_SNR = 5 # dB
THRESHOLD = SNR_THRESHOLD - link_SNR

# Función para procesar cada archivo XML
def process_xml(file_info):
    file_path, scenario = file_info
    tree = ET.parse(file_path)
    root = tree.getroot()

    data = []

    for time_step in root.findall("time_step"):
        time = float(time_step.attrib.get("time", 0))
        for detector in time_step.findall("detector"):
            detector_id = detector.attrib.get("id", "unknown")
            nearby_vehicle_count =
int(detector.find("nearby_vehicle_count").text) if
detector.find("nearby_vehicle_count") is not None else 0

            for vehicle in detector.findall("nearby_vehicle"):
                vehicle_id = vehicle.attrib.get("id", "unknown")
                distance = float(vehicle.find("distance").text) if
vehicle.find("distance") is not None else None
```

```

        pos_x = float(vehicle.find("x").text) if vehicle.find("x") is
not None else None
        pos_y = float(vehicle.find("y").text) if vehicle.find("y") is
not None else None
        speed = float(vehicle.find("speed").text) if
vehicle.find("speed") is not None else None
        lane = vehicle.find("lane").text if vehicle.find("lane") is
not None else None
        route = vehicle.find("route").text if vehicle.find("route") is
not None else None
        Pt = float(vehicle.find("Pt").text.split()[0]) if
vehicle.find("Pt") is not None else None
        Pr = float(vehicle.find("Pr").text.split()[0]) if
vehicle.find("Pr") is not None else None
        Pnoise = float(vehicle.find("Pnoise").text.split()[0]) if
vehicle.find("Pnoise") is not None else None
        SNR = float(vehicle.find("SNR").text.split()[0]) if
vehicle.find("SNR") is not None else None

        data.append({
            "time": time,
            "detector_id": detector_id,
            "nearby_vehicle_count": nearby_vehicle_count,
            "vehicle_id": vehicle_id,
            "distance": distance,
            "pos_x": pos_x,
            "pos_y": pos_y,
            "speed": speed,
            "lane": lane,
            "route": route,
            "Pt": Pt,
            "Pr": Pr,
            "Pnoise": Pnoise,
            "SNR": SNR,
            "scenario": scenario # Añadimos el escenario
correspondiente
        })

    return pd.DataFrame(data)

# Procesar todos los archivos XML
all_data = pd.concat([process_xml(file_info) for file_info in xml_files],
ignore_index=True)

# Ordenar los datos por detector_id, vehicle_id y escenario
all_data.sort_values(by=["scenario", "detector_id", "vehicle_id", "time"],
inplace=True)

# Calcular session_duration basado en SNR_THRESHOLD
def calculate_session_duration(group):
    group = group.sort_values("time")

```

```
    group["SNR_above_threshold"] = group["SNR"] > THRESHOLD
    session_duration = group.loc[group["SNR_above_threshold"], "time"].max() -
group.loc[group["SNR_above_threshold"], "time"].min() if
any(group["SNR_above_threshold"]) else 0
    group["session_duration"] = session_duration
    return group.drop(columns=["SNR_above_threshold"])

# Aplicar el cálculo de duración de la sesión por vehículo y detector
all_data = all_data.groupby(["scenario", "detector_id", "vehicle_id"],
group_keys=False).apply(calculate_session_duration)

# Guardar el dataset consolidado con la nueva columna session_duration
output_file =
"C:/Users/dlina/Documents/UNIVERSIDAD/TFG/SUMO/Modelo_ML/dataset_vehicle_sesDu
rat.csv"
all_data.to_csv(output_file, index=False)

print(f"Dataset generado y guardado en: {output_file}")
```

E. Código del modelo de Regresión Lineal Múltiple

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import matplotlib.pyplot as plt

csv_file_path =
"C:/Users/dlina/Documents/UNIVERSIDAD/TFG/SUMO/Modelo_ML/Modelos_ML/dataset_ve
hicle_sesDurat.csv"
data = pd.read_csv(csv_file_path)

# Selección de variables relevantes
selected_features = [
    "nearby_vehicle_count", "distance", "pos_x", "pos_y", "speed", "Pt", "Pr",
    "Pnoise", "SNR"
]
target_variable = "session_duration"
X = data[selected_features]
y = data[target_variable]

# Dividir en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Normalizar los datos
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

X_train_scaled.shape, X_test_scaled.shape, y_train.shape, y_test.shape

# Modelo de Regresión Lineal Simple con "SNR"
model = LinearRegression()
model.fit(X_train_scaled, y_train)
y_pred = model.predict(X_test_scaled)

# Evaluación del modelo
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Resultados del modelo
print("Resultados del Modelo de Regresión Lineal Multiple
(session_duration):")
print(f"MSE: {mse}")
print(f"MAE: {mae}")
print(f"R2: {r2}")
```

```
# Visualizar los resultados
plt.scatter(range(len(y_test)), y_test, color='red', label='Real')
plt.scatter(range(len(y_pred)), y_pred, color='blue', label='Predicción')
plt.xlabel('Índice de muestra')
plt.ylabel('Duración de sesión')
plt.legend()
plt.title('Comparación Real vs Predicción - Regresión Lineal Multiple')
plt.show()
```

F. Código del modelo de SVR

```

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import matplotlib.pyplot as plt

# Cargar el archivo CSV
csv_file_path =
"C:/Users/dlina/Documents/UNIVERSIDAD/TFG/SUMO/Modelo_ML/Modelos_ML/dataset_ve
hicle_sesDurat.csv"
data = pd.read_csv(csv_file_path)

# Selección de variables relevantes
selected_features = [
    "nearby_vehicle_count", "distance", "pos_x", "pos_y", "speed", "Pt", "Pr",
    "Pnoise", "SNR"
]
target_variable = "session_duration"

# Verificar si las columnas existen
for feature in selected_features:
    if feature not in data.columns:
        print(f"La columna '{feature}' no está en el dataset.")

X = data[selected_features]
y = data[target_variable]

# Codificación de variables categóricas
if X.select_dtypes(include=['object']).shape[1] > 0:
    for col in X.select_dtypes(include=['object']).columns:
        le = LabelEncoder()
        X[col] = le.fit_transform(X[col].astype(str))

# División en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Normalización de las características
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Modelo SVM
svm_model = SVR(kernel='rbf', C=1.0, epsilon=0.1)
svm_model.fit(X_train_scaled, y_train)

# Predicciones

```

```
y_pred = svm_model.predict(X_test_scaled)

# Evaluación del modelo
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Resultados del modelo SVR:")
print(f"MSE: {mse}")
print(f"MAE: {mae}")
print(f"R2: {r2}")

# Visualización de Resultados
plt.scatter(range(len(y_test)), y_test, color='red', label='Real')
plt.scatter(range(len(y_pred)), y_pred, color='blue', label='Predicción')
plt.xlabel('Índice de muestra')
plt.ylabel('Duración de sesión')
plt.legend()
plt.title('Comparación Real vs Predicción - SVR')
plt.show()
```

G. Código del modelo de Árbol de Decisión

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import matplotlib.pyplot as plt

# Cargar el archivo CSV
csv_file_path =
"C:/Users/dlina/Documents/UNIVERSIDAD/TFG/SUMO/Modelo_ML/Modelos_ML/dataset_ve
hicle_sesDurat.csv"
data = pd.read_csv(csv_file_path)

# Verificar la existencia de la columna 'session_duration'
if 'session_duration' not in data.columns:
    raise ValueError("La columna 'session_duration' no está presente en el
dataset.")

# Selección de variables relevantes
selected_features = [
    "nearby_vehicle_count", "distance", "pos_x", "pos_y", "speed", "Pt", "Pr",
    "Pnoise", "SNR"
]
target_variable = "session_duration"
X = data[selected_features]
y = data[target_variable]

# Codificación de variables categóricas
for col in X.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    X[col] = le.fit_transform(X[col].astype(str))

# División en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Normalización de las características
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Modelo de Árbol de Decisión
tree_model = DecisionTreeRegressor(random_state=42)
tree_model.fit(X_train_scaled, y_train)

# Predicciones
y_pred = tree_model.predict(X_test_scaled)
```

```
# Evaluación del modelo
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Resultados
print("Resultados del modelo de Árbol de Decisión:")
print(f"MSE: {mse}")
print(f"MAE: {mae}")
print(f"R2: {r2}")

# Visualización de Resultados
plt.scatter(range(len(y_test)), y_test, color='red', label='Real')
plt.scatter(range(len(y_pred)), y_pred, color='blue', label='Predicción')
plt.xlabel('Índice de muestra')
plt.ylabel('Duración de sesión')
plt.legend()
plt.title('Comparación Real vs Predicción - Decision Tree')
plt.show()
```

H. Código del modelo de Bosques Aleatorios

```

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import matplotlib.pyplot as plt

# Cargar el archivo CSV
csv_file_path =
"C:/Users/dlina/Documents/UNIVERSIDAD/TFG/SUMO/Modelo_ML/Modelos_ML/dataset_ve
hicle_sesDurat.csv"
data = pd.read_csv(csv_file_path)

# Selección de variables relevantes
selected_features = [
    "nearby_vehicle_count", "distance", "pos_x", "pos_y", "speed", "Pt", "Pr",
    "Pnoise", "SNR"
]
target_variable = "session_duration"
X = data[selected_features]
y = data[target_variable]

# Codificación de variables categóricas
for col in X.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    X[col] = le.fit_transform(X[col].astype(str))

# División en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Normalización de las características
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Modelo de Bosques Aleatorios
rf_model = RandomForestRegressor(n_estimators=100, max_depth=None,
random_state=42)
rf_model.fit(X_train_scaled, y_train)

# Predicciones
y_pred = rf_model.predict(X_test_scaled)

# Evaluación del modelo
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

```

```
# Resultados
print("Resultados del modelo de Bosques Aleatorios:")
print(f"MSE: {mse}")
print(f"MAE: {mae}")
print(f"R2: {r2}")

# Visualización de Resultados
plt.scatter(range(len(y_test)), y_test, color='red', label='Real')
plt.scatter(range(len(y_pred)), y_pred, color='blue', label='Predicción')
plt.xlabel('Índice de muestra')
plt.ylabel('Duración de sesión')
plt.legend()
plt.title('Comparación Real vs Predicción - Random Forest')
plt.show()
```

I. Código del modelo de ANN

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from tensorflow.keras.models import Sequential # type: ignore
from tensorflow.keras.layers import Dense, Dropout # type: ignore

# Cargar el archivo CSV
file_path =
"C:/Users/dlina/Documents/UNIVERSIDAD/TFG/SUMO/Modelo_ML/Modelos_ML/dataset_ve
hicle_sesDurat.csv"
data = pd.read_csv(file_path, on_bad_lines='skip', quoting=3)

# Eliminar filas con valores nulos
data = data.dropna()

# Seleccionar características y variable objetivo
X = data.drop(columns=["session_duration", "time", "vehicle_id",
"detector_id", "scenario"])
y = data["session_duration"]

# Codificación de variables categóricas
for col in X.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    X[col] = le.fit_transform(X[col].astype(str))

# Dividir el dataset en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Normalizar los datos
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Construcción del modelo ANN
model = Sequential([
    Dense(128, input_dim=X_train_scaled.shape[1], activation='relu'), # Capa
oculta 1
    Dropout(0.3), # Dropout para evitar sobreajuste
    Dense(64, activation='relu'), # Capa oculta 2
    Dropout(0.3), # Dropout adicional
    Dense(32, activation='relu'), # Capa oculta 3
    Dense(1, activation='linear') # Capa de salida (regresión)
])
```

```
# Compilar el modelo
model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])

# Entrenamiento del modelo
history = model.fit(X_train_scaled, y_train, epochs=150, batch_size=32,
validation_split=0.2, verbose=1)

# Predicción
y_pred = model.predict(X_test_scaled).flatten()

# Eliminar posibles NaN antes de la evaluación
valid_indices = ~np.isnan(y_pred) & ~np.isnan(y_test)
y_test_clean = y_test[valid_indices]
y_pred_clean = y_pred[valid_indices]

# Evaluación del modelo
mse = mean_squared_error(y_test_clean, y_pred_clean)
mae = mean_absolute_error(y_test_clean, y_pred_clean)
r2 = r2_score(y_test_clean, y_pred_clean)

print("Resultados del modelo de Red Neuronal Artificial (ANN):")
print(f"MSE: {mse}")
print(f"MAE: {mae}")
print(f"R2: {r2}")

# Visualización de Resultados
plt.figure(figsize=(10, 6))
plt.scatter(range(len(y_test_clean)), y_test_clean, color='red', label='Real',
alpha=0.6)
plt.scatter(range(len(y_pred_clean)), y_pred_clean, color='blue',
label='Predicción', alpha=0.6)
plt.xlabel('Índice de muestra')
plt.ylabel('Duración de sesión')
plt.legend()
plt.title('Comparación Real vs Predicción - ANN')
plt.grid(True)
plt.show()
```

J. Dataset

```

960.0,e3_5,4,738,38.0,833.43,1338.81,10.0,:1074577474_14_0,('5930467#1',
'419119406#1', '-651543114#1', '804457302', '46098085#0', '46098085#1',
'4346299#0', '4346299#1', '4346299#4', '4346299#5', '321307747', '242547871',
'131431688#0', '4363468#0', '554834776', '112538796#0', '112538796#2',
'112538796#3')",20.0,-82.04,-105.98,23.94,SSTNPA,8.0
961.0,e3_5,4,738,31.0,838.19,1346.1,8.0,4363468#0_1,('5930467#1',
'419119406#1', '-651543114#1', '804457302', '46098085#0', '46098085#1',
'4346299#0', '4346299#1', '4346299#4', '4346299#5', '321307747', '242547871',
'131431688#0', '4363468#0', '554834776', '112538796#0', '112538796#2',
'112538796#3')",20.0,-79.63,-105.98,26.35,SSTNPA,8.0
962.0,e3_5,4,738,22.0,839.26,1355.93,10.0,:435729007_6_1,('5930467#1',
'419119406#1', '-651543114#1', '804457302', '46098085#0', '46098085#1',
'4346299#0', '4346299#1', '4346299#4', '4346299#5', '321307747', '242547871',
'131431688#0', '4363468#0', '554834776', '112538796#0', '112538796#2',
'112538796#3')",20.0,-81.24,-105.98,24.74,SSTNPA,8.0
963.0,e3_5,5,738,10.0,837.15,1367.62,12.0,:435729007_6_1,('5930467#1',
'419119406#1', '-651543114#1', '804457302', '46098085#0', '46098085#1',
'4346299#0', '4346299#1', '4346299#4', '4346299#5', '321307747', '242547871',
'131431688#0', '4363468#0', '554834776', '112538796#0', '112538796#2',
'112538796#3')",20.0,-72.19,-106.48,34.29,SSTNPA,8.0
964.0,e3_5,5,738,7.0,838.79,1380.49,13.0,:3323813668_2_2,('5930467#1',
'419119406#1', '-651543114#1', '804457302', '46098085#0', '46098085#1',
'4346299#0', '4346299#1', '4346299#4', '4346299#5', '321307747', '242547871',
'131431688#0', '4363468#0', '554834776', '112538796#0', '112538796#2',
'112538796#3')",20.0,-73.0,-106.48,33.48,SSTNPA,8.0

```

Se muestra un pequeño ejemplo del dataset que contiene los campos que se han empleado para las simulaciones de los modelos. El nuevo dataset consta de las siguientes columnas:

- *Time*: muestra el segundo de la simulación cuando fue detectado.
- *Detector_id*: muestra el nombre del detector.
- *Nearby_vehicle_count*: muestra los vehículos que se encuentran en un radio de 50 metros del vehículo detectado.
- *Vehicle_id*: muestra el nombre del vehículo detectado.
- *Disatnce*: muestra la distancia del vehículo detectado al detector.
- *Pos_x* y *Pos_y*: muestra la posición del vehículo en el mapa.
- *Speed*: muestra la velocidad del vehículo en m/s.
- *Lane*: muestra el carril por el que circula el vehículo detectado.
- *Route*: muestra la ruta que va a realizar el vehículo detectado.
- *Pt*: muestra la potencia de transmisión de cada vehículo (20 dBm).
- *Pr*: muestra la potencia de recepción del vehículo detectado.
- *Pniose*: muestra la potencia de ruido del vehículo detectado.
- *SNR*: muestra la relación señal a ruido del vehículo detectado.
- *Escenario*: muestra el escenario del vehículo.
- *Session_duration*: muestra el tiempo que dura la sesión de comunicación.

